计算机的概要

硬件概要

软件概要

计算机的概要

个人电脑

个人电脑（英语：Personal Computer，简称：PC），普遍称为电脑，又称为个人计算机，是在大小、性能以及价位等多个方面适合于个人使用，并由最终用户直接操控的计算机的统称。它与批处理计算机或分时系统等一般同时由多人操控的大型计算机相对。从台式电脑（或称桌面电脑）、笔记本电脑到平板电脑等都属于个人电脑的范畴。

定义

一般来说个人电脑分为两大机型与两大系统，在机型上分为常见的桌面型电脑与笔记本电脑。在系统上分别是国际商用机器公司（IBM）集成制定的IBM PC/AT系统标准，以及苹果电脑所开发的麦金塔系统。狭义来说，个人电脑是指前者（IBM集成制定的PC/AT），IBM PC/AT标准由于采用x86开放式架构而获得大部分厂商所支持，成为市场上主流，因此一般所说的PC意指IBM PC/AT兼容机种，此架构中的中央处理器采用Intel或AMD等厂商所生产的中央处理器。而桌面型电脑因采用开放式硬件架构，所以除了品牌外，自行组装的无品牌电脑也是极度盛行。

1962年11月3日《纽约时报》在相关报道中首次使用“个人电脑”一词，逮至1968年时HP公司即把其产品Hewlett-Packard 9100A称为“个人电脑”。世界公认第一部个人电脑，则为1971年Kenbak Corporation推出的Kenbak-1。第一部大量生产的个人电脑，则为1971年Computer Terminal Corporation所推出的Datapoint 2200。至1973年，法国工程师François Gernelle和André Truong所发明的Micral个人电脑，则为首部使用Intel微处理器的商用个人电脑。

东芝采用x86架构开发出世界第一部笔记本电脑。

1980年代，IBM推出以英特尔的x86的硬件架构及微软公司的MS-DOS操作系统的个人电脑，并制定以PC/AT为PC的规格。之后由英特尔所推出的微处理器以及微软所推出的操作系统发展几乎等同于个人电脑的发展历史。Wintel架构全面取代了IBM在个人电脑主导的地位。

分类

按照大小和移动性分类：

台式机，也称桌面计算机、台式电脑和台式机

工作站，高端的个人电脑，主要由单人操作并一般与局域网相连。

可移动计算机，与笔记本电脑相近

笔记本电脑，小型、可以方便携带的个人电脑

上网本，更为小型、售价较低的笔记本电脑

平板式电脑，笔记本电脑的派生型，输入设备由鼠标和键盘变成了触控笔

个人数码助理，PDA，也称个人数码助理

可穿戴式电脑，概念型产品，未具市场规模与趋势，美国VA公司曾于1999年主推过，但如今不再强调。

硬件

对于台式电脑而言，在具备了一定的知识与经验之后，普通人也可以较为容易地将各个电脑部件组装起来，组成一部完整的电脑。

由于个人电脑产业发展迅速，每三至五年就会有一次质的飞跃，因此对电脑硬件的升级有时是必要的。例如，一部五、六年前被认为是高端配置个人电脑，在今天看来，其中央处理器、内存和周边设备的速度、容量等都已落后，已很难顺畅运行今天的软件。而市场上主流操作系统软件（例如微软公司的Windows）的每一次升级在带来新的便捷的同时，也对个人电脑的硬件系统提出新的要求，促进了个人电脑硬件的更新换代。通常而言，存储器和一些外围设备（比如显示卡和磁盘存储器）是容易升级的，最终用户甚至可以自己动手。为了升级计算机的处理能力等，有时甚至可以更换处理器甚至主板。

个人电脑的硬件功能可以通过插到主板扩展槽的附加扩张界面卡来扩充。到2005年主要的标准系统总线有PCI-E、PCI和AGP。个人电脑也可以通过添加额外的驱动器来升级（如光驱、硬盘、USB驱动器等），标准的存储设备接口有ATA、SATA／SATA-2以及SCSI。

机箱

计算机的许多硬件，如主板、硬盘和电源等，都会被安放与固定在机箱当中。机箱是一个相对封闭的空间，箱体一般由钢和铝等金属制成（其他材料亦可用，但不多见），同时设有许多通风口，以促进箱内空气流动，防止内部温度过高。机箱的颜色、大小乃至形状等可根据内部硬件之实际需要以及个人的喜好而变。

目前市场比较普遍的机箱类型有AT、ATX、Micro ATX三种。

主板

主板是计算机的主要电路板（PCB）。计算机的其他硬件一般直接插入到主板中来交换信息。主板通常由芯片组、BIOS、CMOS、并口（也称：印表端口）、PS/2键盘和鼠标接口和扩展插槽组成。有时为了减小主板尺寸，节省成本也通过插到主板的子板来扩充更多扩展槽。现时主流的主板规格有ATX、MicroATX。而未来的主流规格会是BTX（Balanced Technology Extended）。

picoBTX：主板最长203.20毫米，最多一个扩充卡插槽。

microBTX：主板最长264.16毫米，最多四个扩充卡插槽。

BTX：主板最长325.12毫米，最多七个扩充卡插槽。

中央处理器

中央处理器（central processing unit，缩写CPU），又称中央处理单元，简称处理器，是整个计算机系统中最重要的部件之一，控制整个电脑主要的算术逻辑单元（arithmetic logic unit，简称ALU），使得电脑程序和操作系统可在它上面运行，其性能直接影响计算机系统的工作效率。不同类型的CPU安装到主板上使用不同类型的插槽中（主流的插槽有Socket 775和Socket AM2。当中Intel的处理器使用Socket775以及LGA1366，AMD的则使用Socket AM2及AM3）。市面上的主流CPU都是采用45纳米或65纳米制程核心，但到了2009年下半年后便踏进了32纳米制程时代，中央处理器变得更省电，温度更低。

大多数IBM PC兼容机使用x86-架构的处理机，他们主要由英特尔（Intel）和超微（AMD）两家公司生产，此外威盛电子（VIA）或者全美达（Transmeta）也有参与CPU的生产。与IBM PC兼容机不同，在2006年之前苹果电脑所使用的处理器一直是IBM PowerPC RISC，之后的苹果电脑开始转而采用Intel的处理器以及系统。

内存

主存储器，或称随机存取存储器（RAM），通称内存，属于内部存储器的一种，是计算机的“短期临时存储器”，用于存放当前正在运行的程序以及目前所需的数据等。它的读写速度要远远高于像硬盘驱动器或者光盘驱动器这些大容量存储设备，但是当系统关闭或没有电源供应的时候它的存储内容就会丢失。在个人电脑中，随机存取存储器分为DDR、DDR2及DDR3，而显卡上则还有GDDR4及GDDR5。而随机存取存储器随着系统支持，可利用双通道或者三通道技术让带宽变的更宽。内存的大小与性能是影响整个计算机系统运行效率的重要因素之一。

除了随机存取存储器外，广义上而言，主板上的只读存储器（ROM）也属内存之一，其中存放的数据在断电情况下不会丢失。但它主要用于保存系统的引导程序和自检程序等，并且容量很小，故此俗称中的“内存”一般不将其包括在内。

电源

电源供应器属外设之一，作用为将家用交流电源转换为个人电脑硬件所需的直流电。电源供应器的功率应满足电脑各部件的需要，否则或会影响到整个系统的性能。许多旧式个人电脑出现的问题都归咎于老化或有缺陷的电源供应器运作时会出现过热现象或无法提供稳定电流和电压等。

显卡

显卡，或称显卡，是电脑显示图像的重要器件，它负责处理CPU传送来的图像指令和数据，并将处理后的结果输出至显示器，以最终形成图像。随时代发展显卡芯片的运作性能亦随之取得了长足的提升。显卡可以分为独立显卡以及集成在主板芯片组上的集成显卡两类。一般而言，同一时期的独立显卡在性能上明显优于集成显卡，同时在功耗及发热量上则亦大于集成显卡。

目前主要的显示芯片（GPU）生产厂商包括NVIDIA和ATI等。

显示器

又称监视器、屏幕等，是电脑的图像显示设备，一般独立于机箱。被广泛应用于电脑上的显示器主要有CRT显示器和平板显示器（如液晶显示器）。

键盘

键盘是电脑的重要输入设备之一，是向电脑输入文本及其他数据的首要方式。现代个人电脑标准键盘一般有107个按键，多沿用打字机所采用的QWERTY布局，只是新增了功能键、方向键等电脑所需的键，有的键盘还设有一些额外的功能键。键盘的每个键上均有标明其所对应的字母、数字或功能。在用键盘向计算机输入数据时，通常一次只需按一个键，但也可能需要同时按下多个键，亦即组合键。各个按键所对应的功能不是固定的，许多程序都会对键盘各键的功能重新定义，实际操作时需重新记忆。

鼠标

鼠标是一个小巧、可滑动的设备，也是当代电脑重要和十分常见的输入设备之一。在图形用户界面上，用户通过滑动和点击鼠标等操作可向电脑输入命令和数据。鼠标可通过USB和PS/2接口与电脑相连，新式的鼠标还可以通过无线USB及蓝牙连接。

早期的鼠标（及部分苹果电脑鼠标）只有一个按键，而现在则有两或三个以上，其中或包括一个滑轮，可以运行的操作包括移动、单击（包括右击）、双击、拖曳等。

外部存储媒体

硬盘

硬盘是用于存储数据的最重要和传统的部件之一。与内存不同，其中的数据在断电之后不会丢失。硬盘在使用时会经由马达带动一个坚硬的盘片飞速旋转，以此写入和读取数据。随着科技进步，马达转速愈来愈高，以IDE（一种数据接口）与SATA的现时主力皆为7200转/分的硬盘，但SCSI的已经达到10000转/分。随着垂直录写技术的发展，硬盘容量亦愈来愈大，现时3.5吋的硬盘最大容量已经来到2TB。硬盘依大小可分为3.5吋（台式机主流）、2.5吋（笔记本电脑主流）、1.8吋（UMPC等小型电脑使用）。

软式磁盘驱动器

软式磁盘驱动器，简称软驱，是以往个人电脑的重要设备之一，可插入可移动式的软盘。2000年前由于价格便宜，并且为当时系统恢复及升级BIOS时必备的工具，因而在大多数电脑上都有装备。然而软盘容量极小，稳定性不佳，现时已几乎被闪存所制成的闪存盘取代。在新配置的个人电脑上已罕见软驱的身影。

光盘驱动器

光盘驱动器，台湾作光碟机，是电脑、电子游戏机用来读写光盘内容的机器。

其他存储媒体

闪存盘

闪存盘（又称优盘、U盘、电子盘、随身碟、记忆棒、手指），是一种用闪存来进行数据存储的介质，通常使用USB插头。通常闪存盘体积极小、重量轻、可热插拔也可以重复写入。面世后迅速普及并取代传统的软盘及软盘驱动器。在2006年11月时，市面上贩售的闪存盘的存储容量介于32MB到64GB之间[1]。容量大小因为闪存目前的密度而被限制，容量越大，则每MB的成本可能因为较贵的的组件而提高。有时读卡器也会被归类为闪存盘。这类设备的记忆芯片并不是自带的，而是可以抽换的存储卡。

相较于其他可携式存储设备（尤其是软盘片），闪存盘有许多优点：较不占空间，通常操作速度较快（USB1.1标准），能存储较多数据，并且可能较可靠（由于没有机械设备），在读写时断开而不会损坏硬件（软盘在读写时断开马上损坏），只会丢失数据。这类的磁盘使用USB大量存储设备标准，在近代的操作系统如Linux、Mac OS X、Unix与Windows中皆有自带支持。

闪存盘通常使用塑胶或金属外壳，内部含有一张小的印刷电路板，让闪存盘尺寸小到像钥匙圈饰物一样能够放到口袋中，或是串在颈绳上。只有USB连接头突出于保护壳外，且通常被一个小盖子盖住。大多数的闪存盘使用标准的Type-A USB接头，这使得它们可以直接插入个人电脑上的USB端口中。

要访问闪存盘的数据，就必须把闪存盘连接到电脑；无论是直接连接到电脑自带的USB控制器或是一个USB集线器都可以。只有当被插入USB端口时，闪存盘才会启动，而所需的电力也由USB连接供给。然而，有些闪存盘（尤其是使用USB 2.0标准的高速闪存盘）可能需要比较多的电源，因此若接在像是自带在键盘或屏幕的USB集线器，这些闪存盘将无法工作，除非将它们直接插到控制器（也就是电脑本身提供的USB端口）或是一个外接电源的USB集线器上。

磁光盘

磁光盘（magneto-optical disc, MO disc），它的读取方式是基于克尔效应（Kerr Effect）。磁光盘由对温度敏感的磁性材料制成。Sony推出的用于音乐的MiniDisc，实际上就是一种小型化的磁光盘。

3.5" MO常见的容量

128MB: 512Bytes/Sector

230MB: 512Bytes/Sector

540MB: 512Bytes/Sector

650MB: 2048Bytes/Sector

1.3GB（称为“GIGAMO”）: 2048Bytes/Sector

2.3GB

其他

通信相关硬件

网卡

网卡, 网络适配器或NIC (网络接口控制器)是一块被设计用来允许计算机在计算机网络上进行通讯的计算机硬件。由于其拥有MAC地址，因此属于OSI模型的第2层。它使得用户可以透过电缆或无线相互连接。 每一个网卡都有一个被称为MAC地址的独一无二的48位串行号，它被写在卡上的一块ROM中。在网络上的每一个计算机都必须拥有一个独一无二的MAC地址。没有任何两块被生产出来的网卡拥有同样的地址。这是因为电气电子工程师协会（IEEE）负责为网络接口控制器销售商分配唯一的MAC地址。

网卡以前是作为扩展卡插到计算机总线上的，但是由于其价格低廉而且以太网标准普遍存在，大部分新的计算机都在主板上集成了网络接口。这些主板或是在主板芯片中集成了以太网的功能，或是使用一块通过PCI (或者更新的PCI-Express总线)连接到主板上的廉价网卡。除非需要多接口或者使用其它种类的网络，否则不再需要一块独立的网卡。甚至更新的主板可能含有内置的双网络（以太网）接口。

调制解调器

调制解调器（英语：Modem，这个命名源自 modulator-demodulator），它是一个将数字信号调制到模拟载波信号上进行传输，并解调收到的模拟信号以得到数字信息的电子设备。 它的目标是产生能够方便传输的模拟信号并且能够通过解码还原原来的数字数据。根据不同的应用场合，调制解调器可以使用不同的手段来传送模拟信号, 比如使用光纤，射频无线电或电话线等。

使用普通电话线音频波段进行数据通信的电话调制解调器是人们最常接触到的调制解调器。在口语中，根据英语语音取首音节，很多人将电话调制解调器称为“猫”。

其他常见的调制解调器还包括用于宽带数据接入的有线电视电缆调制解调器，DSL调制解调器。 数字式移动电话实际上也是一种无线方式的调制解调器。现代电信传输设备是为了在不同的介质上远距离的传输大量信息，因此也都以调制解调器的功能为核心。其中，微波调制解调器速率可以达上百万比特每秒；而使用光纤作为传输介质的光调制解调器可以达到几十Gbps以上，是现在电信传输手段的骨干。

其他硬件

麦克风

麦克风（又称微音器或话筒，正式的中文名是传声器），译自英文Microphone，是一种将声音转换成电子信号的换能器。

耳机或喇叭

耳机（又称耳筒、听筒），是一对转换单元，它接受媒体播放器或接收器所发出的电讯号，利用贴近耳朵的扬声器将其转化成可以听到的音波。耳机一般是与媒体播放器可分离的，利用一个插头连接。好处是在不影响旁人的情况下，可独自聆听音响；亦可隔开周围环境的声响，对在录音室、DJ等在噪吵环境下工作的人很有帮助。

耳机原是给电话和无线电上使用的，但随着可携式电子装置的盛行，耳机多用于手机、随身听、收音机。可携式电玩和数码音乐播放器等。亦同时见用于电脑和Hi-fi音响之中。

摄像头

摄像头（webcam）一般具有视讯摄影/传播和静态图像捕捉等基本功能，它是借由镜头采集图像后，由摄像头内的感光元件电路及控制元件对图像进行处理并转换成电脑所能识别的数位讯号，然后借由Parallel或USB连接输入到电脑后由软件再进行图像还原。

目前市面上摄像头分为两种, 一种为直接连接电脑可用于视讯通话的消费型摄像头(Webcam),另一种为保全监控专用的网络监控摄影机(IP Camera/Network Camera)。

打印机

打印机（Printer）或称作列印机，是一种电脑输出设备，可以电脑内储存的数据按照文字或图形的方式永久的输出到纸张或者透明胶片上。

扫瞄仪

扫描仪是一个能够把相片、印刷文件或手写文件等图像，或装饰品等小对象扫描、分析并化成數碼图像的器材。

今日，大部份扫描仪都是桌面扫描仪的变种。以手动作扫描的手提扫描仪曾一度流行，但由于很难获得理想图像质素，现已被淘汰。这些扫描仪都使用电荷耦合器（Charge-coupled device，CCD）作为图像感应器，而鼓式扫描仪则采用光电倍增管（Photomultiplier Tube）作为图像感应器。扫描仪在我们日常生活中十分重要，随处都有它的存在。

其他种类的扫描仪包括planetary扫描仪，可为书籍及文件拍摄图像，及激光距离扫描仪（Laser Range scanner），可制作立体三维空间对象的模型。

软件

操作系统

操作系统用于管理计算机资源，并可让用户通过各种程序使用这些资源。目前PC市场上占有率最高的是微软所推出的Windows系列，其次则是类Unix系列的操作系统，例如有开放式架构且逐渐窜起的GNU/Linux，以及FreeBSD等，此外尚有Solaris，OS/2及eComStation等。

需要注意的是，Mac OS并不属于个人电脑操作系统，但仍可用于个人电脑。

微软Windows

微软Windows是指微软公司所推出的一系列操作系统。1985年11月微软推出了一个名为“Windows”的拥有图形用户界面（GUI）的作业环境，这之后，又相继推出了不同版本的Windows。目前最新的Windows操作系统是Windows 7和Windows Server 2008 R2

Mac OS X

Mac OS X是苹果公司研发和推出的一系列操作系统，是原Mac OS的继承者。但与后者不同的是，它是基于Unix的图形操作系统。目前最新版本的Mac OS X是Mac OS X 10.6“Snow Leopard”最新的服务器版本是Mac OS X Server 10.6。

Linux

Linux，是一类Unix计算机操作系统的统称。该操作系统的核心的名字也是“Linux”。Linux操作系统也是自由软件和开放源代码发展中最著名的例子。

严格来讲，Linux这个词本身只表示Linux核心，但在实际上人们已经习惯了用Linux来形容整个基于Linux核心，并且使用GNU工程各种工具和数据库的操作系统（也被称为GNU/Linux）。基于这些组件的Linux软件被称为Linux发行版。一般来讲，一个Linux发行包包含大量的软件，比如软件开发工具、数据库（例如PostgreSQL、MySQL）、网络服务器（例如Apache）、X Window、桌面环境（例如GNOME和KDE）、办公包（例如OpenOffice.org）、脚本语言（例如Perl、PHP和Python）等等。

Linux核心最初是为英特尔386微处理器设计的。现在Linux核心支持从个人电脑到大型主机甚至包括嵌入式系统在内的各种硬件设备。

现在，Linux已经成为了一种受到广泛关注和支持的操作系统。包括国际商用机器公司和惠普、戴尔在内的一些信息业巨头也陆续支持Linux，并且成立了一些组织支持其发展，如Open Invention Network（OIN）（成员有IBM，索尼，NEC，Philips，Novell，Red Hat 等）购买了微软专利，允许任何个体以开放的原则使用。很多人认为，和微软Windows相比，作为自由软件的Linux具有低软件成本，高安全性，更加可信赖等优势，但是同时却需要更多的人力成本。

应用程序

为了完成各种任务，计算机必须安装各种应用程序。应用程序要依赖于操作系统等系统软件的支持，系统软件同时也会为其提供内存管理、网络连接以及设备驱动等常规服务。

常见的应用程序包括文字处理程序、媒体播放程序以及游戏软件等。

制造厂商

戴尔（DELL）、惠普（HP）、富士通（Fujitsu）、西门子（Siemens）、联想（Lenovo）、IBM[7]、索尼（SONY）、NEC、东芝（TOSHIBA）、华硕（ASUS）、宏碁（ACER））、技嘉（GIGABYTE）等都是世界性的知名电脑品牌。

另类用途

电子游戏机

PC被很多人作为电子游戏机使用，不同于游戏机，PC的硬件可以更换而且更新速度快。但尽管如此，游戏的效果始终取决于对游戏机的兼容度，多数游戏机专用游戏只为游戏机作优化，而不是为PC优化，不管PC硬件多先进，该类游戏上始终不如游戏机。PC用模拟器模拟游戏机中的游戏较缓慢及差劣，因为PC并非该电视游戏专用的游戏机。

多媒体设备

PC可扩展性使很多人利用PC作为家庭的多媒体系统，例如加装电视卡、遥控器、5.1声道音响，以及利用DVD、蓝光光驱等代替专业的家庭剧院设备，以同时享受多种乐趣。

公用服务站

公共场所的公用信息服务站很多也是用PC零件组装生产，内载特制化过的操作系统和软件，只能运行特定公共服务功能，例如向特定网络商订票、订机位、查数据，连接特定金融机构转帐或金融服务等。

参考文献

直接内存存取

直接内存存取（Direct Memory Access，DMA）是计算机科学中的一种内存访问技术。它允许某些电脑内部的硬件子系统（电脑外设），可以独立地直接读写系统内存，而不需绕道CPU。在同等程度的CPU负担下，DMA是一种快速的数据传送方式。很多硬件的系统会使用DMA，包含硬盘控制器、绘图显卡、网络卡和声卡。

直接内存存取

DMA 是所有现代电脑的重要特色，他允许不同速度的硬件装置来沟通，而不需要依于 CPU 的大量 中断 负载。否则，CPU 需要从 来源 把每一片段的资料复制到 暂存器，然后把他们再次写回到新的地方。在这个时间中，CPU 对于其他的工作来说就无法使用。

DMA 传输重要地将一个内存区从一个装置复制到另外一个。当 CPU 初始化这个传输动作，传输动作本身是由 DMA 控制器 来实行和完成。典型的例子就是移动一个外部内存的区块到芯片内部更快的内存去。像是这样的操作并没有让处理器工作拖延，反而可以被重新排程去处理其他的工作。DMA 传输对于高效能 嵌入式系统 算法和网络是很重要的。

举个例子，PC ISA DMA 控制器拥有 8 个 DMA 通道，其中的 7 个通道是可以让 PC 的 CPU 所利用。每一个 DMA 通道有一个 16位元 位址暂存器和一个 16 位元 计数暂存器。要初始化资料传输时，装置驱动程式一起设定 DMA 通道的位址和计数暂存器，以及资料传输的方向，读取或写入。然后指示 DMA 硬件开始这个传输动作。当传输结束的时候，装置就会以中断的方式通知 CPU。

"分散-收集" (Scatter-gather) DMA 允许在一次单一的 DMA 处理中传输资料到多个内存区域。相当于把多个简单的 DMA 要求串在一起。再一次，这个动机是要减轻 CPU 的多次输出输入中断和资料复制任务。

DRQ 意为 DMA 要求；DACK 意为 DMA 确认。这些符号一般在有 DMA 功能的电脑系统硬件概要上可以看到。他们表示了介于 CPU 和 DMA 控制器之间的电子讯号传输线路。

Cache coherency problem

DMA 会导致 cache 一致性问题。想像 CPU 带有 cache 与外部内存的情况，DMA 的运作则是去存取外部内存，当 CPU 存取外部内存 X 位址的时候，暂时先将新的值写入 cache 中，但并未将外部内存 X 位址的资料更新，若在 cache 资料尚未更新到外部内存前发生了 DMA，则 DMA 过程将会读取到未更新的资料。

相同的，如果外部装置写入新的值到外部内存内，则 CPU 若存取 cache 的值则会存取到尚未更新的资料

这些问题可以用两种方法来解决：

一、快取同调系统（Cache-coherent system） 以硬件方法来完成，当外部装置写入内存时以一个信号来通知 cache 控制器某内存位址的值已经过期或是应该重新更新资料。

二、非同调系统（Non-coherent system） 以软件方法来完成，OS 必须确认 cache 读取时，DMA 程序已经开始或是禁止 DMA 发生。

第二种的方法会造成 DMA 的系统负担。

DMA 引擎

除了与硬件交互相关外，DMA也可为昂贵的内存耗费减负。比如大型的拷贝行为或scatter-gather操作，从CPU到专用的DMA引擎。Intel的高端服务器包含这种引擎，它被称为I/O Acceleration Technology (IOAT)。

Direct memory access

Direct memory access (DMA) is a feature of modern computers and microprocessors that allows certain hardware subsystems within the computer to access system memory for reading and/or writing independently of the central processing unit. Many hardware systems use DMA including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors, especially in multiprocessor system-on-chips, where its processing element is equipped with a local memory (often called scratchpad memory) and DMA is used for transferring data between the local memory and the main memory. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time and allowing computation and data transfer concurrency.

