FAT12 and FAT16 description.

HIW Document Server Index
Updated 09 Nov 96

Contents:

ATA NEWS

November 1996

by Hale Landis <landis@sugs.tware.com>

Well... It is about time I got around to updating the document! Much has happened during 1996 in the ATA/ATAPI world. Mostly the level of chaos has increased but a few good things did happen.

X3T13

One of the good things that happened was in Jan 1996 -- the X3T13 committee was formed. X3T13 is responsible for all of the ATA standards (ATA-1,...ATA/ATAPI-4, etc). Finally the ATA committee work is free of the X3 SCSI committtees (X3T10, etc) and can proceed at its own pace. The ATA/ATAPI Working Group of X3T13 meets once a month and during 1996 the meetings have been attended by many people and much good progress has been made towards producing an ATA/ATAPI-4 document (see below).

X3T13 is one of many Technical Committees in the Accredited Standards Committee (ASC) X3. This committee is authorized to create and maintain certain computer industry standards within the USA. Many of these standards are submitted to the International Standards Organization (ISO) to become worldwide standards.

X3T13 is currently responsible for all of the ATA standards. A different committee, X3T10, is responsible for the SCSI standards. X3T13 (and X3T10) replaced an older committee know as X3T9 which did the early ATA and SCSI standards.

THE ATA STANDARDS

The ATA standard (the real IDE/EIDE standard) comes in several flavors: ATA (also called ATA-1), ATA-2, ATA-3 and now ATA/ATAPI-4. Note that while an X3 committee is working on such a document, you can usually get the document via the Internet. However, once it becomes an official ANSI standard the ONLY way to get a copy is on paper and such copies generally cost $50 each. (This is really stupid and I think you should tell ANSI and X3 that. I'll tell you how to do that in a future version of this document).

ATA (ATA-1)

ATA (ATA-1) is ANSI document number X3.221-1994. ATA is the real standard for what is widely known as IDE.

ATA-2

ATA-2 is the real standard for what is widely known as EIDE. The ATA-2 document has been delayed about two years while many editorial changes where made. There have been no technical changes made to the document. ATA-2 is expected to be published as an ANSI standard in Dec 96 or Jan 97. ATA-2 is ANSI document number X3.279-1996. ATA-2 introduced higher speed data transfer modes, PIO Modes 3 and 4 plus Multword DMA Mode 1 and 2). These modes allow the ATA interface to run data transfers up to about 16MB/second.

ATA-3

ATA-3 introduces some new features of questionable value: SMART and Security. (more about these below). Note that ATA-3 does NOT introduce any new (faster) PIO or DMA data transfer modes (there is no such thing as PIO mode 5!). ATA-3 has completed its public review and should be approved in early 1997. When published lated in 1997 as an ANSI standard it will be know as X3.298-1996 (or perhaps X3.298-1997).

ATA/ATAPI-4

The charter of X3T13 is to watch over the previously published ATA standards (ATA-1, ATA-2 and ATA-3) and to produce the next ATA standard. The next standard will merge the ATA interface, used mostly by hard disk drives, with the relatively new ATAPI interface used by all the new CD-ROM drives and tape backup devices. The original ATA-4 document has been renamed to ATA/ATAPI-4. ATA/ATAPI-4 adds and changes many things. Here is a brief list:

- Many old ATA commands and features are now obsolete, such as the Format Track and Read/Write Long commands.

- There is a new data transfer protocol named Ultra DMA that adds data integity (via a CRC check) and much higher data transfer rates (up to 33MB/second).

- The ATAPI command and reset protocols are new.

- There is a command overlapping and command queuing protocol for ATAPI devices and most likely this will be extended to include ATA devices. See below.

- There are many new minor features for both ATA and ATAPI devices.

ATAPI

So just what is ATAPI anyway?

ATAPI is the real name of the new CD-ROM (EIDE CD-ROM) and tape (ATAPI tape or EIDE tape) interface. This interface was originally developed by a group of CD-ROM companies with lots of help from Western Digial. ATAPI did not start as an ANSI standard -- it was a specification published by the Small Form Factor (SFF) committee. SFF is an ad hoc disk drive industry committtee that usually concerns itself with things like connectors, the location of mounting holes and other physical configuration stuff. The original SFF document for ATAPI was called SFF-8020.

ATAPI introduced a new command execution protocol for use on the ATA interface so that these new CD-ROM and tape drives could, in theory, be on the same ATA cable with an ATA hard disk drive. Basically, the ATAPI Packet command, command code A0H, is used to send what looks like a SCSI CDB accross the ATA interface. The actual data transfer (from/to the device's media) is done using the ATA PIO or DMA protocols.

If you want to know what "SCSI like" commands are accepted by ATAPI devices then you should probably read the appropriate SCSI-3 document(s) for back ground information. Then get the appropriate SFF document for the ATAPI device type, for example, SFF-8020 describes the ATAPI CD-ROM "SCSI like" command set. There are many of these ATAPI "command set" documents floating around the industry today and even keeping a list of them is difficult. Some others I know of are: QIC-157 (ATAPI tape), SFF-8070 (ATAPI Removable Rewritable Media), SFF-8080 (ATAPI CD-R/E) and SFF-8090 (Commands for DVD). Locating some of these documents can be difficult.