Without DMA, using programmed input/output (PIO) mode for communication with peripheral devices, or load/store instructions in the case of multicore chips, the CPU is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU would initiate the transfer, do other operations while the transfer is in progress, and receive an interrupt from the DMA controller once the operation has been done. This is especially useful in real-time computing applications where not stalling behind concurrent operations is critical. Another and related application area is various forms of stream processing where it is essential to have data processing and transfer in parallel, in order to achieve sufficient throughput.

Principle

DMA is an essential feature of all modern computers, as it allows devices to transfer data without subjecting the CPU to a heavy overhead. Otherwise, the CPU would have to copy each piece of data from the source to the destination, making itself unavailable for other tasks. This situation is aggravated because access to I/O devices over a peripheral bus is generally slower than normal system RAM. With DMA, the CPU gets freed from this overhead and can do useful tasks during data transfer (though the CPU bus would be partly blocked by DMA). In the same way, a DMA engine in an embedded processor allows its processing element to issue a data transfer and carries on its own task while the data transfer is being performed.

A DMA transfer copies a block of memory from one device to another. While the CPU initiates the transfer by issuing a DMA command, it does not execute it. For so-called "third party" DMA, as is normally used with the ISA bus, the transfer is performed by a DMA controller which is typically part of the motherboard chipset. More advanced bus designs such as PCI typically use bus mastering DMA, where the device takes control of the bus and performs the transfer itself. In an embedded processor or multiprocessor system-on-chip, it is a DMA engine connected to the on-chip bus that actually administers the transfer of the data, in coordination with the flow control mechanisms of the on-chip bus.

A typical usage of DMA is copying a block of memory from system RAM to or from a buffer on the device. Such an operation usually does not stall the processor, which as a result can be scheduled to perform other tasks unless those tasks include a read from or write to memory. DMA is essential to high performance embedded systems. It is also essential in providing so-called zero-copy implementations of peripheral device drivers as well as functionalities such as network packet routing, audio playback and streaming video. Multicore embedded processors (in the form of multiprocessor system-on-chip) often use one or more DMA engines in combination with scratchpad memories for both increased efficiency and lower power consumption. In computer clusters for high-performance computing, DMA among multiple computing nodes is often used under the name of remote DMA. There are two control signal used to request and acknowledge a DMA transfer in microprocess-based system.The HOLD pin is used to request a DMA action and the HLDA pin is an output acknowledges the DMA action.

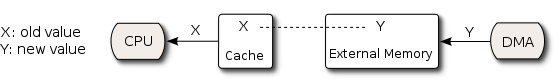
Cache coherency problem

DMA can lead to cache coherency problems. Imagine a CPU equipped with a cache and an external memory that can be accessed directly by devices using DMA. When the CPU accesses location X in the memory, the current value will be stored in the cache. Subsequent operations on X will update the cached copy of X, but not the external memory version of X. If the cache is not flushed to the memory before the next time a device tries to access X, the device will receive a stale value of X.

Similarly, if the cached copy of X is not invalidated when a device writes a new value to the memory, then the CPU will operate on a stale value of X.

This issue can be addressed in one of two ways in system design: Cache-coherent systems implement a method in hardware whereby external writes are signaled to the cache controller which then performs a cache invalidation for DMA writes or cache flush for DMA reads. Non-coherent systems leave this to software, where the OS must then ensure that the cache lines are flushed before an outgoing DMA transfer is started and invalidated before a memory range affected by an incoming DMA transfer is accessed. The OS must make sure that the memory range is not accessed by any running threads in the meantime. The latter approach introduces some overhead to the DMA operation, as most hardware requires a loop to invalidate each cache line individually.

Hybrids also exist, where the secondary L2 cache is coherent while the L1 cache (typically on-CPU) is managed by software.



DMA engine

In addition to hardware interaction, DMA can also be used to offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. Intel includes such engines on high-end servers, called I/O Acceleration Technology (IOAT).

Examples

ISA

For example, a PC's ISA DMA controller is based on the Intel 8237 Multimode DMA controller, that is a software-hardware combination which either consists of or emulates this part. In the original IBM PC, there was only one DMA controller capable of providing four DMA channels (numbered 0-3). These DMA channels performed 8-bit transfers and could only address the first megabyte of RAM. With the IBM PC/AT, a second 8237 DMA controller, was added (channels 5-7; channel 4 is unusable), and the page register was rewired to address the full 16 MB memory address space of the 80286 CPU. This second controller performed 16-bit transfers.

Due to their lagging performance (2.5 Mbit/s[1]), these devices have been largely obsolete since the advent of the 80386 processor and its capacity for 32-bit transfers. They are still supported to the extent they are required to support built-in legacy PC hardware on modern machines. The only pieces of legacy hardware that use ISA DMA and are still fairly common are the built-in Floppy disk controllers of many PC mainboards and those IEEE 1284 parallel ports that support the fast ECP mode.

Each DMA channel has a 16-bit address register and a 16-bit count register associated with it. To initiate a data transfer the device driver sets up the DMA channel's address and count registers together with the direction of the data transfer, read or write. It then instructs the DMA hardware to begin the transfer. When the transfer is complete, the device interrupts the CPU.

Scatter-gather DMA allows the transfer of data to and from multiple memory areas in a single DMA transaction. It is equivalent to the chaining together of multiple simple DMA requests. The motivation is to off-load multiple input/output interrupt and data copy tasks from the CPU.

DRQ stands for DMA request; DACK for DMA acknowledge. These symbols, seen on hardware schematics of computer systems with DMA functionality, represent electronic signaling lines between the CPU and DMA controller. Each DMA channel has one Request and one Acknowledge line. A properly configured device that uses DMA must be jumpered (or software-configured) to use both lines of the assigned DMA channel.

Standard ISA DMA assignments:

0 DRAM Refresh (obsolete),

1 User hardware,

2 Floppy disk controller,

3 Hard disk (obsoleted by PIO modes, and replaced by UDMA modes),

4 Cascade from XT DMA controller,

5 Hard Disk (PS/2 only), user hardware for all others,

6 User hardware,

7 User hardware.

PCI

As mentioned above, a PCI architecture has no central DMA controller, unlike ISA. Instead, any PCI component can request control of the bus ("become the bus master") and request to read from and write to system memory. More precisely, a PCI component requests bus ownership from the PCI bus controller (usually the southbridge in a modern PC design), which will arbitrate if several devices request bus ownership simultaneously, since there can only be one bus master at one time. When the component is granted ownership, it will issue normal read and write commands on the PCI bus, which will be claimed by the bus controller and forwarded to the memory controller using a scheme which is specific to every chipset.

As an example, on a modern AMD Socket AM2-based PC, the southbridge will forward the transactions to the northbridge (which is integrated on the CPU die) using HyperTransport, which will in turn convert them to DDR2 operations and send them out on the DDR2 memory bus. As can be seen, there are quite a number of steps involved in a PCI DMA transfer; however, that poses little problem, since the PCI device or PCI bus itself are an order of magnitude slower than rest of components (see list of device bandwidths).

A modern x86 CPU may use more than 4 GB of memory, utilizing PAE, a 36-bit addressing mode, or the native 64-bit mode of x86-64 CPUs. In such a case, a device using DMA with a 32-bit address bus is unable to address memory above the 4 GB line. The new Double Address Cycle (DAC) mechanism, if implemented on both the PCI bus and the device itself,[2] enables 64-bit DMA addressing. Otherwise, the operating system would need to work around the problem by either using costly double buffers (Windows nomenclature) also known as bounce buffers (Linux), or it could use an IOMMU to provide address translation services if one is present.

IO Accelerator in Xeon

As an example of DMA engine incorporated in a general-purpose CPU, newer Intel Xeon chipsets include a DMA engine technology called I/O Acceleration Technology (I/OAT), meant to improve network performance on high-throughput network interfaces, in particular gigabit Ethernet and faster.[3] However, various benchmarks with this approach by Intel's Linux kernel developer Andrew Grover indicate no more than 10% improvement in CPU utilization with receiving workloads, and no improvement when transmitting data.

AHB

In systems-on-a-chip and embedded systems, typical system bus infrastructure is a complex on-chip bus such as AMBA High-performance Bus. AMBA defines two kinds of AHB components: master and slave. A slave interface is similar to programmed I/O through which the software (running on embedded CPU, e.g. ARM) can write/read I/O registers or (less commonly) local memory blocks inside the device. A master interface can be used by the device to perform DMA transactions to/from system memory without heavily loading the CPU.

Therefore high bandwidth devices such as network controllers that need to transfer huge amounts of data to/from system memory will have two interface adapters to the AHB bus: a master and a slave interface. This is because on-chip buses like AHB do not support tri-stating the bus or alternating the direction of any line on the bus. Like PCI, no central DMA controller is required since the DMA is bus-mastering, but an arbiter is required in case of multiple masters present on the system.

Internally, a multichannel DMA engine is usually present in the device to perform multiple concurrent scatter-gather operations as programmed by the software.

Cell

As an example usage of DMA in a multiprocessor-system-on-chip, IBM/Sony/Toshiba's Cell processor incorporates a DMA engine for each of its 9 processing elements including one Power processor element (PPE) and eight synergistic processor elements (SPEs). Since the SPE's load/store instructions can read/write only its own local memory, an SPE entirely depends on DMAs to transfer data to and from the main memory and local memories of other SPEs. Thus the DMA acts as a primary means of data transfer among cores inside this CPU (in contrast to cache-coherent CMP architectures such as Intel's coming general-purpose GPU, Larrabee).

DMA in Cell is fully cache coherent (note however local stores of SPEs operated upon by DMA do not act as globally coherent cache in the standard sense). In both read ("get") and write ("put"), a DMA command can transfer either a single block area of size up to 16KB, or a list of 2 to 2048 such blocks. The DMA command is issued by specifying a pair of a local address and a remote address: for example when a SPE program issues a put DMA command, it specifies an address of its own local memory as the source and a virtual memory address (pointing to either the main memory or the local memory of another SPE) as the target, together with a block size. According to a recent experiment, an effective peak performance of DMA in Cell (3 GHz, under uniform traffic) reaches 200GB per second

免责声明

此信息分发的。笔者明确声明对任何利润或任何间接，偶然，或其他损害赔偿因使用或误用这些信息造成的损失的责任。这些信息可以自由发布任何形式，只要本声明保持不变。

简介

声霸卡16是FM和数字化声音的能力。数字化的声音capibilities范围从8位单声道5000赫兹的声音在44khz一路攀升到16位立体声。本FAQ文档编程SB16 DSP的数字化录音和音频播放CT1341芯片。早在编程之前Sound Blaster声卡的知识是必要的。

声霸卡16 DSP的I/ O端口

该SB16的DSP芯片编程使用多个I/在一个I / O地址可通过跳线设置确定相应的输出端口。在SB16，有16个I/这对于调频合成音乐，调音台的设置，DSP编程和CD - ROM访问使用O端口。这些端口用于五种编程的DSP。他们在下面列出。

2x6h-DSP的复位

2xAh-DSP的阅读

2xCh-DSP的写（指令/数据），数字信号处理器写缓冲区的状态（位7）

2xEh-DSP的读缓冲区的状态（位7），DSP的中断响应

2xFh-DSP的16位中断应答

DSP的复位

您必须重置的DSP程序中，然后才可以。 DSP可以重置使用以下步骤：

写1到复位端口（2x6）

等待3微秒

写一个0到复位端口（2x6）

调查的读缓冲区的状态端口，直到第7位（2xE）设置

轮询读取数据端口（2xA），直到您收到机管局

该DSP通常需要大约100至初始化自身微秒。经过这一段时间，如果返回值不是AA或没有数据可言，那么在SB卡可能无法安装或不正确的I / O地址正在使用。

写作到DSP

要编写的SB16字节，应使用下列程序：

读，直到第7位被清除了写缓冲区的状态端口（2xC）

写入的值写入端口（2xC）

从DSP读

要宣读了SB16字节，应使用下列程序：

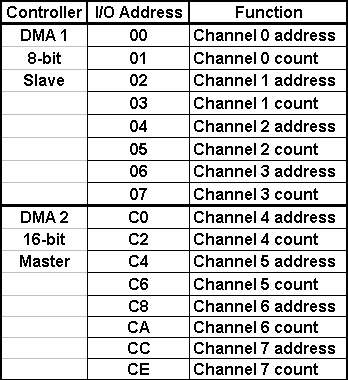
读，直到第7位被设置了只读的缓冲区状态端口（2xE）

读出读端口（2xA）值

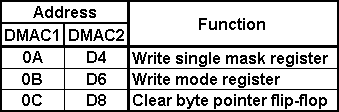
DMA控制器编程

的DMA（直接内存访问）控制器控制，而无需使用的CPU之间的I / O设备和存储器的数据传输。集成的英特尔8237 DMAC的circut是用来控制这一点。 IBM兼容的计算机有两个DMA控制器，一个为8位传输和16位传输之一。 DMA控制器，与外部页寄存器相结合，是64K的转移块组成的SB16的能力。这是对I / O口和寄存器设置声卡的I / O必要的信息：

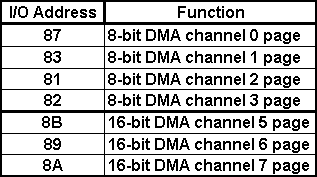
的I / DMA地址和计数寄存器O端口地址



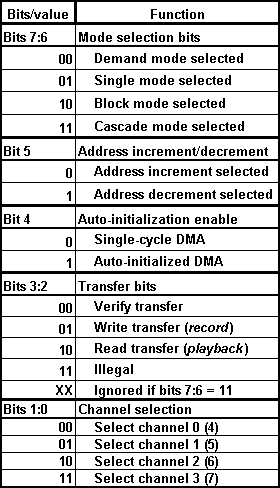
的I / O端口控制寄存器地址



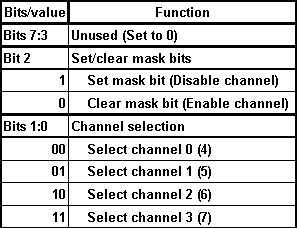
的I / O端口较低的页面寄存器地址



模式寄存器的位分配



写单屏蔽位分配



DMAC2是用于16位I / O和DMAC1为8位使用的I / O用于启动一个传输过程是复杂的，所以我会列出启动类型使用的DMA传输声音I / O的步骤：

计算绝对的线性地址的缓冲区

LinearAddr：=赛格（Ptr的^）\* 16 +的OFS（Ptr的^））;

禁用通过设置适当的屏蔽位声卡的DMA通道

端口[MaskPort]：= 4 +（通道模4）;

清除字节的指针触发器

端口[ClrBytePtr]：= AnyValue;

写出了DMA传输模式

模式选择位应设置为01的单模式。公司的地址/十二月位应设置为地址增量为0。自动初始化位应适当设置。我将讨论自动初始化DMA的更新版本。传输位应设置为10播放和录制01。通道选择应设置声卡的DMA通道。要知道，“读”是指从内存中读取（写入声卡），以及“写”是指写入系统内存。

端口[ModePort]：=模式+（通道模4）;

一些经常使用的模式如下：

48小时+通道 - 单周期回放

58h +通道 - 自动初始化播放

44高离子通道 - 单周期记录

54h +通道 - 自动初始化记录

写缓冲区的偏移，低字节的高字节。对于十六位数据，文字的偏移量应该是从一个128kbyte页的开头。用于计算的16位参数最简单的方法是将其分摊计算抵消了前两个线性地址。

如果SixteenBit

然后

开始

BufOffset：=（LinearAddr部2）模65536;

端口[BaseAddrPort]：=罗（BufOffset）;

端口[BaseAddrPort]：=你好（BufOffset）;

结束

其他

开始

BufOffset：= LinearAddr模65536;

端口[BaseAddrPort]：=罗（BufOffset）;

端口[BaseAddrPort]：=你好（BufOffset）;

结束;

写入传输长度，低字节的高字节。对于8位传输，写入的字节- 1号。对于16位传输，写的字1号。

端口[CountPort]：=罗（TransferLength - 1）;

端口[CountPort]：=喜（TransferLength - 1）;

写入缓冲区页的DMA页面寄存器。

端口[PagePort]：= LinearAddr分区65536;

通过清零相应的屏蔽位声卡的DMA通道

端口[MaskPort]：= DMAChannel国防部4;

设置采样率

与早先的爆破声中，SB16编程时间常数，而不是实际的采样率。在SB16，采样率设置命令41H章采用DSP和42h。命令41H章用于输出和42h是用于输入。我听说在SB16，目前这两个命令做同样的事情，但我会建议使用单独的命令，以保证未来的音效卡的兼容性。用于设置采样率的程序是：

写（输出率，输入速度42h41H章）命令

写的采样率的高字节（56h为22050赫兹）

写的采样率低字节（22小时为22050赫兹）

数字化的声音的I / O

要记录或播放声音，您应该使用以下顺序：

分配一个缓冲区，不跨一个64K的物理页边界

安装一个中断服务程序

工程规划中后台传输DMA控制器

设置采样率

写的I / O命令给DSP

写的I / O传输到DSP模式

写块大小到DSP（低字节/高字节）

当中断时使用单周期DMA：

计划明年块DMA控制器

程序DSP的下一个块

复制下一个块如果双缓冲

承认与SB中断由8位声音或端口2xF港口2xE16位声音读。

承认与写20H到港20小时知情同意结束中断。如果声卡IRQ8-15，你还必须写20H到A0h。

DSP的命令

D0的

暂停8位DMA模式数字化的声音I / O的命令Cxh启动。

适用于单周期和自动初始化的DMA的I / O

D4类

继续8位DMA模式数字化的声音I / O的暂停使用命令D0的。

适用于单周期和自动initialzied的DMA的I / O

D 5类

暂停16位DMA模式数字化的声音I / O的命令Bxh启动。

适用于单周期和自动初始化的DMA的I / O

D6级

继续16位DMA模式数字化的声音I / O的暂停使用命令D 5类。

适用于单周期和自动初始化的DMA的I / O

D9公路

退出16位自动初始化DMA模式数字化的声音后，当前块结束的I / O。

多巴胺

出口8位自动初始化DMA模式数字化的声音后，当前块结束的I / O。

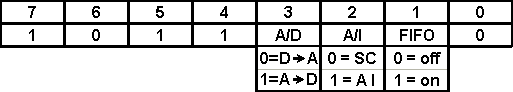
素E1

获得DSP的版本号。该命令发出后，回读两个字节形式的DSP。第一个字节是主版本号，第二个字节是次要版本号。应该有一个SB164.00或更高的DSP版本。检查前使用SB16特定的命令的。

Bx的

计划16位DMA模式数字化的声音的I / O

命令序列：命令，模式，芦（长度- 1），高（长度- 1）

**Command:**   


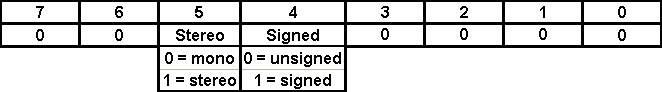
常用命令：

B8座- 16位单周期输入

B0的- 16位单周期输出

是 -16位自动初始化输入

B6的- 16位自动初始化输出

**Mode:** 

Cx的

计划8位DMA模式数字化的声音的I / O

同样的程序为16位声音I / O使用命令Bx的

常用命令：

C8的- 8位单周期输入

的C0 -8位单周期输出

行政长官- 8位自动初始化输入

的C6- 8位自动初始化输出

该FIFO用于消除在样本期间的音效卡时无法得到的DMA时，它需要它的不一致。带有FIFO残疾人，卡正是试图在瞬间，它需要一个样品的DMA。如果另一个具有更高优先级的设备正在访问的DMA，声卡等待样品和采样率可能会降低。允许的FIFO的DMA样本期间更不影响音频质量不稳定。当FIFO中被清除的命令被发送到DSP。在单周期模式中，DSP是不断重新编程。 FIFO的可能仍然包含一直没有清除时，输出的数据。为了避免这个问题，应关闭先进先出单周期模式。当在自动初始化模式下，DSP是永远不会重新编程。该FIFO可留在和健全的质量将得到改善。

自动初始化的DMA

当单周期使用DMA，声音输出停止在每个块的结尾。中断处理程序可以启动另一个传输，但不会有突破的产值。这将导致每块之间单击，降低音质。当自动初始化DMA的使用，声音输出循环围绕在缓冲区的末尾。 DMA控制器传输保持相同的内存块的DMA传输与发起。当到达缓冲区的末尾，它会开始传送通过自动初始化当前偏移和计数与在基偏移和计数寄存器中存储的值寄存器中的缓冲区了。要达到点击更少的输出通常的方法是分配一个缓冲区，它分为两个区块。计划与整个缓冲区的长度，但随着程序的块长度SB16 DMA控制器。 （缓冲区的一半），每一个中断发生块声卡，所以会出现两个中断每次播放缓冲时间，一旦在中点（第二个块的开始），并在最后一次（实际上，在启动首块）的中断处理程序应复制到这是刚刚结束，以便将数据准备好当它是输出所需的数据块。用于自动初始化DMA传输编程过程是相同的一个单周期DMA传输过程，但不包括DMA模式寄存器第4位和位DSP的命令3载。

当使用中断后自动初始化的DMA：

下一块复制到输出缓冲区块刚刚完成

承认与SB中断由8位声音或端口2xF港口2xE 16位声音读。

承认与写20H到港20小时知情同意结束中断。如果声卡IRQ8 - 15，你还必须写20H到A0h。

要立即停止声音：

8位 - 写DSP的命令D0h（稍停8位DMA模式数字化的声音的I / O）

16位 - 写DSP的命令D5h（稍停16位DMA模式数字化的声音的I / O）

这两个命令立即停止声音，如果没有中断。

要停止在当前块结束的声音：

8位 - 写DSP的命令大新（站8位自动初始化DMA的声音的I / O）

16位 - 写DSP的命令D9h（停止16位自动初始化DMA的声音的I / O）

这两个命令将停止在当前块结束的声音。如果您的程序不中断输出后编制完成，这可能会导致问题。

您还可以重新编程结束单周期模式的DSP自动初始化模式。该卡然后切换从A / I模式为S / C模式后，下一个中断。然后它将继续合播放或录制指定的长度，产生一个中断和停止。这将允许你停止输出恰好在年底的数据，而不需要使用DMA缓冲区的其余部分与沉默填写。这种技术可能会或可能不会对你有用。我会建议使用的命令记录在立即停止部分，除非另一种方法是更适合你的目的暂停。

7.1 DMA技术概述

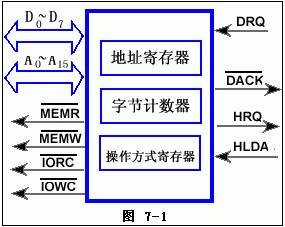
　　DMA（Driect Memory Access）技术是一种代替微处理器完成存储器与外部设备或存储器之间大量数据传送的方法，也称直接存储器存取方法。

　　在微机系统内，外设与内存间的数据传送通常是通过CPU执行一段程序来实现的，但利用DMA技术则可不用CPU介入就能实现外设与内存之间数据的直接传送。DMA的主要优点是当需要把一个外设的大量数据送到指定内存时，它可以自动完成传送任务，也就是说外设发出一个DMA请求，则DMA电路暂停CPU的操作，并控制外设与内存之间进行一次传数，然后再让CPU继续执行程序，这样就使CPU节省了大量对外设的查询时间，从而提高了系统的整体性能。

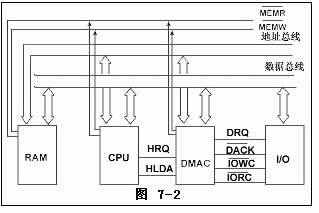
　　在DMA技术中，数据的传送是在DMA控制器（简称DMAC）下进行的，DMAC是一种能独立于微处理器进行操作的专用芯片或电路。在DMA数据传送过程中，DMA控制器接管了微处理器的地址总线、数据总线和控制总线。在当前总线周期结束之前，有一条外部控制线终止了微处理器的操作。在DMA开始传送数据时，微处理器的内存控制信号禁止使用；当DMA传送结束后，DMA控制器将终止信号复位，这样微处理器就恢复了它的一切权力并开始执行自己的操作。

　　由此可见，DMA必须具有控制系统总线的能力，即DMA能够像微处理器一样输出地址信号，接收或发出控制信号，输入或输出数据信号。

**7.1.2 DMA控制器的基本组成**

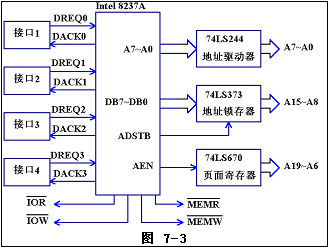
**1.DMAC的基本功能**   
　　实现DMA操作主要靠DMA控制器，其结构相当复杂。在DMA操作期间，CPU停止了工作，而由DMAC代替CPU进行工作，因此它必须具备CPU的寻址及数据传送能力，这主要包括如下几点：  
　(1)DMA控制器必须提供地址码以指明I/O设备变换数据的存储器起始地址；  
　(2)DMA控制器必须提供读/写脉冲，以规定数据在存储器与I/O设备之间的传输方向；   
　(3)DMA控制器必须修改内存地址指针并计算传送的字节数，以判断什么时候传送结束。  
**2.DMAC的基本组成**  
　　一个DMAC至少必须包括如图7-1所示的基本电路：  
　

§7.1.3 DMAC的操作过程

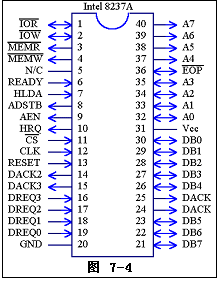


　　DMAC与CPU及存储器、I/O的连接方框图如图7-2所示。  
　　在进行DMA操作之前应先对DMAC编程。例如，确定传送数据的存储器的起始地址，要传送的字节数，以及传送方式等。  
　　下面以I/O设备将一个数据块写入存储器的操作为例，简述DMA操作的基本过程。  
　（1）I/O设备向DMAC发出请求信号DREQ；   
　（2）DMAC向CPU发出总线请求信号HRQ=1;   
　（3）CPU向DMAC发出总线响应信号HLDA=1，此时，DMAC获取了总线控制权；   
　（4）DMAC向I/O设备发出DMA响应信号DACK，表示DMAC已控制了总线，允许I/O设备与存储器交换数据；   
　（5）DMAC按地址寄存器的内容发出16位地址信号作为存储地址的选择，同时地址寄存器的内容加1（或减1，由编程定）；  
　（6）DMAC发出IORC信号到I/O设备，将I/O设备数据读入总线，同时发出MEMW信号，将数据总线的数据写入由地址总线选中的内存单元；  
　（7）字节计数器减1；   
　（8）重复5.、6.、7.步骤，直至字节计数器减到0为止，数据块的DMA方式传送工作宣告完成。这时,DMAC的HRQ降为低电平（HRO=0），总线控制权交给CPU。

7.2 Intel8237A的结构和引脚

§7.2.1 Intel 8237A的基本结构  
　　一片DMAC通常可以控制一个或几个外设接口电路与内存的DMA传输。习惯上，将DMAC中与某个外设备接口电路相联系的部分称为一个“通道（channel）”。如，Intel 8237A（简称8237），有四个通道如图7-3所示。每个通道都有独立的与相应外设接口相联系的信号，四个通道共享与CPU相连的控制信号、地址信号、数据信号,并且还可以用级联的方来扩充更多的信道,它允许在外部设备与系统存储器以及系统存储器与存储器之间交换信息,其数据传输率可达1.5MB/s.它提供了多种控制方式和操作类型,大大增强的系统的性能,8237A是一个高性能通用可编程的DMAC.  
　　8237A内部有四个独立通道,每个通道都有各自的四个寄存器(基地址,当前地址,基字节计数,当前字节计数),另外还有各个通道公用的寄存器(工作方式寄存器,命令寄存器,状态寄存器,屏蔽寄存器,请求寄存器以及暂存寄存器等).通过对这些寄存器的编程,可实现8237A的三种DMA操作类型和操作方式,2种工作时序,2种优先级排队,自动预置传送地址和字节数,以及实现存储器-存储器之间的传诵等一系列操作功能.

§7.2.2 Intel 8237A的引脚信号

**1.Intel 8237引脚信号**  
　　Intel 8237A控制器是一个40引脚的双列直插式组件,如图7-4所示。由于它即是主控者又是受控者,故其外部引脚设置也具有特色,如它的I/O读/写线(IOR，IOW)和部分地址线(A0~A3)都是双向的,另外,还设置了存储器读/写线(MEMR,MEMW)和16位地址输出线(DB0~DB7,A0~A7)。这些都是其他I/O接口芯片所没有的。下面对各引脚功能加以说明。  
（1）请求/应答信号

　　　外设接口电路向8237的请求信号：DREQ3~DREQ0。

　　　8237对外设接口电路的应答信号：DACK3~DACK0。

　　　8237向CPU申请总线的信号：HRQ（连至CPU的HOLD）。

　　　CPU向8237传送的允许使用总线信号：HLDA。

（2）地址信号

　　　CS：CPU初始化8237或读8237状态时所需的片选信号。

　　　A7~A0（输出）：8237访问存储器的地址信号的低8位。

　　　A3~A0（输入）：CPU初始化8237或读8237状态时，用于寻址8237内部寄存器。

（3）数据信号：

　　D7~D0。

（4）地址允许信号：AEN

　　8237作为主控时（8237控制总线），输出AEN=1。  
　　8237作为从控时（CPU控制总线），输出AEN=0。

（5）DMA传输结束信号：EOP

　　双向信号。当DMAC内部任一通道传输结束，8237发出EOPE。  
　　若由外部给DMAC送入有效的EOP，则强制DMAC内部所有通道结束传输。

（6）MEMR、CAS：

　　DMAC发出的存储器读/写信号。

（7）IOR、IOW：双向信号。

　　8237作为主控时，输出的I/O读/写信号。

　　8237作为从控时，CPU发出的I/O读/写信号，用于读/写8237内部寄存器。

（8）地址选通信号：ADSTB。

　　8237作为主控时，为存储器提供16位地址信号，其地址引脚A7~A0输出地址低8位，高8位地址由数据引脚　　DB7~DB0输出，经地址　　锁存器后形成存储器地址的A15~A8。ADSTB用于启动地址锁存器。

（9）READY：

　　 存储器或I/O的就绪信号。

2.8237引脚信号的加载

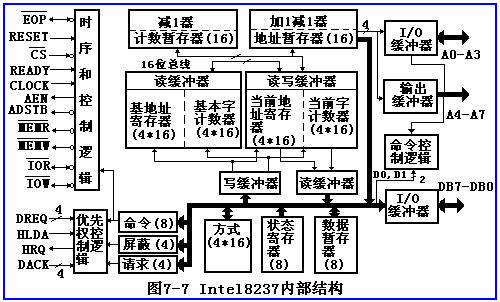
**§7.2.3 Intel 8237A的信号配合**

（1）8237作为从控时，与其他I/O接口一样，可以接收CPU的命令、数据，CPU也可读取其状态。这时，8237接收CPU发出的16位I/O地址信号， 高12位地址产生片选信号CS，据此判别本片8237是否被选中，低4位地址选择8237内部寄存器。

　　当IOR为低电平时，CPU读取8237内部寄存器的值；当IOW为低电平时，CPU将数据写入8237的内部寄存器。8237在从控状态中，输出AEN为低电平。

　　8237在复位后，处于从控模式，程序可对其进行初始化。初始化后，8237仍处于从控模式。当外设接口电路准备就绪，向8237请求传输（DREQ），8237再向CPU请求总线(HRQ)。　　　　　　　　　  
**（2）8237作为主控时**，当8237收到CPU允许使用总线的通知（HLDA）后，8237获得总线控制权，从而转入主控工作模式。  
　　在主控模式下，8237输出AEN为高电平，产生控制信号IOR、IOW、MEMW、MEMR，产生存储器地址信号。按初始化程序规定的方式执行I/O与存储器之间的数据传输。（传输过程本身不需CPU干预）。在传输完成后，8237放弃总线，转入从控模式，撤消总线请求（HRQ为低），CPU检测到HRQ（HOLD）为低后，撤消HLDA，CPU重新获得总线控制权。在传输完成后，8237还发出结束信EOP，CPU可以查询EOP信号，以便进行后续数据处理。或者把EOP信号作为中断请求信号，CPU在中断处理程序中进行后续数据处理。  
　　在主控模式下，8237可以产生16位存储器地址信号，其地址引脚A7~A0输出地址低8位，高8位地址由数据引脚DB7～DB0输出，经地址锁存器后形成存储器地址的A15~A8。用ADSTB启动地址锁存器。当系统的存储器地址多于16位时，多余位地址由一个称为“DMA页面寄存器”的接口电路提供。例如，在8086系统中，需要20位地址，高4位地址A19～A16由DMA页面寄存器电路74LS670提供。  
　　在DMA传输之前，用指令将高4位地址送入页面寄存器，在DMA传输过程中，由AEN信号使页面寄存器输出该4位地址。

§7.3 Intel 8237的编程结构

**Intel 8237的内部结构如图7-7所示:**  


[各寄存器/触发器的结构](http://course.cug.edu.cn/21cn/%E5%BE%AE%E5%9E%8B%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/content/chapter7/7-3/7-3-11.htm)

各寄存器/触发器的结构: [将鼠标指针移到上表第一列字的上面将在右边显示其结构] 各寄存器/触发器名称 共有位数

1、控制寄存器

8位 D7 D6 D5 D4 D3 D2 D1 D0

|DACK|DREQ|写入|优先|时序|工作|通道|存储

|极性|极性|选择|级编|选择|允许|口寻|器间

| 　 | 　 | 　 |码　|　　|　　|址　|传送 它们是16位地址寄存器，存放DMA传送的内存首址，在初始化时，由CPU以先底字节后高字节的顺序写入。传送过程中基值地址寄存器的内容不变。不能读，只能写。

2、状态寄存器 8位

3、暂时存储器

该存储器对存储器传送时，暂时保存从源地址读出的数据，RESET信号清除暂存器存储的内容。 8位

4、模式寄存器 8位

5、基地址寄存器 16位

6、基字节计数器

它存放DMA传送的总字节数，在初始化时，由CPU以先底字节后高字节的顺序写入。传送过程中，其内容不变。对其的预置应予以注意。

16位

7、先/后触发器

CPU写入基地址寄存器、基字节计数器时，均分两次写入，先/后触发器用于控制写入的值是低8位还是高8位。先/后触发器=0，写入低8位；先/后触发器=1，写入高8位

8位

8、请求触发器

D7 D6 D5 D4 D3 D2 D1 D0

|-----------未用---------|请求|通道选择| 8位

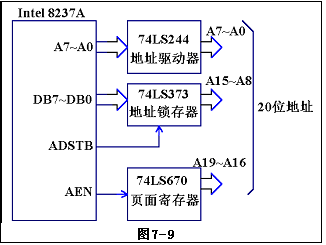
9、屏蔽触发器 8位

D7 D6 D5 D4 D3 D2 D1 D0

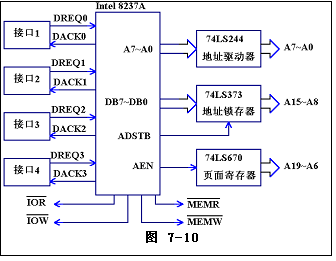
|----------未用---------|屏蔽| 通道选择|

§7.3.2 Intel8237A对主存和I/O寻址

1. **存储器地址**

　　如图7-9所示,在主控模式下，8237可以产生16位存储器地址信号，其地址引脚A7～A0输出地址低8位，高8位地址由数据引脚DB7～DB0输出，经地址锁存器后形成存储器地址的A15～A8。用ADSTB启动地址锁存器。当系统的存储器地址多于16位时，多余位地址由一个称　为“DMA页面寄存器”的接口电路提供。   
　　在PC/XT系统中，需要20位地址，高4位地址A19～A16由DMA页面寄存器电路74LS670提供。在DMA传输之前，用指令将高4位地址送入页面寄存器，在DMA传输过程中，由AEN信号使页面寄存器输出该4位地址。   
　　一片74LS670有四个4位寄存器，分别用于存放四个通道的A19～A16，这四个寄存器的地址分别为80H、81H、82H、83H。  
　　在PC/AT系统中，使用2片8237， 并用74LS612作页面寄存器，  
　　第一片8237对应的页面寄存器地址分别为：80～83H，第二8237对应的页面寄存器地址分别为：87～8AH。

**2. I/O寻址**

　 　8237的四个通道,如图7-10所示每一个只接一个外设接口电路，不使用I/O地址来区分I/O设备。当某I/O请求DMA传输时，8237向该I/O发　出DACK信号，当DACK和IOR/IOW同时有效时，该I/O设备就能输入/输出数据，而与I/O设备的端口地址无关。  
　I/O设备的端口地址只用于CPU对I/O设备的寻址，因此，I/O端口译码电路要加入AEN信号。只有当AEN=0（由CPU控制总线）时，才使片选信号CS有效，即才使用I/O端口的概念。

**§7.4 Intel8237程序设计举例**

**7.4.1 8237的工作模式和模式寄存器**  
**8237有四种传输模式:**  
　1.单字节传输模式：每传输一字节后，当前字节计数器减1，当前地址寄存器加1或减1，接着8237释放总线。这样，CPU至少可得到一个总线周期。但是，8237在释放总线后，立即对DREQ信号进行检测，若DREQ有效，又立即向CPU申请总线，以便传输下一字节。

　　每传输一字节后，当前字节计数器减1，当前地址寄存器加1或减1，接着8237释放总线。这样，CPU至少可得到一个总线周期。但是，8237在释放总线后，立即对DREQ信号进行检测，若DREQ有效，又立即向CPU申请总线，以便传输下一字节。

　2.块传输模式：8237获得总线控制权后，连续传送多个字节，每传输一个字节，当前字节计数器减1，当前地址寄存器加1或　减1，直到所要求的字节数传输完（当前字节计数器减至0），8237在EOP引脚上发出结束信号，才释放总线。在块传输过程中8237的EOP引脚上输入低电平，也可强行结束传输。

　　8237获得总线控制权后，连续传送多个字节，每传输一个字节，当前字节计数器减1，当前地址寄存器加1或　减1，直到所要求的字节数传输完（当前字节计数器减至0），8237在EOP引脚上发出结束信号，才释放总线。在块传输过程中8237的EOP引脚上输入低电平，也可强行结束传输。

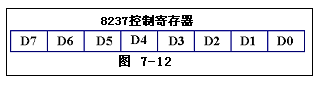
　3.请求传输模式（又称询问传输模式）：与块传输模式类似，但在传输过程中，8237要检测DREQ信号（询问外设），当DREQ为低时，8237暂停传输（不释放总线），当DREQ再次有效后，继续进行传输。

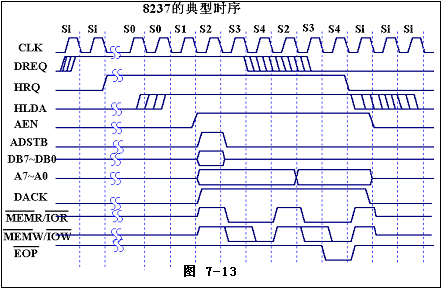
　　与块传输模式类似，但在传输过程中，8237要检测DREQ信号（询问外设），当DREQ为低时，8237暂停传输（不释放总线），当DREQ再次有效后，继续进行传输。

　4.级联传输模式：如图7-11多片8237级联时，可以构成主从式DMA系统。级联的方式是把从片的请求线HRQ连至主片的DREQ引脚，主片的DACK联至从片的HLDA引脚，主片的HRQ连至CPU的HOLD引脚。这样，最多可用5片8237构成16通道DMA系统。若主8237的某通道（DREQ）连接从8237的HRQ，主8237的该通道应设置为级联传输模式，但从8237不设置级联传输模式，而是设置其中三种模式之一。  
　　D7，D6：用于模式选择  
　　00= 请求传输模式　01=单字节传输模式  
　　10= 块传输模式　　11=级联传输模式  
　　D5：存储器地址增减选择，0=地址增1，1=地址减1。  
　　D4：自动预置功能，0=禁止， 1=允许。  
　　自动预置：在当前字节计数器到达0时，当前字节计数器和当前地址寄存器从基字节计数器和基地址寄存器中自动获得新的初值，从而可以进入下一个数据传输过程。 D3，D2：传输类型选择：   
　　00=校验传输 01=写传输（I/O->内存）   
　　10=读传输（内存->I/O）11=无意义   
　　校验传输用来对读传输和写传输功能进行校验，它是一种虚拟传输，8237也会产生存储器地址信号和EOP信号，但不产生存储器读/写、I/O读/写信号。校验传输一般用于器件测试，使用者不对此感兴趣。D1，D0：通道选择   
　　00=通道0 01=通道1   
　　10=通道2 11=通道3

**§7.4.2控制寄存器及其相关问题**

**1.8237控制寄存器**

　如图7-12所示:  
　D7： 0= DACK低电平有效，1=DACK高电平有效。     
　D6： 0= DREQ高电平有效，1=DREQ低电平有效。  
　D5： 0=不扩展写信号，1=扩展写信号。  
　D4： 0=固定优先级，1=循环优先级。  
　D3： 0=普通时序，1=压缩时序。  
　D2： 0=启动（指允许）8237工作，1=停止（指禁止）8237工作。  
　D1：内存到内存传输时，D1=1使源地址保持不变。  
　D0：0=内存  
　（1）内存到内存的传输 （PC系列机未使用这种方式）多数情况下，8237进行外设接口与内存间的传输，它也可完成内存到内存的传输。即，把数据从一个内存区域传送到另一内存区域。为了实现这种传输，先把源区的数据送到8237的暂存器，然后从暂存器送到目的区，因此，内存到内存的传输需使用两个总线周期。  
` 　执行内存到内存的传输时，8237固定使用通道0的地址寄存器存放源地址，通道1的地址寄存器和字节计数器存放目的地址和计数值。在传输时，目的地址的值像通常一样，进行加1或减1。源地址的值可以加1或减1，也可以通过设置使其保持不变，当保持不变时，就使同一个数据传输到整个目的区。  
　（2）8237的启动和停止  
　注意：这里的启动和停止是指允许与禁止，不同于设置“请求触发器”来发出DMA请求。  
　由控制寄存器的D2位控制8237的允许与禁止。  
　（3）8237的操作时序



**1.8237控制寄存器**

（3）8237的操作时序如图7-13所示:  
　　8237的时序包括SI、S0、tAV、S2、S3、S4、Sw七个状态。  
　SI：空转状态（Idle）

无DMA请求时，8237处于SI状态。在SI状态，8237对DREQ引脚进行测试，判断是否有请求。同时，它也对CS进行测试，如果CS=0，并且四个通道均无请求，则8237工作于从控状态，CPU可像普通接口电路那样对其进行初始化或读取其状态。

　S0：请求状态

如果，某一通道有DREQ请求，则8237将驱动HRQ为高电平，向CPU提出DMA请求，8237进入S0状态，等待CPU交出总线控制权。

[tAV：获得总线控制权](http://course.cug.edu.cn/21cn/%E5%BE%AE%E5%9E%8B%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/content/chapter7/7-4/7-4-11.htm)

当8237收到CPU发出的总线允许信号HLDA，8237脱离S0进入tAV状态。8237获得总线控制权，输出AEN信号。在8237占用总线期间，AEN信号不变，因此，一般情况下，tAV状态被跳过，仅在第一次8237才执行tAV状态。

　S2：输出地址并发出读/写控制信号

8237在A7~A0引脚输出地址低8位，在DB7～DB0引脚输出高8位地址A15～A8及锁存信号ADSTB，并发出读/写控制信号IOR、IOW、MEMR、MEMW。注意，高8位地址A15～A8要在S3状态才出现在系统地址总线上。

　SW：等待状态

如果外设的速度不够快，在S2状态后插入SW状态。

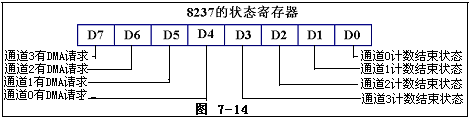
　S3：锁存地址高8位

8237仅在第一次执行S2时，才输出地址高8位及其锁存信号ADSTB，然后在S3状态锁存高8位地址，高8位地址出现在系统地址总线上。一般情况下，高8位地址不变，因此，不执行S3状态，仅在需要时（第一次，及以后需修改高8位地址时），才执行S3状态。

　S4：测试传输模式

在S4状态，8237对传输模式进行测试。如果是块传输模式，立即进入S2。如果是请求传输模式，测试DREQ，当DREQ无效时，8237暂停传输，当DREQ有效时，进入S2。如果是单字节传输模式，则释放总线，进入SI空转状态。如果当前字节计数器的值为0，表明本次传输结束，则释放总线，发出EOP信号，进入SI空转状态。 　8237有两种传输时序：普通时序和压缩时序。  
　普通时序：使用S2、S3、S4；  
　压缩时序：使用S2、S4。

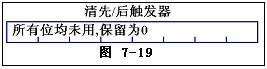
**2.状态寄存器**

2.状态寄存器 如图7-14所示:  
　 　状态寄存器的低四位用来指出四个通道的计数结束状态。比如，D3位为1，表示通道3计数到达0，因而到达计数器结束状态。与此类似，D0,D1,D2位为1分别表示通道0，通道1，通道2到达计数结束状态。  
　状态寄存器的高四位表示当前四个通道DMA请求响应情况。D7位为１，表示通道３当前有DMA请求需要处理，与此类推，D6,D5,D4位为１分别表示通道２，通道１和通道０有等待处理的DMA请求。   
　D3～D0：分别表示通道3～0的计数结束状态；  
　D7～D4：分别表示通道3～0有DMA请求。

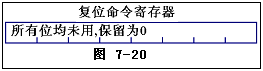
3.请求与屏蔽寄存器

3.请求与屏蔽寄存器所示:  
　软件设置DMA请求。  
　D2：=1设置DMA请求。  
　D1～D0：指定软件DMA请求的通道。8237的每个通道都配备1个DMA请求触发器和1个DMA屏蔽触发器，它们分别用来设置DMA请求标志。在物理上，4个请求触发器对应1个DMA请求寄存器，4个屏蔽触发器对应1个屏蔽寄存器。  
　当一个通道的DMA屏蔽标志为1时，这个通道就不能接收DMA请求了，这时，不管是硬件的DMA请求，还是软件的DMA请求，都不会被受礼。如果一个通道并没有设置自动谕旨功能，那么，EOP信号有效时，就会自动设置屏蔽标志。  
　在软件上，是怎样设置DMA请求标志和DMA屏蔽标志呢？  
　在DMA请求寄存器中请求字节就可以实现对DMA请求标志的设置如[图7-15](http://course.cug.edu.cn/21cn/%E5%BE%AE%E5%9E%8B%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/content/chapter7/7-4/7-4-13.htm)。DMA请求寄存器中的D1,D0 为用来指出通道号，D2位用来表示是否对相应 通道设置DMA请求， 如D2为1，则使相应的通道的DMA请求触发器置1，于是产生DMA请求，如D2为0，则无请求。   
   当EOP信号有效时，请求标志被清除如图7-16所示: DMA屏蔽标志是通过往屏蔽寄存器中写入屏蔽字节来设置的D2：=1，设置屏蔽； =0时则去除屏蔽。  
　此外，8237还允许综合屏蔽命令来设置通道屏蔽触发器如图7-17所示:D1～D0：选通道 D3～D0：分别对通道3～0设置屏蔽。这样，用综合屏蔽命令就可以一次 完成对4个通道的屏蔽设置。  
　　清除屏蔽如图7-18所示: 对该寄存器的写入操作（写入的值无关），将清除所有四个通道的屏蔽。

**4.清先/后触发器**

4.清先/后触发器   
　　如图7-19所示:  
　　CPU写入基地址寄存器、基字节计数器时，均分两次写入，先/后触发器用于控制写入的值是低8位还是高8位。先/后触发器=0，写入低8位；先/后触发器=1，写入高8位。  
对该寄存器的写入操作（写入的值无关），使8237先/后触发器=0。

**５.复位命令**

５.复位命令   
　如图7-20所示:（又称综合清除命令或总清命令）  
　复位命令又叫综合清除命令，它的功能和RESET信号相同。也就是说，复位命令使控制寄存器，状态寄存器,DMA请求寄存器以及先/后触发器都清0，而使屏蔽寄存器置位。

|  |  |
| --- | --- |
| 控制寄存器 | 清0 |
| 状态寄存器 |
| 请求寄存器 |
| 暂存器 |
| 先/后触发器 |
| 屏蔽寄存器 | 置1 |

**练习题**

**[基本练习]**  
1、中断传输方式和DMA方式比较，有什么不同之处？  
2、叙述用DMA方式传送单个数据的全过程。  
3、对一个DMA控制器的初始化工作包括哪些内容？  
4、8237A有哪几种工作模式？什么叫DMA控制器的自动预置功能？  
5、试编写出一个用8237A请求方式执行数据传送的程序，把500个字节数据从首址为BUF+500传送到首址为BUF的数据区中。设8237A的地址为1020H~102FH.

**[拓展练习]**  
1、CPU对DMA控制器的总线请求响应要比对中断请求快，请分析原因。  
2、Intel8237A在进行单字节DMA传输时方式和块方式DMA传输时，有什么区别？  
3、DMA控制器8237A什么时候作为主模块工作？什么时候作为从模块工作？在这两种情况下，各控制信号处于什么状态，试作说明。  
4、采用DMA方式在内存与I/O设备之间传送数据时，DMA控制器8237A怎样实现对I/O设备的寻址访问？

**自测题**

**自测题**

1、8237A只有8根数据线，为何能支持16位RAM和16位的I/O接口之间的DMA传输？DMA控制器选择储存器到储存器的传送操作必须具备哪些条件？  
2、简述DMA计数主要用于什么场合？  
3、简述DMA系统的主要环节。  
4、简述DMA控制器有效地址的生成过程。  
5、试说明DMA控制器初始化编程需要注意那些问题？  
6、利用8237A的通道2，由磁盘输入32KB的数据块，传送至内存68000H开始的区域，采用增量、块连续的方式传送，磁盘的DREQ和DACR都是高电平有效。试编写初始化程序。

**自测题答案**

1、解： DMA控制器取得总线控制权，向系统总线发出控制储存器和I/O设备操作需要的读写信号。传送数据的途径是I/O设备到I/O设备接口，再到系统数据总线，最后到存储器，不经过DMA控制器。DMA控制器只起控制作用。所以，虽然8237只有8根数据线，但可以支持16位RAM和16位I/O接口之间的DMA传送。  
　DMA选择存储器到存储器的传送操作必须具备通道0、通道1均未被占用的条件，通道0产生源地址，通道1产生目标地址。  
  
2、答：采用DMA计数传送是让存储器与高速外设直接交换，不需要CPU干预，减少中间环节，并且内存地址的修改，传送完毕的结束报告都由硬件完成，大大提高了传输速度。  
  
3、答：DMA系统主要由：DMA控制器（如：Intel8257、8237）、DMA页面寄存器和总线裁决逻辑三个部分组成。  
  
4、答：DMA控制器有效地址的生成是指：当DMA控制器取得总线控制权后，作为系统的主控制器如何向存储器和I/O设备发地址信号。（1）提供存储器地址：由DMA系统页面地址寄存器产生DMA通道的高4位地址A16~A19，并与控制器（如：8237A）输出的16位地址一起组成20位地址，以访问存储器（1M）的全部单元。（2）提供I/O设备地址：对请求以DMA方式传送的I/O设备，在进行读写数据时，只要BACK信号和RD/WR信号同时有效，就完成对I/O设备的读写操作，而与I/O设备的端口地址无关。   
  
5、答：（1）将要编程的通道加以屏蔽；编程完成后，再允许芯片工作或清除屏蔽位。（2）所有通道方式字寄存器都要加载。（3）在系统上电期间，要对芯片进行检测，只有检测通过后方可初始化。  
  
6、解：设首地址为DMA，初始化程序片段如下：  
  
　OUT DMA+0DH，AL ；送复位命令  
　MOV AL，00H  
　OUT DMA+4，AL ；送基地址和当前地址底8位  
　MOV AL，80H  
　OUT DMA+4 ，AL ；送基地址和当前地址高8位  
　MOV DX，0081H  
　MOV AL，06H  
　OUT DX，AL ；送最高4位的地址  
  
　MOV AL，0FFH  
　OUT DMA+5，AL ；送基字节和当前字节计数器初值  
　MOV AL，7FH  
　OUT DMA+5，AL  
　  
　MOV AL，86H  
　OUT DMA+8，AL ；根据题目要求写入方式字  
　MOV AL，02H  
　OUT DMA+0AH，AL ；写入屏蔽字（2号通道被屏蔽）  
　MOV AL，0A0H  
　OUT DMA+8，AL ；写入命令字

**课堂讨论**

对8237A的4个通道进行初始化前测试检查，先写如FFFFH后再读出，比较读写操作是否正确；如正确，则再写入0000H, 同样读出校验，若仍正确，则认为8237A工作正常，就可以对其初始化；如有错，则执行暂停命令。  
　解：程序中用标号DMA表示首地址。程序段如下：  
　MOV AL,04H  
　OUT DMA+8H,AL;禁止DMA工作  
　OUT DMA+0DH,AL；送复位命令  
　MOV AL,0FFH  
　C16: MOV BL,AL  
　MOV BH,AL  
　MOV CX,8  
　MOV DX,DMA  
　C17:OUT DX,AL  
　OUT DX,AL  
　IN AL,DX；读入底字节  
　MOV AH,AL  
　IN AL,DX；读入高字节  
　CMP BX,AX；判断是否正确  
　JE C18  
　HLT  
　C18:INC DX ；端口地址加一，指向下一寄存器  
　LOOP C17  
　INC AL；使AL=0，再循环一次  
　JZ C16  
　讨论：  
　[1]对8237A进行初始化编程时首先应利用硬件产生RESET信号，或用软件复位命令对8237A复位，使8237A进入空闲状态。这样才可使8237A处于从属地位，进行编程。  
　[2]本题中是对8237A的4个通道的基地址和当前地址寄存器、基字节和当前字节计数器进行检测。端口地址分别为DMA+(0~7)共8个，故设置计数器指针CX为08，这样两次循环即可完成题目要求。

**计算机的启动过程**

当按下电源开关时，电源向主板及其它设备供电，这时的电压还不太稳定，主板上的**控制芯片组**会向CPU发出并保持一个RESET（重置）信号，让CPU的内部部件处于初始状态(◆CS=FFFFH,IP=0000H)。此时的CPU也不会执行指令。

当芯片组检测到电源已经开始稳定供电了，它便撤去RESET信号，CPU马上就从地址**FFFF0**H处开始执行指令。

这个地址是主板（系统）BIOS的低值范围，放在这里的只是一条跳转指令，跳转到系统BIOS中真正的启动代码处，执行BIOS中的程序。系统BIOS的启动代码首先要做的事情就是进行**POST（Power-On Self Test，加电后自检）**。

POST的主要任务是检测系统中一些关键设备是否存在和能否正常工作。由于POST是最早进行的检测过程，此时显卡还没有被初始化，不能提供检测的结果。如果系统BIOS在进行POST的过程中发现了一些致命的错误，那么系统BIOS就会通过控制喇叭的发声来报告错误。声音的长短和次数代表了错误的类型。

接下来，系统BIOS将查找显卡BIOS，在C0000H处调用显卡BIOS的初始化代码。由显卡BIOS来初始化显卡，此时多数显卡都会在屏幕上显示出一些初始化信息，介绍生产厂商，图形芯片类型等内容。

接着，系统BIOS将查找其他设备的BIOS程序。找到后同样要调用这些BIOS内部的初始化代码来初始化相关的设备。完成以后，系统BIOS将显示出它自己的启动画面，其中包括系统BIOS的类型，型号和版本等内容。

接着，系统BIOS将检测和显示CPU的类型和工作频率，然后开始测试收有的RAM，并同时在屏幕上显示内存测试的进度。

内存测试通过之后，系统BIOS将开始检测系统中安装的一些标准硬件设备，包括硬盘，CD-ROM，串口，并口，软驱等设备，另外绝大多数较新版本的系统BIOS在这一过程中还要自动检测和设置内存的定时参数，硬盘参数和访问模式等。

标准设备检测完毕后，系统BIOS内部的支持即插即用的代码将开始检测和配置系统中安装的即插即用设备，每找到一个设备之后，系统BIOS都会在屏幕上显示出设备的名称和型号等信息，同时为该设备分配中断，DMA通道和I/O端口等资源。

到这一步为止，所有硬件都已经检测配置完毕了，多数系统BIOS会重新清屏并在屏幕上方显示出一个表格，其中，概略地列出了系统中安装的各种标准硬件设备，以及他们使用的资源和一些相关工作参数。

系统BIOS的最后一项工作是，根据用户指定的启动顺序（软盘，硬盘，光驱等，一般保存在CMOS中的）启动计算机。BIOS负责将启动盘的**第1扇区**的内容读到**0：7C00**H处，并交出控制权，从0:7C00H处开始执行。以C盘启动DOS为例，系统BIOS将读取并执行硬盘上的主引导记录（MBR），主引导记录(MBR)接着从分区表中找到第一个活动分区，然后读取并执行这个活动分区的分区引导记录。分区引导记录将负责读取并执行IO.SYS，这是DOS和Windown9x最基本的系统文件。IO.SYS为操作系统的启动做好准备。

如果系统之中安装有引导多个操作系统的启动的话，通常主引导记录将被替换成该软件的引导代码，这些代码将允许用户选择一个操作系统，然后读取并执行该操作系统的基本引导代码。

加电自检

加电自检又称为开机自我检测（英文Power-on self-test，常用简称POST），是计算机BIOS的一个功能，在开机后会运行，针对计算机硬件如CPU、主板、存储器等进行检测，结果会显示在固件可以控制的输出接口，像屏幕、LED、打印机等等设备上。

测试流程:

1.开机系统重置REST 启动CPU。

2.CPU指向BIOS自我测试的地址FFFF0H并打开CPU运行第一个指令。

3.CPU内部暂存器的测试。

4.CMOS 146818 SRAM检查。

5.ROM BIOS 检查码测试。

6.8254计时/计数器测试。

7.8237 DMA控制器测试。

8.74612页暂存器测试。

9.REFRESH刷新电路测试。

10.8042键盘控制器测试。

11.DRAM 64KB基本存储器测试。

12.CPU保护模式的测试。

13.8259中断控制器的测试。

14.CMOS 146818电力及检查码检查。

15.DRAM IMB以上存储器检查。

16.显卡测试。

17.NMI强制中断测试。

18.8254计时/计数器声音电路测试。

19.8254计时/计数器计时测试。

20.CPU保护模式SHUT DOWN测试。

21.CPU回至真理模式 (REAL MODE)。

22.键盘鼠标测试。

23.8042键盘控制器测试。

24.8259中断控制器IRQ0至IRQ18创建。

25.磁盘驱动器及界面测试。

26.设置并行打印机及串行RS232的界面。

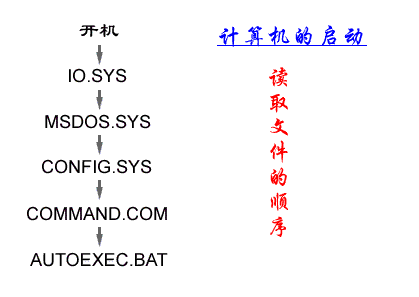
27.检查CMOS IC时间、日期。

28.检查完成

DOS启动

电脑刚启动时，要先从某个磁盘上将一些系统程序读入内存后才能正常工作，我们将这个磁盘叫做启动盘。

　　通常，我们用A盘或C盘作为启动盘，在特殊情况下，硬盘的其它盘符和少数电脑的光盘也可以作启动盘。制作启动盘需要使用sys命令，可以参看后面的命令部分。



　　DOS系统启动时，要先从启动盘中读取两个系统文件IO.SYS、MSDOS.SYS，然后在启动盘的根目录下寻找并执行 CONFIG.SYS、COMMAND.COM、AUTOEXEC.BAT三个文件。其中IO.SYS、MSDOS.SYS和 COMMAND.COM这三个文件缺一不可，否则电脑将无法正常启动

　　CONFIG.SYS和AUTOEXEC.BAT这两个文件是用来配置系统运行环境和自动执行一些命令，它们对电脑的运行性能以及许多设备的驱动起到了至关重要的作用，我们可以根据自己的需要对它们的内容进行修改,

　　autoexec.bat叫做自动批处理文件，它是批处理文件的一种，因为DOS启动时会自动运行它，所以叫做自动批处理文件。你可以把自己每次启动电脑时都要运行的程序放在里面。

　　如果电脑在根目录下找不到这两个文件，也是可以运行的，但有许多复杂的软件和设备将无法正常工作。比如光驱、声卡以及Windows程序等。所以，初学者应注意不要将上面的几个文件删除，否则，电脑将不能正常工作。

**内存地图**

内存的每一个字节都有一个用16进制数标识的地址。如果一位16进制数能表示4位的内存单元地址，那么20位的内存单元地址（实模式）就需要5位16进制数表示。这段内存地址空间是0-FFFFFH，容量是1MB。其中，低端的640KB是基本内存，A0000H-BFFFFH是给显存使用，C0000H-FFFFFH是给BIOS使用的，显卡BIOS一般在C0000H-C7FFFH处，IDE控制器的BIOS在C8000H-CBFFFH处，系统BIOS一般占用了最后的64KB或更多一点。

空闲的地址空间是00500H-07BFFH和07E00H-9FBFFH。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 10FFFFH |  |  |  |  |
|  |  |  | High Memory Area(A20 on) |  |  |  |
|  |  | 100000H |  |  |  |  |
| 386 Real mode Address | Upper Memory Block | FFFFFH |  | 1048575 | 1024 | KB |
|  | BIOS Boot Block |  |  |  |
| FFFF0H |  | 1048560 |  |  |
| FEFFFH |  | 1044479 | 1020 | KB |
|  | BIOS Boot Block |  |  |  |
| FE000H |  | 1040384 |  |  |
| FDFFFH | BIOS Routine＆Card BIOS | 1040383 | 1016 | KB |
| C0000H |  | 786432 |  |  |
| BFFFFH |  | 786431 | 768 | KB |
|  | CGA/EGA+chroma Text Video Buffer |  |  |  |
| B8000H |  | 753664 |  |  |
| B7FFFH |  | 753663 | 736 | KB |
|  | Mone Text Video Buffer |  |  |  |
| B0000H |  | 720896 |  |  |
| AFFFFH |  | 720895 | 704 | KB |
|  | EGA/VGA/XVGA Graphic Video Buffer |  |  |  |
| A0000H |  | 655360 |  |  |
| Base Memory | 9FFFFH |  | 655359 | 640 | KB |
|  | Extended BIOS Data Area(usually 1K) |  |  |  |
| 9FC00H |  | 654336 |  |  |
| 9FBFFH |  | 654335 | 639 | KB |
|  |  |  |  |  |
| 80000H |  | 524288 |  |  |
|  |  |  |  |  |
| 07E00H |  | 32256 |  |  |
| 07DFFH |  | 32255 | 512 | B |
|  | Boot Sector(512B) |  |  |  |
| 07C00H |  | 31744 |  |  |
| 07BFFH |  | 31743 | 31 | KB |
|  |  |  |  |  |
| 00500H |  | 1280 |  |  |
| 004FFH |  | 1279 |  |  |
|  | BIOS Data Area |  |  |  |
| 00400H |  | 1024 |  |  |
| 003FFH | 256\*4B=1024B | 1023 | 1 | KB |
|  | Interrupt Vector(中断向量表) |  |  |  |
| 00000H |  | 0 |  |  |

**文件系统**

是一种存储和组织计算机文件和数据的方法，它使得对其访问和查找变得容易。文件系统通常使用硬盘和光盘这样的存储设备，并维护文件在设备中的物理位置。

文件系统是一种用于向用户提供底层数据访问的机制。它将设备中的空间划分为特定大小的块（扇区），一般每块512字节。数据存储在这些块中，大小被修正为占用整数个块。由文件系统软件来负责将这些块组织为文件和目录，并记录哪些块被分配给了哪个文件，以及哪些块没有被使用。

文件名

不论文件系统底层是不是存储设备，文件系统都可以把数据组织为**文件**及**目录**的形式。文件系统一般会把文件名链接到某种文件分配表中（MS-DOS的FAT文件系统），或者链接到一个文件链表的节点上（Unix-like文件系统）。目录可以是平的结构，也可以是分层式结构，后者可以在目录中创建子目录。

有的文件系统中，文件名是结构化的，带有文件名扩展信息及版本号等；而另一些文件系统里，文件名只是一个简单的字符串，每个文件的属性信息保存在其它地方。

元数据（Metadata）

其它文件保存信息常常伴随着文件自身保存在文件系统中。文件长度可能是分配给这个文件的区块数，也可能是这个文件实际的字节数。文件最后修改时间也许记录在文件的时间戳中。有的文件系统还保存文件的创建时间，最后访问时间及属性修改时间。（不过大多数早期的文件系统不记录文件的时间信息）其它信息还包括文件设备类型（如：区块数，字符集，套接口，子目录等等），文件所有者的ID，组ID，还有访问权限（如：只读，可执行等等）。

安全访问

针对基本文件系统操作的安全访问可以通过访问控制列表或capabilities实现。研究表明访问控制列表难以保证安全，这也就是研发中的文件系统倾向于使用 capabilities 的原因。然而目前多数商业性的文件系统仍然使用访问控制列表。

文件系统的类型

文件系统可以分为磁盘文件系统、网络文件系统和特殊文件系统。

磁盘文件系统

磁盘文件系统是一种设计用来利用数据存储设备来保存计算机文件的文件系统，最常用的数据存储设备是磁盘驱动器，可以直接或者间接地连接到计算机上。例如：FAT、exFAT、NTFS、HFS、HFS+、ext2、ext3、ext4、ISO 9660、ODS-5和UDF。 有些文件系统是进程文件系统（也有译作日志文件系统）或者追踪文件系统。

**FAT（文件配置表，File Allocation Table）**

在1977年由比尔·盖茨和马斯·麦当劳为了管理软盘而设计。在1980年被添·彼得逊的86-DOS操作系统采用。

FAT有一个严重的缺点：当文档删除后写入新数据，FAT不会将文件整理成完整片段再写入，长期使用后会使文件数据变得逐渐分散，而减慢了读写速度。软盘重组是一种解决方法，但必须经常重组来保持FAT文档系统的效率。

FAT12

初期的FAT就是现在俗称的FAT12。作为软盘的文档系统，它有几项限制：不支持分层性结构，簇寻址只有12位（这使得控制FAT有些棘手）而且只支持最多32M（2^16）的分区。

当时入门级的磁盘是5.25"、单面、40磁道、每个磁道8个扇区、容量略少于160KB。上面的限制超过了这个容量一个或几个数量级，同时允许将所有的控制结构放在第一个磁道，这样在读写操作时移动磁头。这些限制在随后的几年时间里被逐步增大。

由于唯一的根目录也必须放在第一个磁道，能够存放的文档个数就限制在了几十个。

目录

为了很好地支持以带有10MB硬盘为特色的IBM PC XT新计算机，在1983年年初MS-DOS 2.0几乎与该计算机在同时发布，它引进了层次目录结构。除了允许更好地组织文档外，目录允许在硬盘上存储更多的文档，这是因为最大文档个数不再受制于（仍然是固定的）根目录大小。这个数目现在能够等同于簇的数目（甚至更大，这是考虑到长度为0的文档并不占据任何FAT簇）。

FAT本身的格式并没有改变。PC XT的10MB的硬盘有4KB大小的簇。如果后来安装了一个20MB的硬盘，并且使用MS-DOS 2.0格式化，最后的簇大小将变为8KB，硬盘容量将变为15.9MB。

FAT16的开始

在1984年，IBM发布PC AT，内含一个20 MB的硬盘。微软公司也同步发布了MS-DOS 3.0。簇集地址增加至16位，允许更大数量的簇（最大 65,517），所以有更大的文件系统大小。但是，最大数量扇区及最大分区区（相当于磁盘）的大小仍是32 MB。所以，尽管技术上已经是“FAT16”，这种格式并不是我们今天常见到的这个名字所代表的格式。在MS-DOS 3.0格式化一个20 MB的硬盘，这硬盘将不能被MS-DOS 2.0或之前的版本所访问。当然，MS-DOS 3.0 仍然可访问MS-DOS 2.0的格式（8 KB簇的分区区）。

MS-DOS 3.0也开始支持高密度1.2MB 5.25"磁盘，最著名的是每个磁道有15个扇区，这样就允许FAT有更大的空间。这或许促进了一个对于簇大小的不确定的优化，簇大小从2个扇区减到1个。这样做的最后结果是高密度磁盘比旧的双密度磁盘的速度大幅度降低。

扩展分区和逻辑驱动器

除了改进FAT文件系统本身的结构之外，另一个提高FAT存储空间的方式是支持多个磁盘分区。最初，受限于主引导分区中文档分配表的固定结构一个硬盘最多只能切出多达4个分区。然而，由于DOS设计要求只能有一个分区标识为“活动的（Active）”，它也是主引导代码启动所用的分区。使用DOS工具不可能创建几个“主”DOS分区，并且第三方的工具也至少会警告这样一个机制将与DOS不兼容。

为了用一种兼容的方式使用更多的分区，一种新的分区类型被开发出来（1986年1月的MS-DOS 3.2），扩展分区它实际上是另外称为逻辑分区的一个容器。最初它里面只允许有一个逻辑分区、支持最大64MB的硬盘。在MS-DOS 3.3（1987年8月）这个限制更改到24个分区；它可能来自于强制性的C:-Z：的磁盘命名规则。逻辑分区表使用盘上的数据结构来描述，可能是为了简化编码它与主引导记录非常相似，并且它们组织成类似于俄罗斯套娃那样的结构。一颗硬盘中只能有一个扩展分区。

在扩展分区观念导入之前，一些硬盘控制器（当时采用独立的硬盘控制卡，IDE标准尚未出现）能够将大硬盘显示为两个独立的硬盘。有一些软件，如Ontrack[1]的磁盘管理员同样可以做到。

最终的FAT16

1987年11月，我们今天称为FAT的格式，最终到来，它在康柏 DOS 3.31中去掉了磁盘扇区的16位计数器。这个结果曾经一度被称为DOS 3.31大文件系统。尽管看起来磁盘上的变动很小，这个DOS的磁盘代码都必须检查并转换到32位的扇区数，由于它全部是16位的汇编语言这样一个现实，这项工作就变得非常复杂。

1988年，这项改进通过MS-DOS 4.0得到广泛应用。现在分区大小受限于每个簇的8位有符号扇区计数，它最大能达到2的64次方，对于一个常用的有32KB个簇每扇区512字节的硬盘来说，将FAT16分区大小的“明显”限制扩充到2GB。在磁光盘媒体上，它能使用1或者2KB的扇区，这样大小限制也就成比例地增大。

后来，视窗NT通过将每个簇的扇区数当作无符号数将最大的簇大小增加到64KB。然而这个格式与当时其它的任何FAT实现都不兼容并且产生了大量内部碎片。视窗98也支持这种格式的读写操作但是它的磁盘管理工具不支持这种格式。

长文件名（VFAT, LFN）

Windows 95设计人员的一个用户体验目标就是：除了传统的8.3文件名以外，在新操作系统中使用长文件名（LFN）。长文件名通过在目录条目排列时，使用一个工作区来实现（参见下面）。按照Windows 95VxD设备驱动程序的命名规则，这个新扩充的文件系统通常称为VFAT。

有意思的是，VFAT驱动在早于Windows 95的Windows for Groups 3.11中就已经出现，但它仅仅用于实现32位文档访问，一个绕过DOS的视窗自带高性能保护模式文档管理系统，它能够直接使用BIOS或者更好的32位磁盘访问，如视窗自带的保护模式磁盘驱动程序。它是一个后门；微软为Windows for Groups 3.11所作的广告说32位文档访问基于“芝加哥项目的32位文件系统”。

在Windows NT中，FAT文件系统对于长文件名的支持从3.5版就已经开始了。

FAT32

为了解决FAT16对于卷大小的限制同时让DOS的真实模式在非必要情况下不减少可用常规内存状况下处理这种格式，微软公司决定实施新一代的FAT，它被称为FAT32，带有32位的簇数，目前用了其中的28位。

理论上，这将支持总数达268,435,438（<228）的簇，允许磁盘容量达到2字节。然而，由于微软公司scandisk工具的限制，FAT不能大于4,177,920（<222）个簇，这将卷的容量限制在了124.55GB，除非不再使用“scandisk”。

FAT32随着视窗95 OSR2发布，尽管需要重新格式化才能使用这种格式并且DriverSpace 3（视窗95 OSR2和视窗98所带版本）从来都不支持这种格式。视窗98提供了一个工具用来在不丢失数据的情况下将现有的硬盘从FAT16转到FAT32格式。在NT产品线上对于它的支持从视窗2000开始。

视窗2000和视窗XP能够读写任何大小的FAT32文件系统，但是这些平台上的格式化程序只能创建最大32GB的FAT32文件系统。Thompson and Thompson (2003)写道“奇怪的是微软公司说这种现象是故意设计的”微软公司知识库文章 184006的确是这么说的，但是没有提出任何关于这个限制的合理解释。Peter Norton的观点是“微软公司在有意地削弱FAT32文件系统”。

exFAT

在Windows Embedded CE 6.0 中引入，Windows Vista SP1也引入了exFAT的支持。在很多方面exFAT有了相当大的改进，但兼容性是exFAT推广的障碍。

第三方支持

其它IBM PC的可选操作系统—如Linux、FreeBSD和BeOS—都支持FAT格式，并且大部分都在相应的视窗版本发布以后很快就支持VFAT和FAT32格式。早期的Linux发布版本还包括称为UMSDOS的格式，它是保存在一个独立的称为--linux-.——的带有Unix文档属性（如长文件名和访问许可）的FAT。UMSDOS在VFAT发布以后就不再使用。Linux内核从2.5.7开始就禁止了这项功能。Mac OS X操作系统在除启动盘之外的其它卷上也支持FAT文件系统。

FAT和其它数据流

FAT文件系统本身不是为支持ADS（资源分支）而设计的，但是一些高度依赖它们的操作系统创造出不同的方法在FAT驱动器上处理它们。这些方法或者在额外的文档或路径中存储附加的信息（Mac OS），或者给那些磁盘数据结构中以前没有使用的变量赋予新的含义（OS/2和视窗NT）。第二种设计，尽管想象起来会更有效率，但是它们不能被不认识这种格式的工具复制或者备份；使用不能识别这种格式的磁盘工具（如碎片整理或CHKFSK）控制这些磁盘可能会破坏这些信息。

Mac OS使用PC Exchange存储不同的数据，文档属性和文件名存在一个名为FINDER.DAT的隐藏文档中，资源分支（ADS）存在名为RESSOURCE.FRK的子目录中，这些数据都存在使用它们的每个目录中。从PC Exchange 2.1开始，它们将Mac OS的长文件名保存为标准的FAT长文件名，并且将超过31个字符的FAT长文件名转换为唯一的31字符能够被Macintosh应用程序识别的文件名。

Mac OS X将元数据（资源分支、不同的ADS、文档属性）保存在与所有人相同并以“。\_”开始的名字的隐藏文档中，并且Finder将一些文件夹和文档元数据存在名为“.DS\_Store”的隐藏文档中。

OS/2高度依赖于扩展属性（EA）并且将它们存在位于FAT12或FAT16的根目录下名为“EA DATA. SF”的隐藏文档中。这个文档使用以前文档（或者目录）的目录清单中的两个保留字节索引。在FAT32格式中，这些字节中存有文档或者目录开始簇号的高16位，这样就使它难于在FAT32上保存EA。扩展属性可以通过Workplace Shell桌面、REXX脚本、许多系统图形用户接口和命令行工具（如4OS2来访问。

Windows NT支持HPFS、NTFS和FAT中所有扩展属性的处理（所用处理机制完全类似于OS/2），但是不能处理其它一些存于NTFS驱动器的ADS数据。试图从复制带有与NTFS驱动器属性不同扩展属性的ADS到FAT驱动器将报告一个警告信息提示ADS将会丢失。

Windows 2000以后产品的处理类似于视窗NT但复制到FAT32时它们没有显示任何警告信息直接丢弃扩展属性（但报告其它象“Macintosh Finder Info”和“Macintosh Resource Fork”这些ADS引起的警告）。

设计

主磁盘结构

主引导区 文档分配表 #1 文档分配表 #2 根目录 其他所有数据...剩下磁盘空间

一个FAT文件系统包括四个不同的部分。

1.保留扇区，位于最开始的位置。第一个保留扇区是引导区（分区启动记录）。它包括一个称为基本输入输出参数块的区域（BPB，BIOS Parameter Block，包括一些基本的文件系统信息尤其是它的类型和其它指向其它扇区的指针），通常包括操作系统的启动调用代码。保留扇区的总数记录在引导扇区中的一个参数中。引导扇区中的重要信息可以被DOS和OS/2中称为驱动器参数块的操作系统结构访问。

2.FAT区域。它包含有两份文档分配表，这是出于系统冗余考虑，尽管它很少使用，即使是磁盘修复工具也很少使用它。它是分区信息的映射表，指示簇是如何存储的。

3.根目录区域。它是在根目录中存储文档和目录信息的目录表。在FAT32下它可以存在分区中的任何位置，但是在早期的版本中它永远紧随FAT区域之后。

4.数据区域。这是实际的文档和目录数据存储的区域，它占据了分区的绝大部分。通过简单地在FAT中添加文档链接的个数可以任意增加文档大小和子目录个数（只要有空簇存在）。然而需要注意的是每个簇只能被一个文档占有，这样的话如果在32KB大小的簇中有一个1KB大小的文档，那么31KB的空间就浪费掉了。

启动扇区

|  |  |  |  |
| --- | --- | --- | --- |
| **偏移 （字节）** | **长度 （字节）** | **说明** | |
| 0x00 | 3 | 跳转指令（跳过开头一段区域） | |
| 0x03 | 8 | OEM名称（空格补齐）。 MS-DOS检查这个区域以确定使用启动记录中的哪一部分数据。常见值是IBM 3.3（在“IBM”和“3.3”之间有两个空格）和MSDOS5.0. | |
| 0x0b | 2 | 每个扇区的字节数。*基本输入输出系统参数块*从这里开始。（512） | |
| 0x0d | 1 | 每簇扇区数，扇区数/簇。（1） | |
| 0x0e | 2 | 保留扇区数（包括启动扇区）（1） | |
| 0x10 | 1 | 文档分配表数目（2） | |
| 0x11 | 2 | 最大根目录条目个数（224） | |
| 0x13 | 2 | 总扇区数（如果是0，就使用偏移0x20处的4字节值）（2880） | |
| 0x15 | 1 | 介质描述　（0xf0） | |
| 0xF8 | 单面、每面80磁道、每磁道9扇区 |
| 0xF9 | 双面、每面80磁道、每磁道9扇区 |
| 0xFA | 单面、每面80磁道、每磁道8扇区 |
| 0xFB | 双面、每面80磁道、每磁道8扇区 |
| 0xFC | 单面、每面40磁道、每磁道9扇区 |
| 0xFD | 双面、每面40磁道、每磁道9扇区 |
| 0xFE | 单面、每面40磁道、每磁道8扇区 |
| 0xFF | 双面、每面40磁道、每磁道8扇区 |
| 同样的介质描述必须在重复复制到每份FAT的第一个字节。有些操作系统（MSX-DOS 1.0版）全部忽略启动扇区参数，而仅仅使用FAT的第一个字节的介质描述确定文件系统参数。 | |
| 0x16 | 2 | 每个文档分配表的扇区（FAT16）（9） | |
| 0x18 | 2 | 每磁道的扇区（18） | |
| 0x1a | 2 | 磁头数（2） | |
| 0x1c | 4 | 隐藏扇区（0） | |
| 0x20 | 4 | 总扇区数（如果超过65535，参见偏移0x13）2880 | |
| 0x24 | 4 | 每个文档分配表的扇区（FAT32）。*扩展基本输入输出系统参数块*从这里开始。（0） | |
| 0x24 | 1 | 物理驱动器个数（FAT16） | |
| 0x25 | 1 | 当前磁头（FAT16） | |
| 0x26 | 1 | 签名（FAT16） | |
| 0x27 | 4 | ID (FAT16) | |
| 0x28 | 2 | Flags (FAT32) | |
| 0x2a | 2 | 版本号 (FAT32) | |
| 0x2c | 4 | 根目录启始簇 (FAT32) | |
| 0x2b | 11 | 卷标（非FAT32） | |
| 0x30 | 2 | FSInfo 扇区 (FAT32) | |
| 0x32 | 2 | 引导扇区备份 (FAT32) | |
| 0x34 | 12 | 保留未使用 (FAT32) | |
| 0x36 | 8 | FAT文件系统类型（如FAT、FAT12、FAT16） | |
| 0x3e | 2 | 操作系统自引导代码 | |
| 0x40 | 1 | BIOS设备代号 (FAT32) | |
| 0x41 | 1 | 未使用 (FAT32) | |
| 0x42 | 1 | 标记 (FAT32) | |
| 0x43 | 4 | 卷序号 (FAT32) | |
| 0x47 | 11 | 卷标（FAT32） | |
| 0x52 | 8 | FAT文件系统类型（FAT32） | |
| 0x1FE | 2 | 扇区结束符（0x55 0xAA） | |

这里描述的启动扇区能在如OS/2 1.3的启动盘上看到。早期的版本使用一个较短的基本输入输出系统参数块，它们的启动代码在前面开始（如OS/2 1.1中是偏移0x2b）。

例外情况

Apricot PC的MS-DOS所用FAT的实现有一个不同的启动扇区组织以使用计算机与IBM不兼容的基本输入输出系统。跳转指令和OEM名被省略并且MS-DOS文件系统参数位于0x50（在标准扇区中偏移为0x0B - 0x17）。后来的Apricot MS-DOS版本除了Apricot特有的引导区之外也具有了读写标准启动分区的能力。

BBC Master 512 上的DOS Plus根本就不使用传统的引导区。数据磁盘省略了引导区并且以一个单份的FAT开始（FAT的第一个字节用来确定磁盘容量），启动磁盘使用一个包含启动调用程序的小型ADFS文件系统，后面跟随一个单份的FAT。

**文档分配表**

一个分区分成同等大小的**簇**，也就是连续空间的小块。簇的大小随着FAT文件系统的类型以及分区大小而不同，典型的簇大小介于2KB到32KB之间。每个**文档**根据它的大小可能占有**一个或者多个簇**；这样，一个文档就由这些这些（称为单链表）簇链所表示。然而，这些链并不一定一个接着一个在磁盘上存储，它们经常是在整个数据区域零散的储存。

文档分配表（FAT）是映射到分区每个簇的条目列表。每个条目记录下面五种信息中的一种。

链中下一个簇的地址

一个特殊的文档结束符（EOF）符号指示链的结束

一个特殊的符号标示坏簇

一个特殊的符号标示保留簇

0来表示空闲簇

每个版本的FAT文件系统使用不同大小的FAT条目。这个大小已经由名字表示出来，例如FAT16文件系统的每个条目使用16位表示，32位文件系统使用32位表示。这个不同意味着FAT32系统的文档分配表能比FAT16映射更多的簇，它也允许FAT32有更大的分区大小。这也使得FAT32比FAT16更能有效地利用磁盘空间，因为每个驱动器能够寻址更小的簇，这也就意味着更少的空间浪费。

FAT条目值：

FAT12 FAT16 FAT32 描述

0x000 0x0000 0x?0000000 空闲簇

0x001 0x0001 0x?0000001 保留簇

0x002 - 0xFEF 0x0002 - 0xFFEF 0x?0000002 - 0x?FFFFFEF 被占用的簇；指向下一个簇

0xFF0 - 0xFF6 0xFFF0 - 0xFFF6 0x?FFFFFF0 - 0x?FFFFFF6 保留值

0xFF7 0xFFF7 0x?FFFFFF7 坏簇

0xFF8 - 0xFFF 0xFFF8 - 0xFFFF 0x?FFFFFF8 - 0x?FFFFFFF 文档最后一个簇

注意FAT32只使用32位中的28位。高4位通常是0但它们是保留位，不要更改它们。在上面的表中它们用问号表示。

目录表

目录表是一个表示目录的特殊类型文档（现今通常称为文件夹）。它里面保存的每个文档或目录使用表中的32位条目表示。每个条目记录名字、扩展名、属性（档案、目录、隐藏、只读、系统和卷）、创建的日期和时间、文档／目录数据第一个簇的地址，最后是文档／目录的大小。

除了FAT12和FAT16文件系统中的根目录表占据特殊的根目录区域位置之外，所有其它的目录表都存在数据区域。

合法的DOS文件名包括下面一些字符：

大写字母A-Z

数字0-9

空格（尽管结尾的空格被作为填充而不是文件名的一部分）

! # $ % & ( ) - @ ^ \_ ` { } ~ '

数值 128-255

DOS文件名位于OEM字符集。

位于根目录区域和子目录区域的目录条目都是下面的格式：

长文件名（LFN）使用一个技巧存储在FAT文件系统上——在目录表中添加假的条目。这些条目使用一个普通文档无法使用的卷标属性标识，普通文档无法使用是由于它们被大多数旧的MS-DOS程序忽略。很显然，一个只包含卷标的目录被当作空卷，这样就允许删除；使用长文件名创建的文档在从普通的DOS删除就会发生这样的情形。

校验和也允许检验长文件名是否与8.3文件名匹配；当一个文档删除之后使用DOS在同一个目录位置重新创建之后就会出现不匹配现象。校验和使用下面的算法计算。（注意pFcbName是指向如正常目录条目中所显示的文件名的指针，例如前八个字符是文件名，最后三个是扩展名。点是隐含的。文件名中没有使用的空间将使用空格（ASCII 0x20）补齐。例如，“Readme.txt”将记录为"README TXT"。

旧版的PC-DOS错误地将根目录中的长文件名当作卷标，这样它们就会显示错误的卷标。

每个假条目包含13UTF-16个字符（26字节），通过使用包含文档大小或者时间记录的区域获得除了旧的8+3之外的另外15个字节（但是出于安全和磁盘检查工具的考虑开始簇的区域没有使用保留值为0）。参见8.3中另外的解释。

长文件名条目使用下面的格式：

如果一个文件名只包含小写字母、或者是一个小写字母的名加上大写扩展名的混合或者与此相反，没有特殊的字符并且满足8.3的限制，在视窗NT上就不创建VFAT的条目。相反，在目录条目的偏移0x0c处的没有说明的位用来指示文件名全部或者部分是小写字母。特别明确的是，位4意味着小写字母的扩展名，位3意味着名是小写字母，这样就允许如“example.TXT”和“HELLO.txt”这样的组合，但是不允许“Mixed.txt”这样的组合。很少有操作系统支持这种功能。非NT视窗版本当这个扩展使用时将把文件名当作大写字母。缺省情况下，Linux的最近版本将认识这个扩展但是在写时并不使用它。

第三方扩展

在微软公司添加长文件名和创建／访问时间戳之前，其它的操作系统使用目录表字节0x0C-0x15存储其它的元数据。它们包括：

-----------

磁盘的分区格式

FAT12概要

扇区(Sector)

簇(Cluster)

分区(Parition)

FAT16概要

FAT32概要

NTFS概要

MBR主引导记录

磁盘分区表（Disk Partition Table，DPT）

存储于447（1BE）开始的64字节中，共有4个表项，每个表项16字节，记录对应分区的信息。

字节 含义

1. ActiveFlag，活动标志。0x80H：活动分区，0x00H非活动分区

1 2 3 该分区的起始磁头号，扇区号，柱面号。1字节：磁头号（0x00-0xFF，0-255），2字节的低6位：扇区号（0x01-0x3F，1-63），2字节的高2位+3字节：柱面号（0x000-0x3FFF，0-1023）

4 分区文件系统标志，0x00H未使用，0x05H，0x0FH扩展分区，0x06HFAT16分区，0x0BH，0x1BH，0x0CH，0x1CHFAT32分区，0x07HNTFS分区

5 6 7 该分区的结束磁头号，扇区号，柱面号。含义同上。

8 9 10 11 逻辑起始扇区号，表示分区起点之前已用了的扇区数

12 13 14 15 该分区所占的扇区数

柱面（cylinder）

　　磁盘柱面：

　　硬盘最基本的组成部分是由坚硬金属材料制成的涂以磁性介质的盘片，不同容量硬盘的盘片数不等。每个盘片有两面，都可记录信息。盘片被分成许多扇形的区域，每个区域叫一个扇区，每个扇区可存储128×2的N次方（N＝0.1.2.3）字节信息。在DOS中每扇区是128×2的2次方＝512字节，盘片表面上以盘片中心为圆心，不同半径的同心圆称为磁道。硬盘中，不同盘片相同半径的磁道所组成的圆柱称为柱面。

磁道

　　磁道当磁盘旋转时，磁头若保持在一个位置上，则每个磁头都会在磁盘表面划出一个圆形轨迹，这些圆形轨迹就叫做磁道。磁盘上的磁道是一组记录密度不同的同心圆。磁表面存储器是在不同形状(如盘状、带状等)的载体上，涂有磁性材料层，工作时，靠载磁体高速运动，由磁头在磁层上进行读写操作，信息被记录在磁层上，这些信息的轨迹就是磁道。磁盘的磁道是一个个同心圆，磁带的磁道是沿磁带长度方向的直线，这些磁道用肉眼是根本看不到的，因为它们仅是盘面上以特殊方式磁化了的一些磁化区，磁盘上的信息便是沿著这样的轨道存放的。相邻磁道之间并不是紧挨着的，这是因为磁化单元相隔太近时磁性会产生相互影响，同时也为磁头的读写带来困难。

扇区

磁盘上的每个磁道被等分为若干个弧段，这些弧段便是磁盘的扇区。

简介

　　磁盘驱动器在向磁盘读取和写入数据时，要以扇区为单位。在磁盘上，DOS操作系统是以“簇”为单位为文件分配磁盘空间的。硬盘的簇通常为多个扇区，与磁盘的种类、DOS 版本及硬盘分区的大小有关。每个簇只能由一个文件占用，即使这个文件中有几个字节，决不允许两个以上的文件共用一个簇，否则会造成数据的混乱。这种以簇为最小分配单位的机制，使硬盘对数据的管理变得相对容易，但也造成了磁盘空间的浪费，尤其是小文件数目较多的情况下，一个上千兆的大硬盘，其浪费的磁盘空间可达上百兆字节。

坏扇区

　　在硬盘中无法被正常访问或不能被正确读写的扇区都称为Badsector。一个扇区能存储512Bytes的数据，如果在某个扇区中有任何一个字节不能被正确读写，则这个扇区为Badsector。除了存储512Bytes外，每个扇区还有数十个Bytes信息，包括标识（ID）、校验值和其它信息。这些信息任何一个字节出错都会导致该扇区变“Bad”。例如，在低级格式化的过程中每个扇区都分配有一个编号，写在ID中。如果ID部分出错就会导致这个扇区无法被访问到，则这个扇区属于Badsector。有一些Badsector能够通过低级格式化重写这些信息来纠正。

引导扇区（Boot Sector）

----------

MBR编程

程序流程：

1. 将0：7C00H开始的N字节的代码复制到0：0600H开始的内存单元中。（BIOS）
2. 搜索可引导分区，即80H标志。

成功：GOTO 3

失败：跳入ROM BASIC

无效分区表：GOTO 5

1. 读引导扇区

失败：GOTO 5

成功：GOTO 4

1. 验证引导扇区最后是否为55AAH

失败：GOTO 5

成功：GOTO 6

1. 打印错误进入无限循环
2. 跳到0：7C00H进行下一步启动工作。

FDISK

FORMAT

BPB(BIOS Parameter Block)

FAT16的BPB（BIOS Parameter Block）表，描述逻辑盘结构组成，包含隐藏扇区数目（从0-1-1开始计算）、FAT扇区数、FAT拷贝数、硬盘磁头总数、根目录表项最大值等。

FAT32系统中，BPB表的偏移与FAT16不同，但表项基本相同。整个隐藏扇区部分都作为逻辑盘的描述区域。

硬盘BPB主要结构说明：

(Cylinder柱面/磁道-Side/Head磁头-Sector扇区地址以下简称为?-?-?)

主分区

名称 地址 长度(扇区)

主引导记录（Main Boot Record) C0-H0-S1 1

系统扇区（System Secotrs） C0-H0-S2 C0-H0-S63 62

引导扇区（Boot） C0-H1-S1 1

扩展分区

名称 地址 长度(扇区)

扩展分区（Extend Partition） C?-Hy-S1 1

系统扇区（System Secotrs） C?-Hy-S2 C?-Hy-S63 62

引导扇区（Boot） C?-H(y+1)-S1 1

其后各项与主分区相同……

隐藏扇区（Hidden Secotrs）:

FAT16 C0-H1-S1 1

FAT32 C0-H1-S1 32

文件分配表(File Allocation Table):

FAT16 C0-H1-S2 根据逻辑盘容量变化

FAT32 C0-H1-S33 根据逻辑盘容量变化

说明：

1. FAT16的每个表项由2字节（16位）组成，通常每个表项指向的簇包含64个扇区，即32K字节。 逻辑盘容量最大为2047MB。

2. FAT32的每个表项由4字节（32位）组成，通常每个表项指向的簇包含8个扇区，即4K字节。逻辑盘容量最小为512MB。

3. 对于C分区，在MBR的偏移01c2H处，FAT16为06H，FAT32为0CH。

DBR

　　（1）DBR位于柱面0，磁头1，扇区1，其逻辑扇区号为0

　　（2）DBR包含DOS引导程序和BPB。

　　（3）BPB十分重要，由此可算出逻辑地址与物理地址。

　　以上仅DOS（FAT16）为例，由于DOS（FAT16）已经退出历史舞台，但现在0磁道1柱面1扇区这个位置仍然是起着类似的作用，所以准确地说，DBR应该改称为 OBR（OS Boot Record）即操作系统引导扇区，如WINXP的OBR（FAT32或NTFS）就是在DOS的DBR基础逐步演变而来的。OBR（DBR）是高级格式化程序产生的，如FORMAT，PM，DM，DISKPART，WINXP磁盘管理器。

常见问题

　　DBR（DOS BOOT RECORD，DOS引导记录），位于柱面0，磁头1，扇区1，即逻辑扇区0。DBR分为两部分：DOS引导程序和BPB（BIOS参数块）。其中DOS引导程序完成DOS系统文件（IO.SYS，MSDOS.SYS）的定位与装载，而BPB用来描述本DOS分区的磁盘信息，BPB位于DBR偏移0BH处，共13字节。 它包含逻辑格式化时使用的参数，可供DOS计算磁盘上的文件分配表，目录区和数据区的起始地址，BPB之后三个字提供物理格式化（低格）时采用的一些参数。引导程序或设备驱动程序根据这些信息将磁盘逻辑地址（DOS扇区号）转换成物理地址（绝对扇区号）。

相关数据参数

　　BPB格式 序号 偏移地址 意义

　　1 03H－0AH OEM号

　　2 0BH－0CH 每扇区字节数

　　3 0DH 每簇扇区数

　　4 0EH－0FH 保留扇区数

　　5 10H FAT备份数

　　6 11H－12H 根目录项数

　　7 13H－14H 磁盘总扇区数

　　8 15H 描述介质

　　9 16H－17H 每FAT扇区数

　　10 18H－19H 每磁道扇区数

　　11 1AH－1BH 磁头数

　　12 1CH－1FH 特殊隐含扇区数

　　13 20H－23H 总扇区数

　　14 24H－25H 物理驱动器数

　　15 26H 扩展引导签证

　　16 27H－2AH 卷系列号

　　17 2BH－35H 卷标号

　　18 36H－3DH 文件系统号

DOS引导记录公式

　　DOS引导记录公式：

　　文件分配表≡保留扇区数

　　根目录≡保留扇区数＋FAT的个数×每个FAT的扇区数

　　数据区≡根目录逻辑扇区号＋（32×根目录中目录项数＋（每扇区字节数－1））DIV每扇区字节数

　　绝对扇区号≡逻辑扇区号＋隐含扇区数

　　扇区号≡（绝对扇区号MOD每磁道扇区数）＋1

　　磁头号≡（绝对扇区号DIV每磁道扇区数）MOD磁头数

　　磁道号≡（绝对扇区号DIV每磁道扇区数）DIV磁头数

OBR

　　OBR（OS Boot Record）即操作系统引导扇区，通常位于硬盘的0磁道1柱面1扇区（这是对于DOS来说的，对于那些以多重引导方式启动的系统则位于相应的主分区/扩展分区的第一个扇区），是操作系统可直接访问的第一个扇区，它也包括一个引导程序和一个被称为BPB（BIOS Parameter Block）的本分区参数记录表。其实每个逻辑分区都有一个OBR，其参数视分区的大小、操作系统的类别而有所不同。引导程序的主要任务是判断本分区根目录前两个文件是否为操作系统的引导文件（例如MSDOS或者起源于MSDOS的Win9x/Me的IO.SYS和MSDOS.SYS）。如是，就把第一个文件读入内存，并把控制权交予该文件。BPB参数块记录着本分区的起始扇区、结束扇区、文件存储格式、硬盘介质描述符、根目录大小、FAT个数、分配单元（Allocation Unit，以前也称之为簇）的大小等重要参数。OBR由高级格式化程序产生（例如DOS 的Format.com）。

第1扇区：引导扇区512B

第2-10扇区：FAT表9\*512B

FAT表项，1.5字节，0，1不用，从2开始。

第11-19扇区：FAT表9\*512B

第20-33扇区：根目录 14\*512B，由目录条目组成，每个目录条目的大小是32字节。最大条目个数在BPBRootEntCnt中定义（224=0xE0）。224\*32=7168字节。

DIRName 0，0xB

Dirattr，0xB，1

Resaved，0xC， 10

DirWrtTime，0x16，2

DirWrtDate，0x18，2

DirFrstClus，0x1A，2

DirFileSize，0X1C，4

第34-2880扇区：数据区。

--------------------

基本输入输出系统(Basic Input/Output System, BIOS)

--------------------

BIOS是一组固化到计算机内主板上一个ROM芯片上的程序，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机后自检程序和系统自启动程序。 其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。

固化在ROM中的基本输入输出系统包含了主要IO设备的处理程序和许多常用例行程序，它们一般以中断处理程序的形式存在.BIOS直接建立在硬件基础上。

BIOS这个字眼是第一次由CP/M操作系统中出现，描述在开机阶段加载CP/M与硬件直接沟通的部份。（CP/M机器通常只有ROM里面的一个简单开机加载程序）

BIOS技术源于IBM PC/AT机器的流行以及第一台由康柏公司研制生产的“克隆”PC。在PC启动的过程中，BIOS担负著初始化硬件，检测硬件功能，以及引导操作系统的责任。在早期，BIOS还提供一套运行时的服务程序给操作系统及应用程序使用。BIOS程序存放于一个断电后内容不会丢失的只读存储器中；系统过电或被重置 (reset)时，处理器第一条指令的地址会被定位到BIOS的存储器中，让初始化程序开始运行。英特尔公司从2000年开始，发明了可扩展固件接口（Extensible Firmware Interface），用以规范BIOS的开发。而支持EFI规范的BIOS也被称为EFI BIOS。之后为了推广EFI，业界多家著名公司共同成立了统一可扩展固件接口论坛（UEFI Forum），英特尔公司将EFI 1.1规范贡献给业界，用以制订新的国际标准UEFI规范。目前UEFI规范的最新版本是2.1b，而根据来自英特尔公司的预测，到2010年，全世界将有60%以上的个人电脑使用支持UEFI规范的BIOS产品。

磁盘操作系统DOS(Disk Operating System)建立在BIOS的基础上,通过BIOS操纵控制硬件.

应用程序

DOS

BIOS

外设硬件接口

注意：尽量减少应用程序对硬件的依赖

输入设备：键盘，鼠标等

输出设备：显示器，打印机等

BIOS供应商：

Award Software与General Software均被Phoenix Technologies收购

Microid Research被Unicore Software收购

SystemSoft被Insyde Software收购。

Phoenix Technologies，美国凤凰科技，全球最大的BIOS供应商。

American Megatrends，美国安迈科技，已被凤凰科技超过，曾经是最大的BIOS供应商。

Insyde Software，台湾系微公司。

Byosoft，新兴厂商，中国南京的百敖软件公司。

未来发展趋势

2006年，微软曾在WinHEC会议上表示UEFI将会取代已经使用了20多年的BIOS。这两个技术都在电脑启动的时候发出第一个命令指示，并使得操作系统能够被顺利加载。

主要功能

BIOS可以被视为专门与硬件沟通的微型操作系统，通常是使用针对兼容CPU的汇编语言写成的。（参考CMOS）

开机自我测试：检查中央处理器及各控制器之状态是否正常。

初始化动作：针对存储器、主板芯片组、显卡及周边设备做初始化动作。

纪录系统设置值：提供各组件的基本设置，如显示器、键盘、处理器、存储器等。

提供常驻程序库：提供操作系统或应用程序调用的中断常驻程序，如INT 10H等。

加载操作系统：从存储设备中加载操作系统到随机存取存储器。

功能

从功能上看，BIOS分为三个部分：

**自检及初始化**

这部分负责启动电脑，具体有三个部分：

第一个部分是用于电脑刚接通电源时对硬件部分的检测，也叫做加电自检（Power On Self Test，简称POST），功能是检查电脑是否良好，通常完整的POST自检将包括对CPU，640K基本内存，1M以上的扩展内存，ROM，主板，CMOS存储器，串并口，显示卡，软硬盘子系统及键盘进行测试，一旦在自检中发现问题，系统将给出提示信息或鸣笛警告。自检中如发现有错误，将按两种情况处理：对于严重故障（致命性故障）则停机，此时由于各种初始化操作还没完成，不能给出任何提示或信号；对于非严重故障则给出提示或声音报警信号，等待用户处理。

第二个部分是初始化，包括创建中断向量、设置寄存器、对一些外部设备进行初始化和检测等，其中很重要的一部分是BIOS设置，主要是对硬件设置的一些参数，当电脑启动时会读取这些参数，并和实际硬件设置进行比较，如果不符合，会影响系统的启动。

第三个部分是引导程序，功能是引导DOS或其他操作系统。BIOS先从软盘或硬盘的开始扇区读取引导记录，如果没有找到，则会在显示器上显示没有引导设备，如果找到引导记录会把电脑的控制权转给引导记录，由引导记录把操作系统装入电脑，在电脑启动成功后，BIOS的这部分任务就完成了。

**程序服务处理**

程序服务处理程序主要是为应用程序和操作系统服务，这些服务主要与输入输出设备有关，例如读磁盘、文件输出到打印机等。为了完成这些操作，BIOS必须直接与计算机的I／O设备打交道，它通过端口发出命令，向各种外部设备传送数据以及从它们那儿接收数据，使程序能够脱离具体的硬件操作。

硬件中断处理

硬件中断处理则分别处理PC机硬件的需求，BIOS的服务功能是通过调用中断服务程序来实现的，这些服务分为很多组，每组有一个专门的中断。例如视频服务，中断号为10H；屏幕打印，中断号为05H；磁盘及串行口服务，中断14H等。每一组又根据具体功能细分为不同的服务号。应用程序需要使用哪些外设、进行什么操作只需要在程序中用相应的指令说明即可，无需直接控制。

（二）（三）两部分虽然是两个独立的内容，但在使用上密切相关。这两部分分别为软件和硬件服务，组合到一起，使计算机系统正常运行。

另外需注意：**BIOS设置不当会直接损坏计算机的硬件，甚至烧毁主板，建议不熟悉者慎重修改设置。**

记录设置值

用户可以通过设置BIOS来改变各种不同的设置，比如onboard显卡的内存大小。

加载操作系统

用户手上所有的操作系统都是有BIOS转交给引导扇区，再由引导扇区转到各分区激活响应的操作系统。

启动电脑原理

当电脑的电源打开，BIOS就会由主板上的闪存（flash memory）运行，并将芯片组和存储器子系统初始化。BIOS会把自己从闪存中，解压缩到系统的主存；并且从那边开始运行。PC的BIOS代码也包含诊断功能，以保证某些重要硬件组件，像是键盘、磁盘设备、输出输入端口等等，可以正常运作且正确地初始化。几乎所有的BIOS都可以选择性地运行CMOS存储器的设置程序；也就是保存BIOS会访问的用户自定义设置数据（时间、日期、硬盘细节，等等）。IBM技术参考手册中曾经包含早期PC和AT BIOS的80x86源代码。 现代的BIOS可以让用户选择由哪个设备启动电脑，如光盘驱动器、硬盘、软盘、USB 闪存盘等等。这项功能对于安装操作系统、以LiveCD启动电脑、以及改变电脑找寻开机媒体的顺序特别有用。

有些BIOS系统允许用户可以选择要加载哪个操作系统（例如从第二颗硬盘加载其他操作系统），虽然这项功能通常是由第二阶段的开机管理程序（boot loader）来处理。

BIOS 中断调用是一组功能，提供DOS程式与一些软件（例如启动程式）去使用BIOS的功能。一些操作系统也使用BIOS来探索与初始化硬件资源在电脑启动的时候。

BIOS中断向量表

|  |  |
| --- | --- |
| **中断** | **描述** |
| INT 00h | CPU: Executed after an attempt to 除号为零 or when the quotient does not fit in the destination |
| INT 01h | CPU: Executed after every instruction while the trace flag is set |
| INT 02h | CPU: NMI, used e.g. by [POST](http://zh.wikipedia.org/w/index.php?title=Power-on_self-test&action=edit&redlink=1) for memory errors |
| INT 03h | CPU: The lowest non-reserved interrupt, it is used exclusively for debugging, and the INT 03 handler is always implemented by a debugging program |
| INT 04h | CPU: Numeric Overflow. Usually caused by the INTO instruction when the overflow flag is set. |
| INT 05h | Executed when Shift-Print Screen is pressed, as well as when the BOUND instruction detects a bound failure. |
| INT 06h | CPU: Called when the [Undefined Opcode](http://zh.wikipedia.org/w/index.php?title=Undefined_Opcode&action=edit&redlink=1) (invalid instruction) exception occurs. Usually installed by the operating system. |
| INT 07h | CPU: Called when an attempt was made to execute a floating-point instruction and no numeric coprocessor was available. |
| INT 08h | **IRQ0**: Implemented by the system timing component; called 18.2 times per second (once every 55 ms) by the PIC |
| INT 09h | **IRQ1**: Called after every key press and release (as well as during the time when a key is being held) |
| INT 0Bh | **IRQ3**: Called by serial ports 2 and 4 (COM2/4) when in need of attention |
| INT 0Ch | **IRQ4:** Called by serial ports 1 and 3 (COM1/3) when in need of attention |
| INT 0Dh | **IRQ5**: Called by hard disk controller (PC/XT) or 2nd parallel port LPT2 (AT) when in need of attention |
| INT 0Eh | **IRQ6**: Called by floppy disk controller when in need of attention |
| INT 0Fh | **IRQ7**: Called by 1st parallel port LPT1 (printer) when in need of attention |
| **INT 10h** | Video Services - installed by the BIOS or operating system; called by software programs   |  |  | | --- | --- | | AH=00h | 设定显示式 | | AH=01h | Set Cursor Shape | | AH=02h | 设定光标位置 | | AH=03h | Get Cursor Position And Shape | | AH=04h | Get Light Pen Position | | AH=05h | 设定显示页 | | AH=06h | 清除或卷轴画面(上) | | AH=07h | 清除或卷轴画面(下) | | AH=08h | Read Character and Attribute at Cursor | | AH=09h | Write Character and Attribute atCuror | | AH=0Ah | Write Character at Cursor | | AH=0Bh | Set Border Color | | AH=0Eh | Write Character in TTY Mode | | AH=0Fh | 取得目前显示模式 | | AH=13h | Write String | |
| INT 11h | Installed by the BIOS; returns equipment list |
| INT 12h | Installed by the BIOS or operating system; returns Conventional Memory Size |
| **INT 13h** | 低阶磁盘服务; installed by the BIOS or operating system; called by software programs   |  |  | | --- | --- | | AH=00h | Reset Disk Drives | | AH=01h | Check Drive Status | | AH=02h | Read Sectors From Drive | | AH=03h | Write Sectors To Drive | | AH=04h | Verifies Sectors n Dive | | AH=05h | Format Track On Drive | | AH=08h | Get Drive Parameters | | AH=09h | Init Fixed Drive Parameters | | AH=0Ch | Seek To Specified Track | | AH=0Dh | Reset Fixed Disk Controller | | AH=15h | Get Drive Type | | AH=16h | Get Floppy Drive Media Change Status | |
| INT 14h | Routines for communicating via the serial port. Used by software programs.   |  |  | | --- | --- | | AH=00h | Serial Port Initialization | | AH=01h | Transmit Character | | AH=02h | Receive Character | | AH=03h | Status | |
| INT 15h | Miscellaneous (System services support routines)   |  |  | | --- | --- | | AH=4FH | Keyboard Intercept | | AH=83H | Event Wait | | AH=84H | Read Joystick | | AH=85H | Sysreq Key Callout | | AH=86H | Wait | | AH=87H | Move Block | | AH=88H | Get Extended Memory Size | | AH=C0H | Get System Parameters | | AH=C1H | Get Extended BIOS Data Area Segment | | AH=C2H | Pointing Device Functions | | AH=E8h, AL=01h (AX = E801h) | Get Extended Memory Size(Newer function, since 1994). Gives results for memory size above 64 Mb. | | AH=E8h, AL=20h (AX = E820h) | Query System Address Map. The information returned from e820 supersedes what is returned from the older AX=E801h and AH=88h interfaces. | |
| INT 16h | Implemented by the BIOS or operating system. Provides routines to be called by software programs which communicate with the keyboard.   |  |  | | --- | --- | | AH=00h | Read Character | | AH=01h | Read Input Status | | AH=02h | Read Keyboard Shift Status | | AH=10h | Read Character Extended | | AH=11h | Read Input Status Extended | | AH=12h | Read Keyboard Shift Status Extended | |
| INT 17h | Print Services - used by software programs to communicate with the printer   |  |  | | --- | --- | | AH=00h | Print Character to Printer | | AH=01h | Initialize Printer | | AH=02h | Check Printer Status | |
| INT 18h | Execute Cassette BASIC: True IBM computers contain BASIC in the ROM to be interpreted and executed by this routine in the event of a boot failure (called by the BIOS) |
| INT 19h | After POST this interrupt is used by BIOS to load the operating system. |
| INT 1Ah | 即时的时钟（RTC）服务 - called by software programs to communicate with the RTC   |  |  | | --- | --- | | AH=00h | 读取 RTC | | AH=01h | 设定 RTC | | AH=02h | 读取 RTC 时间 | | AH=03h | 设定 RTC 时间 | | AH=04h | 读取 RTC 日期 | | AH=05h | 设定 RTC 日期 | | AH=06h | 设定 RTC Alarm | | AH=07h | Reset RTC Alarm | |
| INT 1Bh | Installed by the operating system; automatically called by INT 9 when Ctrl-Break has been pressed |
| INT 1Ch | Called automatically by INT 08; available for use by software programs when a routine needs to be executed regularly |
| INT 1Dh | Not to be called; simply a pointer to the VPT (Video Parameter Table), which contains data on video modes |
| INT 1Eh | Not to be called; simply a pointer to the DPT (Diskette Parameter Table), containing a variety of information concerning the diskette drives |
| INT 1Fh | Not to be called; simply a pointer to the VGCT (Video Graphics Character Table), which contains the data for ASCII characters 80h to FFh |
| INT 41h | Address pointer: FDPT = Fixed Disk Parameter Table (1st hard drive) |
| INT 46h | Address pointer: FDPT = Fixed Disk Parameter Table (2nd hard drive) |
| INT 4Ah | Called by RTC for alarm |
| INT 70h | **IRQ8**: 由 RTC 呼叫 |
| INT 74h | **IRQ12**: 由鼠标呼叫 |
| INT 75h | **IRQ13**: Called by math coprocessor |
| INT 76h | **IRQ14**: 由第一个 IDE 控制器所呼叫 |
| INT 77h | **IRQ15**: 由第二个 IDE 控制器所呼叫 |

BOOT编程

即引导扇区编程，引导扇取区，位于磁盘的第一个扇区，0柱面，0磁道，1扇区。对于硬盘而言，一个扇区可能的字节数为128×2n (n=0,1,2,3)。大多情况下，取n=2，即一个扇区(sector)的大小为512字节。一扇区的容量是512字节。

主引导扇区的读取流程

系统开机或者重启。

1.BIOS 加电自检 ( Power On Self Test -- POST )。BIOS执行内存地址为 FFFF:0000H 处的跳转指令，跳转到固化在ROM中的自检程序处，对系统硬件(包括内存)进行检查。

2.读取主引导记录(MBR)。当BIOS检查到硬件正常并与 CMOS 中的设置相符后，按照 CMOS 中对启动设备的设置顺序检测可用的启动设备。BIOS将相应启动设备的第一个扇区(也就是MBR扇区)读入内存地址为0000:7C00H 处。

3.检查0000:7DFEH-0000:7DFFH(MBR的结束标志位)是否等于 AA55H，若不等于则转去尝试其他启动设备(MBR的时候)，如果没有启动设备满足要求则显示"NO ROM BASIC"然后死机。

4.当检测到有启动设备满足要求后，BIOS将控制权交给相应启动设备。启动设备的MBR将自己复制到0000:0600H处, 然后从0000:0600H开始继续执行。

5.根据MBR中的引导代码启动引导程序。

引导扇区

引导扇区是硬盘、软盘或类似的数据存储设备的一个扇区，内含负责启动（booting）“存放在碟片（disk）的其他部份的程序（通常，但不必然是操作系统）”的机器码。

引导扇区有两种：

Master Boot Record(主引导记录) 是磁盘已被分区的第一个扇区，它包含定位活动分区区与唤起它的VBR的码。

Volume Boot Record 是磁盘未被分区的第一个扇区，或已分区的分区的第一个扇区，包含了加载与唤起操作系统(放在这个分区之内或放在这个磁盘上)的码。

IBM PC兼容机上，BIOS不在意VBR与MBR的不同，甚至分区。固件只是加载并运行磁盘的第一个扇区(sector)，在MBR里的码，才知道磁盘分区消息；且是负责加载启动活动分区的VBR的地方。

如果你从“一个没有灌操作系统的磁盘”启动，屏幕会显示"Please Insert a bootable disc and press a key"；这是引导扇区(boot sector)显示的，而不是机器的固件。

主引导记录

主引导记录（Master Boot Record，缩写：MBR），又叫做主引导扇区，是计算机开机后访问硬盘时所必须要读取的首个扇区，它在硬盘上的三维地址为（柱面，磁头，扇区）＝（0，0，1）。

主引导扇区记录着硬盘本身的相关信息以及硬盘各个分区的大小及位置信息，是数据信息的重要入口。如果它受到破坏，硬盘上的基本数据结构信息将会丢失，需要用繁琐的方式试探性的重建数据结构信息后才可能重新访问原先的数据。主引导扇区内的信息是通过FDISK写入的，它是低级格式化的产物，和操作系统没有任何关系(操作系统是建立在高级格式化的硬盘分区之上，是和一定的文件系统相联系的)。

对于硬盘而言，一个扇区可能的字节数为128×2n (n=0,1,2,3)。大多情况下，取n=2，即一个扇区(sector)的大小为512字节。

主引导记录的组成

启动代码

主引导记录最开头是第一阶段引导代码。其中的硬盘引导程序的主要作用是检查分区表是否正确并且在系统硬件完成自检以后将控制权交给硬盘上的引导程序（如GNU GRUB）。MBR是由分区程序（如Fdisk）所产生的，它不依赖任何操作系统，而且硬盘引导程序也是可以改变的，从而能够实现多系统引导。

硬盘分区表

硬盘分区表占据主引导扇区的64个字节(偏移01BEH--偏移01FDH)，可以对四个分区的信息进行描述，其中每个分区的信息占据16个字节。具体每个字节的定义可以参见硬盘分区结构信息。下面是一个例子：

如果某一分区在硬盘分区表的信息如下

80 01 01 00 0B FE BF FC 3F 00 00 00 7E 86 BB 00

则我们可以看到，最前面的"80"是一个分区的激活标志，表示系统可引导；"01 01 00"表示分区开始的磁头号为01，开始的扇区号为01，开始的柱面号为00；"0B"表示分区的系统类型是FAT32，其他比较常用的有04（FAT16）、07（NTFS）；"FE BF FC"表示分区结束的磁头号为254，分区结束的扇区号为63、分区结束的柱面号为764；"3F 00 00 00"表示首扇区的相对扇区号为63；"7E 86 BB 00"表示总扇区数为12289622。

结束标志字

结束标志字55，AA（偏移1FEH－偏移1FFH）是主引导扇区的最后两个字节，是检验主引导记录是否有效的标志。

主引导扇区的读取流程

系统开机或者重启。

1.BIOS 加电自检 ( Power On Self Test -- POST )。BIOS执行内存地址为 FFFF:0000H 处的跳转指令，跳转到固化在ROM中的自检程序处，对系统硬件(包括内存)进行检查。

2.读取主引导记录(MBR)。当BIOS检查到硬件正常并与 CMOS 中的设置相符后，按照 CMOS 中对启动设备的设置顺序检测可用的启动设备。BIOS将相应启动设备的第一个扇区(也就是MBR扇区)读入内存地址为0000:7C00H 处。

3.检查0000:7DFEH-0000:7DFFH(MBR的结束标志位)是否等于 AA55H，若不等于则转去尝试其他启动设备，如果没有启动设备满足要求则显示"NO ROM BASIC"然后死机。

4.当检测到有启动设备满足要求后，BIOS将控制权交给相应启动设备。启动设备的MBR将自己复制到0000:0600H处, 然后继续执行。

5.根据MBR中的引导代码启动引导程序。

主引导记录与硬盘分区

从主引导记录的结构可以知道，它仅仅包含一个64个字节的硬盘分区表。由于每个分区信息需要16个字节，所以对于采用MBR型分区结构的硬盘(其磁盘卷标类型为MS-DOS)，最多只能识别4个主要分区。所以对于一个采用此种分区结构的硬盘来说，想要得到4个以上的主要分区是不可能的。这里就需要引出扩展分区了。扩展分区也是主分区（Primary partition）的一种，但它与主分区的不同在于理论上可以划分为无数个逻辑分区。

扩展分区中逻辑驱动器的引导记录是链式的。每一个逻辑分区都有一个和MBR结构类似的扩展引导记录(EBR)，其分区表的第一项指向该逻辑分区本身的引导扇区，第二项指向下一个逻辑驱动器的EBR，分区表第三、第四项没有用到。

Windows系统默认情况下，一般都是只划分一个主分区给系统，剩余的部分全部划入扩展分区。这里有下面几点需要注意：

在MBR分区表中最多4个主分区或者3个主分区＋1个扩展分区，也就是说扩展分区只能有一个，然后可以再细分为多个逻辑分区。

在Linux系统中，硬盘分区命名为sda1－sda4或者hda1－hda4（其中a表示硬盘编号可能是a、b、c等等）。在MBR硬盘中，分区号1－4是主分区（或者扩展分区），逻辑分区号只能从5开始。

在MBR分区表中，一个分区最大的容量为2T，且每个分区的起始柱面必须在这个disk的前2T内。你有一个3T的硬盘，根据要求你至少要把它划分为2个分区，且最后一个分区的起始扇区要位于硬盘的前2T空间内。[3]如果硬盘太大则必须改用GPT。

MBR分区表与GPT分区表的关系

与支持最大卷为2 TB（Terabytes）并且每个磁盘最多有4个主分区（或3个主分区，1个扩展分区和无限制的逻辑驱动器）的MBR磁盘分区的样式相比，GPT磁盘分区样式支持最大卷为18 EB（Exabytes）并且每磁盘的分区数没有上限，只受到操作系统限制（由于分区表本身需要占用一定空间，最初规划硬盘分区时，留给分区表的空间决定了最多可以有多少个分区，IA-64版Windows限制最多有128个分区，这也是EFI标准规定的分区表的最小尺寸）。与MBR分区的磁盘不同，至关重要的平台操作数据位于分区，而不是位于非分区或隐藏扇区。另外，GPT分区磁盘有备份分区表来提高分区数据结构的完整性。

注释

1.对于一个操作系统而言，系统分区设为活动分区并不是必须的，这主要视引导程序而定，如果使用的引导程序是GRUB4DOS，MBR中的引导代码仅仅按照分区的顺序依次探测第二阶段引导器grldr的位置，并运行第一个探测到的grldr文件。

2.一个硬盘的分区个数还要受到分区大小的限制，因为硬盘是按照柱面分区的：一个分区至少要占一个柱面。但有一点需要注意，由于现在的硬盘结构已经和老式硬盘有了很大区别，其寻址结构也不再是CHS寻址，所以这里的柱面大小不同于相关软件显示的柱面大小。对于物理结构上有n个面的硬盘，其分区空间的最小值为：n × 扇区/磁道 × 512字节。

3.根据16字节分区表的结构：当前分区的扇区数用4个字节表示，前面各分区扇区数的总和也是4个字节，而232 × 512 ＝ 2 199 023 255 552 Byte

编程要点：

　　可执行文件（联接后的二进制文件）的大小为512B。

　　文件的最后两个字节的内容是AA55。

文件的位置特殊，软盘，0柱，0磁道，1扇区。

ES：0X0820

CH：0

DH：0

CL：2

BX：18\*2\*9-1（323=143H）

读磁盘：

MOV AH，0x02 ；功能号（2：读磁盘）

MOV AL， ；要读取的扇区数（Sectors Count）

MOV DH，0x00；磁头（Head）号

MOV DL，0x00；驱动器（Drive）号

（00H：A盘，01H：B盘，80H：第1磁盘，81H：第2磁盘）

MOV CH，0x00；磁道（Track）号

MOV CL，0x02；扇区（Sector）号

MOV ES，0x0820；

MOV BX，0x00；ES：BX数据缓冲区地址

INT 0x13 ；

返回值

CF：1：读取错误，0：正常。

AH：返回值

AL：读取扇区个数。

软盘

硬盘

CHS

简介

　　LBA(Logical Block Addressing)逻辑块寻址模式。在 LBA 模式下，我们知道硬盘上的一个数据区域由它所在的磁头、柱面（也就是磁道）和扇区所唯一确定。早期系统就是直接使用磁头柱面和扇区来对硬盘进行寻址（这称为CHS寻址），这需要分别存储每个区域的三个参数（这称为3D参数），使用时再分别读取三个参数，然后再在送到磁盘控制器去执行。由于系统用8b来存储磁头地址，用10b来存储柱面地址，用6b来存储扇区地址，而一个扇区共有512B，这样使用CHS寻址一块硬盘最大容量为256 \* 1024 \* 63 \* 512B = 8064 MB(1MB = 1048576B)（若按1MB=1000000B来算就是8.4GB）。随着硬盘技术的进步，硬盘容量越来越大，CHS模式无法管理超过8064 MB的硬盘，因此工程师们发明了更加简便的LBA寻址方式。在LBA地址中，地址不再表示实际硬盘的实际物理地址（柱面、磁头和扇区）。LBA编址方式将CHS这种三维寻址方式转变为一维的线性寻址，它把硬盘所有的物理扇区的C/H/S编号通过一定的规则转变为一线性的编号，系统效率得到大大提高，避免了烦琐的磁头/柱面/扇区的寻址方式。在访问硬盘时，由硬盘控制器再将这种逻辑地址转换为实际硬盘的物理地址。

硬盘模式

　　LARGE 大硬盘模式，在硬盘的柱面超过 1024 而又不为 LBA 支持时采用。LARGE 模式采用的方法是把柱面数除以 2，把磁头数乘以 2，其结果总容量不变。 　　在这三种硬盘模式中，现在 LBA 模式使用最多。 　　LBA与C/H/S 之间的转换: 　　设NS为每磁道扇区数，NH为磁头数，C、H、S分别表示磁盘的柱面、磁头和扇区编号，LBA表示逻辑扇区号，div为整除计算，mod为求余计算，则： 　　LBA=NH×NS×C+NS×H+S-1； 　　C=(LBA div NS)div NH； 　　H=(LBA div NS)mod NH； 　　S=(LBA mod NS)+1 　　例如 LBA = 0 则 CHS = 0/0/1 　　从C/H/S到LBA的计算公式： 　　LBA=（C-CS）\*PH\*PS+（H-HS）\*PS+(S-SS)

逻辑区块位址

逻辑区块位址(Logical Block Address, LBA)是描述电脑储存设备上资料所在区块的通用机制，一般用在像硬盘这样的辅助记忆装置。LBA可以意指某个资料区块的位址或是某个位址所指向的资料区块。现今电脑上所谓一个逻辑区块通常是512或1024字节。ISO-9660格式的标准CD则以2048字节为一个逻辑区块大小。LBA寻址机制

LBA是非常单纯的一种寻址模式﹔从0开始编号来定位区块，第一区块LBA=0，第二区块LBA=1，依此类推。这种寻址模式取代了原先操作系统必须面对储存设备硬件构造的方式。最具代表性的首推CHS(cylinders-heads-sectors，磁柱-磁头-磁区)寻址模式，区块必须以硬盘上某个磁柱、磁头、磁区的硬件位置所合成的位址来指定。CHS模式对硬盘以外的装置来说没什么作用(例如磁带或是网络储存设备)，所以通常也不会用在这些地方。过去MFM(Modified Frequency Modulation, 改良调频式)和RLL(Run Length Limited)储存设备都曾使用CHS模式，ATA-1装置更将延伸CHS(Extended Cylinders-Heads-Sectors, ECHS)也派上了用场。

SCSI采用LBA抽象寻址。实际上硬件控制器还是以CHS来寻址区块，但无论驱动程式还是任何以低阶存取磁盘的应用程式(例如数据库软件)通常都不再需要这个参数。各种要求区块低阶存取的系统呼叫把定义好的LBA传给驱动程式﹔最直接的情况下(逻辑装置与实体装置单一对应)驱动程式只是将LBA再传给硬件控制器。

LBA对应与逻辑装置虚拟化

当逻辑装置是经由虚拟化或是集合所构成的，像是RAID(磁盘阵列)和SANs(Storage Area Network)这种复杂的情况，就得把应用程式根据其观点中的磁盘来指定的LBA转换成每个实体储存设备上的LBA。在复杂的分散式储存结构下，从发出要求的应用程式到实体甚至是远端设备之间，会有太多这样的LBA转换。

CHS与LBA互换

CHS位址可用以下公式转成LBA,

#lba=(#c\*H+#h)\*S+#s-1

其中,

#c、#h、#s分别是磁柱、磁头、磁区的编号

#lba是逻辑区块编号

H=heads per cylinder，每个磁柱的磁头数

S=sectors per track，每磁道的磁区数

LBA可用以下公式对应到CHS:

#c=#lba/(S\*H)

#h=(#lba/S)%H

#s=(#lba%S)+1

其中,

/ 是整数除法

% 是取整数除法中的余数

请注意，当今的磁盘使用ZBR(Zone Bit Recording, 等密度记录)方式，实际的每轨磁区数得根据它是哪一轨。不过磁盘还是会提供这个参数来符合公式，内部再自动调整。

其它公式:

#lba/S=q 餘 r

#s=1+r

q/H=#c 餘 #h

例如:

CHS總數=[600, 10, 84]，求#lba=1234所對應的CHS編號:

1234/84=14 餘 58

#s=1+58=59

14/10=1 餘 4

#c=1

#h=4

#chs=(1, 4, 59)

驗算: (1\*10+4)\*84+59-1=14\*84+58=1234

LBA、ATA装置以及Enhanced BIOS

ATA-1规范中定义了28位元寻址模式，当成LBA或是CHS都可以。如果用CHS这28位元拆成: 磁柱16位元、磁头4位元、磁区8位元。注意CHS模式磁区是从1开始算，所以在这个规范中磁区数最多只有255个，最大磁区编号为254(0xFE)。

规范采用当时，CHS的BIOS规范只有24位元: 磁柱10位元、磁头8位元、磁区6位元，定义在BIOS的INT 13H软件中断里，而且已经用在DOS的MBR(Master Boot Record，主要开机记录)。这造成了BIOS CHS跟ATA CHS之间必须经过转换，否则各参数只能用到两者的最大公因子即CHS位元数={10, 4, 6}，也就是1024×16×63个磁区，以每磁区512字节计算得504 MiB。转换方式其一是Large模式或称Enhanced BIOS模式(又名Bit Shift Translation, 位移转换)，此方式会重新对应侦测到的磁柱和磁头数而磁区数不变﹔方式其二是将头一种CHS对应到LBA之后再换算成另一种CHS机制，称为LBA-assist。

即使利用这些转址方式，BIOS定给MS-DOS逻辑磁区(以及Windows NT 4.0硬盘分割区)的CHS位元数={10, 8, 6}机制顶多也只能达到7.84 GiB。以每磁区512字节来计算，ATA-1所定义的28位元LBA上限达到128 GiB。2002年ATA-6规范采用48位元LBA，同样以每磁区512字节计算容量上限可达128 Petabytes。

全局唯一标识分区表

全局唯一标识分区表（GUID Partition Table，缩写：GPT）是一个物理硬盘的分区结构。它是可扩展固件接口标准的一部分，用来替代BIOS中的主引导记录分区表。但因为MBR分区表不支持容量大于2.2TB（2.2 × 1012字节）的分区，所以也有一些BIOS系统为了支持大容量硬盘而用GPT分区表取代MBR分区表。[1]GPT分区表支持最多9.4ZB（9.4 × 1021字节）的硬盘和分区。

截止至2010年，常见操作系统对GPT的支持非常有限，见#操作系统支持。

特点

在MBR硬盘中，分区信息直接存储于主引导记录(MBR)中（主引导记录中还存储者系统的引导程序）。但在GPT硬盘中，分区表的位置信息储存在GPT头中。但出于兼容性考虑，硬盘的第一个扇区仍然用作MBR，之后才是GPT头。

跟现代的MBR一样，GPT也使用逻辑区块地址（LBA）取代了早期的CHS寻址方式。传统MBR信息存储于LBA 0，GPT头存储于LBA 1，接下来才是分区表本身。64位Windows操作系统使用16,384字节（或32扇区）作为GPT分区表，接下来的LBA 34是硬盘上第一个分区的开始。

苹果公司曾经警告说：[3]“不要假定所有设备的块大小都是512字节。”一些现代的存储设备如固态硬盘可能使用1024字节的块，一些磁光盘（MO）可能使用2048字节的扇区（但是磁光盘通常是不进行分区的）。一些硬盘生产商在计划生产4096字节一个扇区的硬盘，但截至2010年初，这种新硬盘使用固件对操作系统伪装成512字节一个扇区。[4]

使用英特尔架构的苹果机也使用GPT。

为了减少分区表损坏的风险，GPT在硬盘最后保存了一份分区表的副本。

传统MBR (LBA 0)

在GPT分区表的最开头，处于兼容性考虑仍然存储了一份传统的MBR，用来防止不支持GPT的硬盘管理工具错误识别并破坏硬盘中的数据，这个MBR也叫做叫做保护MBR。在支持从GPT启动的操作系统中，这里也用于存储第一阶段的启动代码。在这个MBR中，只有一个标识为0xEE的分区，以此来表示这块硬盘使用GPT分区表。不能识别GPT硬盘的操作系统通常会识别出一个未知类型的分区，并且拒绝对硬盘进行操作，除非用户特别要求删除着这分区。这就避免了意外删除分区的危险。另外，能够识别GPT分区表的操作系统会检查保护MBR中的分区表，如果分区类型不是0xEE或者MBR分区表中有多个项，也会拒绝对硬盘进行操作。

在使用MBR/GPT混合分区表的硬盘中，这部分存储了GPT分区表的一部分分区（通常是前四个分区），可以使不支持从GPT启动的操作系统从这个MBR启动，启动后只能操作MBR分区表中的分区。如Boot Camp就是使用这种方式启动Windows。

分区表头 (LBA 1)

分区表头定义了硬盘的可用空间以及组成分区表的项的大小和数量。在使用64位Windows Server 2003的机器上，最多可以创建128个分区，即分区表中保留了128个项，其中每个都是128字节。（EFI标准要求分区表最小要有16,384字节，即128个分区项的大小）

分区表头还记录了这块硬盘的GUID，记录了分区表头本身的位置和大小（位置总是在LBA 1）以及备份分区表头和分区表的位置和大小（在硬盘的最后）。它还储存着它本身和分区表的CRC32校验。固件、引导程序和操作系统在启动时可以根据这个校验值来判断分区表是否出错，如果出错了，可以使用软件从硬盘最后的备份GPT中恢复整个分区表，如果备份GPT也校验错误，硬盘将不可使用。所以GPT硬盘的分区表不可以直接使用16进制编辑器修改。

分区表头的格式 起始字节 长度 内容

0 8字节 签名（"EFI PART", 45 46 49 20 50 41 52 54）

8 4字节 修订（在1.0版中，值是 00 00 01 00）

12 4字节 分区表头的大小（单位是字节，通常是92字节，即 5C 00 00 00）

16 4字节 分区表头前3项（第0－15字节）的CRC32校验，如果值正在计算，则是 0

20 4字节 保留，必须是 0

24 8字节 当前LBA（这个分区表头的位置）

32 8字节 备份LBA（另一个分区表头的位置）

40 8字节 第一个可用于分区的LBA（主分区表的最后一个LBA + 1）

48 8字节 最后一个可用于分区的LBA（备份分区表的最后一个LBA − 1）

56 16字节 硬盘GUID（在类UNIX系统中也叫UUID）

72 8字节 分区表项的起始LBA（在主分区表中是2）

80 4字节 分区表项的数量

84 4字节 一个分区表项的大小（通常是128）

88 4字节 分区串行的CRC32校验

92 \* 保留，剩余的字节必须是0（对于512字节LBA的硬盘即是420个字节）

主分区表和备份分区表的头分别位于硬盘的第二个扇区（LBA 1）以及硬盘的最后一个扇区。备份分区表头中的信息是关于备份分区表的。

分区表项 (LBA 2–33)

GPT分区表使用简单而直接的方式表示分区。一个分区表项的前16字节是分区类型GUID。例如，EFI系统分区的GUID类型是{C12A7328-F81F-11D2-BA4B-00A0C93EC93B}。接下来的16字节是该分区唯一的GUID（这个GUID指的是该分区本身，而之前的GUID指的是该分区的类型）。再接下来是分区起始和末尾的64位LBA编号，以及分区的名字和属性。

GPT分区表项的格式 起始字节 长度 内容

0 16字节 分区类型GUID

16 16字节 分区GUID

32 8字节 起始LBA（小端序）

40 8字节 末尾LBA

48 8字节 属性标签（如：60表示“只读”）

56 72字节 分区名（可以包括36个UTF-16（小端序）字符）

操作系统支持

对于不标准的MBR/GPT混合硬盘，不同的系统中的实现有些不一致。[5]除非另加说明，操作系统在处理混合硬盘时优先读取GPT分区表

以下表格中的“不支持”应该理解成：不能识别GPT分区的硬盘，[6]系统只能识别保护分区。GPT硬盘的数据可以通过第三方管理工具进行操作。

分区类型GUID

相关操作系统 分区类型 GUID[1]

(None) 未使用 00000000-0000-0000-0000-000000000000

MBR分区表 024DEE41-33E7-11D3-9D69-0008C781F39F

EFI系统分区 C12A7328-F81F-11D2-BA4B-00A0C93EC93B

BIOS引导分区 21686148-6449-6E6F-744E-656564454649

Windows 微软保留分区 E3C9E316-0B5C-4DB8-817D-F92DF00215AE

基本数据分区[2] EBD0A0A2-B9E5-4433-87C0-68B6B72699C7

逻辑软盘管理工具元数据分区 5808C8AA-7E8F-42E0-85D2-E1E90434CFB3

逻辑软盘管理工具数据分区 AF9B60A0-1431-4F62-BC68-3311714A69AD

Windows恢复环境 DE94BBA4-06D1-4D40-A16A-BFD50179D6AC

IBM通用并行文件系统(GPFS)分区 37AFFC90-EF7D-4e96-91C3-2D7AE055B174

HP-UX 数据分区 75894C1E-3AEB-11D3-B7C1-7B03A0000000

服务分区 E2A1E728-32E3-11D6-A682-7B03A0000000

Linux 数据分区[2] EBD0A0A2-B9E5-4433-87C0-68B6B72699C7

RAID分区 A19D880F-05FC-4D3B-A006-743F0F84911E

交换分区 0657FD6D-A4AB-43C4-84E5-0933C84B4F4F

逻辑卷管理器(LVM)分区 E6D6D379-F507-44C2-A23C-238F2A3DF928

保留 8DA63339-0007-60C0-C436-083AC8230908

FreeBSD 启动分区 83BD6B9D-7F41-11DC-BE0B-001560B84F0F

数据分区 516E7CB4-6ECF-11D6-8FF8-00022D09712B

交换分区 516E7CB5-6ECF-11D6-8FF8-00022D09712B

UFS分区 516E7CB6-6ECF-11D6-8FF8-00022D09712B

en:Vinum volume manager分区 516E7CB8-6ECF-11D6-8FF8-00022D09712B

ZFS分区 516E7CBA-6ECF-11D6-8FF8-00022D09712B

Mac OS X HFS(HFS+)分区 48465300-0000-11AA-AA11-00306543ECAC

苹果公司UFS 55465300-0000-11AA-AA11-00306543ECAC

ZFS[3] 6A898CC3-1DD2-11B2-99A6-080020736631

苹果RAID分区 52414944-0000-11AA-AA11-00306543ECAC

苹果RAID分区，下线 52414944-5F4F-11AA-AA11-00306543ECAC

苹果启动分区 426F6F74-0000-11AA-AA11-00306543ECAC

Apple Label 4C616265-6C00-11AA-AA11-00306543ECAC

Apple TV 恢复分区 5265636F-7665-11AA-AA11-00306543ECAC

Solaris 启动分区 6A82CB45-1DD2-11B2-99A6-080020736631

根分区 6A85CF4D-1DD2-11B2-99A6-080020736631

交换分区 6A87C46F-1DD2-11B2-99A6-080020736631

备份分区 6A8B642B-1DD2-11B2-99A6-080020736631

/usr 分区[3] 6A898CC3-1DD2-11B2-99A6-080020736631

/var 分区 6A8EF2E9-1DD2-11B2-99A6-080020736631

/home 分区 6A90BA39-1DD2-11B2-99A6-080020736631

备用扇区 6A9283A5-1DD2-11B2-99A6-080020736631

保留分区 6A945A3B-1DD2-11B2-99A6-080020736631

6A9630D1-1DD2-11B2-99A6-080020736631

6A980767-1DD2-11B2-99A6-080020736631

6A96237F-1DD2-11B2-99A6-080020736631

6A8D2AC7-1DD2-11B2-99A6-080020736631

NetBSD[4] 交换分区 49F48D32-B10E-11DC-B99B-0019D1879648

FFS分区 49F48D5A-B10E-11DC-B99B-0019D1879648

LFS分区 49F48D82-B10E-11DC-B99B-0019D1879648

RAID分区 49F48DAA-B10E-11DC-B99B-0019D1879648

concatenated分区 2DB519C4-B10F-11DC-B99B-0019D1879648

加密分区 2DB519EC-B10F-11DC-B99B-0019D1879648

1.^ 本表中的GUID使用小端序表示。例如，EFI系统分区的GUID在这里写成C12A7328-F81F-11D2-BA4B-00A0C93EC93B但实际上它对应的16字节的串行是 28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B ——只有前3部分的字节序被交换了。

2.a b Linux和Windows的数据分区使用相同的GUID。

3.a b Solaris系统中/usr分区的GUID在Mac OS X上被用作普通的ZFS分区。

4.^ 具体定义见src/sys/sys/disklabel\_gpt.h。NetBSD的GUID在单独定义之前曾经使用过FreeBSD的GUID。

**用户界面**

用户介面或用户界面（英语：user interface，缩写UI)，是系统和用户之间进行交互和信息交换的媒介，它实现信息的内部形式与人类可以接受形式之间的转换。

用户接口是介于用户与硬件而设计彼此之间交互沟通相关软件，目的在使得用户能够方便有效率地去操作硬件以达成双向之交互，完成所希望借助硬件完成之工作，用户接口定义广泛，包含了人机交互与图形用户界面，凡参与人类与机械的信息交流的领域都存在着用户接口。

用户和系统之间一般用面向问题的受限自然语言进行交互。目前有系统开始利用多媒体技术开发新一代的用户界面。

组成要素

基本要素：必须有人类感官与'作用体所产生输出、输入与运作内容三方交互设计。

用户接口使得用户与系统之间双向的信息传递成为可能:

输入：允许用户操作运行一套系统。

输出：允许系统向用户传达操作效果。

人性要素 (Human Factor)：

人类感官知觉、心理情绪、认知、学习、记忆、反应、以及处理信息的模式、个别背景之差异等等每一项都和用户接口有密切的关系，直接或间接地影响用户接口的效能。

辅助与说明：

适当的说明文件，不管是联机的或是传统的文件。让用户明了系统究竟可以做与如何来达成工作。并让用户能够花最少的时间在接口的熟悉上，而将时间投注于完成其预定之工作项目。最简单的辅助说明就是图形或符号。

由此产生以下区别:

功能性界面

情感性界面

环境性界面

发展历史

古代

最古老的用户接口是在各式各样物体制作符号、图形，使人类与对象之间产生交互接口，比如说当人类看到招牌即知道这栋房屋的作用为何。

声音、旗帜、手势的运用，是让人与人或人与设备之间的用户接口，例如在战场或乐团演奏上，甚至是运用龟甲、钱币的卜卦，老师使用黑板作为与学生的用户接口，而算盘亦是由珠子所构成的早期人机界面。

早期电脑

1.批量接口 （风行于1945-1968）： 所有的输入数据预先设置于程序或命令行参数中。

2.命令行界面 （风行于1969-1983）：用户通过键盘输入指令，电脑接收到指令后，予以运行。

现代

电脑：图形用户界面

一般指介于用户与电脑之间沟通与交互之硬件以及软件，目的在使得电脑系统之用户能够方便有效率地去操作电脑以达成双向之交互，完成所希望借助电脑完成之工作，其涵盖之范围包括：早期由纸带输入设备到键盘、鼠标、数字版等等数据输入的设备，显示屏幕、声音等等输出设备，参考文件、联机说明、教学课程等等辅助使用之材料，人机交互的模式达到了只认识1与0的电脑与人类之间的用户接口。

在图形用户界面中，电脑画面上显示窗口、图标、按钮等图形表示不同目的的动作，用户通过鼠标等指针设备进行选择。最著名例子就是由苹果公司在麦金塔所创的图形用户界面。

在1980年代，电脑的用户接口革命是当时苹果电脑所发表的麦金塔(Macintosh)使用WIMP（视窗、图键、窗体，以及鼠标鼠标）将图形用户界面 (GUI)带进了大众市场，取代早期计算机使用的命令行界面。

1990年代初期微软所发表的Windows 3.0则巩固此一变革。

工业：人机界面

在工业界亦将传统按钮面板进入到亲切的触控人机界面（Human-Machine Interface，缩写HMI），在工业上的用户接口简单的区分为Input(输入)与Output(输出)两种，Input指的是由人来进行机械或设备的操作，如把手、开关、门、指令（命令）的下达或保养维护等，而Output指的是由机械或设备发出来的通知，如故障、警告、操作说明提示等，好的人机界面会帮助用户更简单、更正确、更迅速的操作机械，也能使机械发挥最大的效能并延长使用寿命，而目前市面上所指的人机界面则多半狭义的指在拥有软件人性化的操作接口的硬件(如触控屏幕。

其他

现在还有很多一些用户界面，由原本的按钮、纸本等传统古代接口进化至直接用手指、或者特殊的笔端触摸触摸屏上显示的按钮、图标进行各种操作，如自动取款机(ATM)，汽车导航、媒体播放器、游戏机、手机等等，一般操作简捷，直观。

未来

未来的用户接口：

更加真实交互的虚拟实境：举例来说，就是比个粗鲁动作就能当作Control-Alt-Delete键的意思。

脑机接口：

它是在人或动物脑（或者脑细胞的培养物）与外部设备间创建的直接连接通路，目前还在实验阶段，科学家在全身瘫痪病患实际脑中植入电脑芯片，已成功利用脑电波来控制电脑，画出简单的图案及电脑游戏，不过未来研究成功时，就可让人类就可利用脑波作为用户接口。

应用产品

用户借由触控式的人机界面与手机、PDA等设备沟通)

工业应用

触控屏幕：是可用以取代机械式的按钮面板，并借由可接收触头（无论是手指或胶笔尖等）等输入信号的感应式液晶显示设备达到了机械设备与人类之间的人机界面。

娱乐应用

游戏机：游戏机就像一台电脑，主要组成组件都十分类似，也有各种的软件和硬件可供安装使用。游戏机主要的硬件组件包括CPU、存储器、存储媒体、影音输出设备、信号输入设备等机械设备。其中，游戏机的信号输入设备如手把或摇杆，将电视、电脑借由简单的一些按钮达到与人类之间巧妙的交互，其游戏软件更借由一些图形与文字达到机械与人类之间娱乐的人机界面。

电脑应用

硬件方面：键盘、鼠标、显示器之间构成

软件方面：微软的Windows与苹果的MacOS。

网络应用：10 个改善 WEB UI 设计的技术

**命令行界面**

命令行界面（Command Line Interface，简写：CLI）是在图形用户界面得到普及之前使用最为广泛的用户界面，它通常不支持鼠标，用户通过键盘输入指令，计算机接收到指令后，予以执行。也有人称之为字符用户界面（CUI）。

通常认为，命令行界面（CLI）没有图形用户界面（GUI）那么方便用户操作。因为，命令行界面的软件通常需要用户记忆操作的命令，但是，由于其本身的特点，命令行界面要较图形用户界面节约计算机系统的资源。在熟记命令的前提下，使用命令行界面往往要较使用图形用户界面的操作速度要快。所以，在现在的图形用户界面的操作系统中，通常都保留着可选的命令行界面。

虽然现在许多电脑系统都提供了图形化的操作方式，但是却都没有因而停止提供文字模式的命令行操作方式，相反的，许多系统反而更加强这部份的功能，例如Windows就不只加强了操作命令的功能和数量，也一直在改善Shell Programming的方式。而之所以要加强、改善，自然是因为不够好；操作系统的图形化操作方式对单一客户端电脑的操作，已经相当方便，但如果是一群客户端电脑，或者是24小时运作的服务器电脑，图形化操作方式有时会力有未逮，所以需要不断增强命令行接口的脚本语言和宏语言来提供丰富的控制与自动化的系统管理能力，例如Linux系统的Bash或是Windows系统的Windows PowerShell。

常见的 CLI 程序

bash / sh / ksh / csh（Unix-like 系统）

COMMAND.COM（MS-DOS 系统）

cmd.exe / 命令提示字符（Windows NT 系统）

Windows PowerShell（支持 .NET Framework 技术的 Windows NT 系统）

**图形用户界面**

图形用户界面（Graphical User Interface，简称 GUI，又称图形用户接口）是指采用图形方式显示的计算机操作用户接口。与早期计算机使用的命令行界面相比，图形界面对于用户来说在视觉上更易于接受。然而这接口若要通过在显示器的特定位置，以“各种美观、而不单调的视觉消息”提示用户“状态的改变”，势必得比简单的文字消息体现，花上更多的计算能力，计算“要改变显示器哪些光点，变成哪些颜色”。

概要

在图形用户界面中，计算机画面上显示窗口、图标、按钮等图形，表示不同目的之动作，用户通过鼠标等指针设备进行选择。

组成

桌面

在启动时显示，也是界面中最底层，有时也指代包括窗口、文件浏览器在内的“桌面环境”。在桌面上由于可以重叠显示窗口，因此可以实现多任务化。一般的界面中，桌面上放有各种应用程序和数据的图标，用户可以依此开始工作。桌面与既存的文件夹构成里面相违背，所以要以特殊位置的文件夹的参照形式来定义内容。比如在微软公司的Windows XP系统中，各种用户的桌面内容实际保存在系统盘（默认为C盘）:\Documents and Settings\[用户名]\桌面 文件夹里。

墙纸，即桌面背景。可以设置为各种图片和各种附件，成为视觉美观的重要因素之一。

窗口

应用程序为使用数据而在图形用户界面中设置的基本单元。应用程序和数据在窗口内实现一体化。在窗口中，用户可以在窗口中操作应用程序，进行数据的管理、生成和编辑。通常在窗口四周设有菜单、图标，数据放在中央。

在窗口中，根据各种数据/应用程序的内容设有标题栏，一般放在窗口的最上方，并在其中设有最大化、最小化(隐藏窗口，并非消除数据)、最前面、缩进(仅显示标题栏)等动作按钮，可以简单地对窗口进行操作。

单一文件界面（Single Document Interface）

在窗口中，一个数据在一个窗口内完成的方式。在这种情况下，数据和显示窗口的数量是一样的。若要在其他应用程序的窗口使用数据，将相应生成新的窗口。因此窗口数量多，管理复杂。

多文件界面（Multiple Document Interface）

在一个窗口之内进行多个数据管理的方式。这种情况下，窗口的管理简单化，但是操作变为双重管理。

标签

多文件界面的数据管理方式中使用的一种界面，将数据的标题在窗口中并排，通过选择标签标题显示必要的数据，这样使得接入数据方式变得更为便捷。

菜单

将系统可以执行的命令以阶层的方式显示出来的一个界面。一般置于画面的最上方或者最下方，应用程序能使用的所有命令几乎全部都能放入。重要程度一般是从左到右，越往右重要度越低。命定的层次根据应用程序的不同而不同，一般重视文件的操作、编辑功能，因此放在最左边，然后往右有各种设置等操作，最右边往往设有帮助。一般使用鼠标的第一按钮进行操作。

即时菜单（又称菜单）

与应用程序准备好的层次菜单不同，在菜单栏以外的地方，通过鼠标的第二按钮调出的菜单称为“即时菜单”。根据调出位置的不同，菜单内容即时变化，列出所指示的对象目前可以进行的操作。

图标

显示在管理数据的应用程序中的数据，或者显示应用程序本身。

数据管理程序，即在文件夹中用户数据的管理、进行特定数据管理的程序的情况下，数据通过图标显示出来。通常情况下显示的是数据的内容或者与数据相关联的应用程序的图案。另外，点击数据的图标，一般可以之间完成启动相关应用程序以后再显示数据本身这两个步骤的工作。

应用程序的图标只能用于启动应用程序。

按钮

菜单中，利用程度高的命令用图形表示出来，配置在应用程序中，成为按钮。

应用程序中的按钮，通常可以代替菜单。一些使用程度高的命令，不必通过菜单一层层翻动才能调出，极大提高了工作效率。但是，各种用户使用的命令频率是不一样的，因此这种配置一般都是可以由用户自定义编辑。

其他

回收站

为了实现文件删除的“假安全”功能而设置了“回收站”（垃圾桶）功能。在文件删除的时候，暂时将其移动到系统特定的地方，一旦用户发现删除错误，还可以将其找回，从而实现防止错误删除的目的。在麦金塔系统中，垃圾桶不仅可以删除文件，还可以进行各种各样对象的删除功能，如将可移动硬盘从系统中移出，将光盘从光驱中取出等等。

应用程序启动器

从图形界面上启动应用程序有很多方式，有好几种操作系统都采用菜单形式的程序启动器。NEXTSTEP和Mac OS X中有一种称为dock的操作面板型的工具，可以存放各种文件和应用程序的信息，并通过鼠标点击调出。

图形用户界面的任务管理

在图形用户界面中，用户操作是以窗口为单位的。除了MDI和Mac OS以外，大多都是“窗口数量=任务数量”。因此在看整体界面的时候，怎样进行任务管理是很重要的。Windows等操作系统中，最常用的方式是在桌面上设置一个棒状的”任务栏“，放置各种窗口的图标和标题，确保系统的可操作性和可视性，方便对窗口进行管理。其他的方法包括，在桌面上的菜单中添加各个窗口管理菜单，在桌面上显示任务的图标，用虚拟桌面的方式表面增加桌面的数量等等。在Mac OS X系统中使用Dock进行任务管理，但是还有Exposé进行窗口一览显示模式的功能。

指针设备的操作

图形用户界面的基本操作是，用指针设备（一般是鼠标）进行指示操作，然后使用设备上的按钮（通常为两到三个）进行动作的激活。因此“位置”和“指示”都非常明了，从而实现可视操作。

指示的内容根据位置而不同。在数据管理应用程序中，第一按钮进行指针所在位置数据的选择，而两次连续按钮（所谓“双击”）可以调出预制的应用程序开始处理数据。 第二按钮通常用来显示即时菜单。第二按钮调出的菜单可以再用第一按钮进行选择打操作。第三按钮在X Window System中比较常用。

另外，最近四键、五键鼠标相继问世，各个按钮可以在操作系统中进行动作定义。

图形用户界面与键盘

和命令用户界面一样，键盘在图形用户界面仍是一个重要的设备。键盘不仅可以输入数据的内容，而且可以通过各种预先设置的“快捷键”等键盘组合进行命令操作达到和菜单操作一样的效果，并极大提高工作效率。

图形用户界面与各种设备

除了上述的设备以外，手写板等操作，特别是在图像数据操作中也扮演重要的角色。

触摸屏图形用户界面

现在还有很多一些用户界面，直接用手指、或者特殊的笔端触摸触摸屏上显示的按钮、图标进行各种操作，已经非常普及，如自动取款机ATM，汽车导航、媒体播放器、游戏机等等，一般操作简捷，直观。苹果公司的iPhone手机还有装有多手指操作系统。

采用图形用户界面的操作系统/应用程序

Smalltalk

Mac OS

NEXTSTEP

Mac OS X

Microsoft Windows

X Window System (类UnixOS、Linux)

BTRON

TownsOS

MSX-View

SX-Window

BeOS

Newton OS

Zaurus OS

Palm OS

等等。

**WIMP**

WIMP是图形界面电脑所采用的界面典范。在人机互动领域之中最普遍的电脑互动界面，WIMP堪称无人能出其右，举凡微软的Windows、苹果电脑的MacOS，甚至其它以X-Window为基础的操作系统，均采用WIMP此一界面典范。WIMP是由“视窗”（Window）、“图标”（Icon）、“选单”（Menu）以及“指标”（Pointer）所组成的缩写，其命名方式也指明了它所倚赖的四大互动元件。

组成元件

视窗

视窗是一个可见的范围，每一个视窗均被视为独立的执行终端。一个视窗通常能够包含文字与图像。一般而言，为了增进视窗的可用性，一个视窗通常会包含各种控制元件，例如卷轴（Scrollbar）、标题列（Title Bar），以及调整大小及缩放功能的元件。当两个或多个视窗同时出现在桌面（desktop）上时，使用者可依个人的需求选择使其交叠、并排或著是串连

图标

图标是用来表示或指示使用者执行程序的简单图像，举凡缩小的视窗、桌面上的垃圾桶或著是程序的执行捷径，都可用图标来表示。图标的设计可以选择写实地再现它所代表的功能（例如：垃圾桶），也可以高度抽象。

指标

一般来说，WIMP这种接口最依赖的人机互动模式之一，就是“点选模式”（pointing & selecting）。因此，指标的存在对WIMP而言当然也是不可或缺的。指标的图示通常不只一种，而不同的指标图示同时也就代表了不同的功能，例如指标旁边带有沙漏即代表系统正在忙碌。然而，无论是何种设计，每一种指标都有“热点”，也就是它负责指定的区块。

选单

选单不仅在视窗接口的系统中扮演重要的角色，在其他非视窗接口的系统中往往也是相当普遍的存在。一个选单通常包含了以文字、符号与图像组成的命令列表，使用者借由指标点选以执行相关的程序。一般而言，选单的类型大致上有以下四种：下拉式选单（Pull-down menus）、下降式选单（Fall-down menus）、钉附式选单（Pin-up menus）、弹出式选单（Pop-up menus）以及环形式选单（Pie menus）。

除了以上四种主要互动元件之外，一般而言WIMP也包含了以下三种元件：

按钮

按钮是一个独立的控件，可供使用者点选以启动特定的程序，或著是在两种系统状态之间做切换的动作。

工具列

许多系统都会有一条或一块区域，上头布满了图标或是按钮，以提供使用者启动他最常使用的执行程序。工具列（toolbar）的功能近似于选单，但因为工具列上往往只用图标，而非选单上常用的文字描述，因此得以在最小的空间内塞入最多的功能。

对话框

对话框（Dialog box）是系统为了要提醒使用者特定资讯而出现的视窗，除了显示警告讯息之外，对话框还可以担任“确认者”的角色，比方说存盘时的位址、档案的删除与否等互动状况。

标准

麦金塔

当苹果电脑取用了全录帕洛阿尔托研究中心（Xerox Palo Alto Research Center）的点子，并大量应用在畅销的麦金塔电脑（Macintosh） 上时，也一并针对WIMP这个互动接口典范确立了数项标准。

以使用者的一般日常生活为隐喻（metaphors），以方便他们学习使用。

看（see）与点（point）的使用哲学

所见即所得（WYSIWYG）

回馈与对话

允许使用者重做的空间

使用最少但最具品味的颜色来设计

开关元件（Toggled Items）

元件、图标等的分类必须有一致的原则

对话视窗的必要性

Windows

Windows大部分的设计理念来自于麦金塔的指引方针。事实上，早期视窗系统的某些功能还是苹果电脑所授权。以下是视窗系统与麦金塔系统明显的相异之处：

主要的程式工具列是可以移动的

所有的选单与对话框都可被操作

鼠标的右键可用来操作元件属性