ATA-3 SMART FEATURE

Every so often the ATA hard disk drive marketing folks think they need a new "feature" to sell. Self Monitoring And Reporting Technology is just that -- a bunch of marketing junk. This feature allows a drive to monitor itself and report to the host system when it thinks it will fail. (Don't laugh!) This ignores the fact that the vast majority of all drive failures are sudden and without warning, usually on Monday morning when the system is turned on after being off all weekend.

ATA-3 SECURITY FEATURE

The Security feature is another marketing idea. Without understanding the true meaning of data security and what it really takes to secure data, this feature was added in ATA-3. Apparently some ATA marketing people think this will prevent "bad guys" from stealing notebook computers in airports! The feature allows a drive to have two 32-byte "passwords" and allows the drive to be "locked" until the one of the passwords is presented to the drive. When locked the drive does not allow reads or writes to the media. One of the many problems with this feature is that a drive with no passwords active can be given a "random" password that no one knows. This can make the drive useless.

ATA/ATAPI-4 COMMAND OVERLAP/QUEUING

ATAPI devices are mostly very slow devices. One of the things that the early computer designers learned in the 1950's was that you don't put slow devices on the same interface with fast devices. But this lesson was lost and the ATAPI folks are trying to convince you that it is no problem to mix ATA and ATAPI devices on the same interface. But, of course there is. And now to "solve" this performance problem, the ATAPI people have invented "command overlap" so that a command to an ATA hard disk can be executed while a command to a slow ATAPI device is in progress. Now a few people think this should be extended to include "command queuing". So far the ATA/ATAPI-4 document describes only a very limited method to do these things and only on an ATAPI device. But there are now some ATA folks saying that ATA devices should be able to do the same things.

The bottom line: If you really want poor performance and hangs and other strange things happening to your computer, then you

definitely want these features! But like SMART and Security, we will probably get this junk in both ATA and ATAPI soon.

To date I know of no devices that actually implement this and I don't know of any host software that tries to use it.

THE FUTURE (OR WHAT IS 1394?)

What do you get when a bunch of software people at Microsoft try to design hardware? You get a document named "PC 97". PC 97 is an attempt by Microsoft to tell the system builders and hardware designers how to design hardware so that it will be supported by Microsoft's operating systems starting in 1997. If this all sounds a little strange to you, just wait until you real the PC 97 documents!

PC 97 is attempting to replace all existing motherboard sockets and device interfaces with two new serial bus interfaces. One of these is USB, a low speed serial interface for keyboards, mice, modems, etc. The other is a questionable higher speed interface know as IEEE P1394 or just as 1394. There are lots of versions of 1394: 1394-95, 1394.2, 1394A, and FireWire (and there are more). One of the problems with 1394 is that it is not currently fast enough to really be used with a hard disk.

In PC 97, higher speed devices, such as hard disks, tapes and printers, would be attached to 1394 serial bus. So far there doesn't seem to be a lot of support for 1394 interface disk drives. One of the reason is that such drives for cost much more than existing ATA and SCSI drives (any new disk drive interface will be expensive for the first several years). Another reason is concern that 1394 is not really fast enough to support high speed devices.

Because there are not likely to be any disk or tape devices in the near future that support the 1394 interface directly, there is an effort underway to produce 1394 to ATA/ATAPI interface conversion boards. These small boards are currently know as "tailgate" boards. Such a board has a 1394 interface on one side and an ATA/ATAPI interface on the other side. The board also has a microprocessor and some amount of memory (both ROM and DRAM). Estimates for the cost of this board are $20 to $50. Compare this with the $10 or less price of existing PCI/ATA host adapter hardware and also remember that there is only ONE thing that drives the PC industry: COST. This whole thing may just be too expensive.

/end ATANEWS/

ATA/ATA-1/ATA-2/IDE/EIDE/etc FAQ

Part 1 of ? -- The Basics

Version 0b -- 7 Feb 95

by Hale Landis -- landis@sugs.tware.com

Note: Major changes from the previous version are marked by a "!" at the left margin on the first line of the changed paragraph.

First the "legal" stuff...

1) This FAQ is not intended to replace any other FAQ on this subject but is an attempt to provide historical and technical information about the ATA interface.

2) This FAQ is not an endorsement of any vendor's product(s).

3) This FAQ is not a recommendation to purchase any vendor's product(s).

4) Every effort is made to insure that all of the information presented here is not copyrighted, not proprietary and unrestricted.

4) When opinions are stated they are clearly identified, including the person's name and email address. Such opinions are offered as long as they contribute to the understanding of the subject being discussed. No "flames" allowed.

This is the first version of this FAQ. It will take some time to get all the significant information into it so it will be rapidly growing and changing during the next several weeks or months. I don't even know how many parts there will be yet! Versions will be numbered with simple integer numbers (no 1.1, 1.2, etc) starting at 0.

If you have a question that is not answered here or if you have unrestricted material that you would like to contribute, please email it to landis@sugs.tware.com. DO NOT send material that is copyrighted, proprietary or otherwise restricted in any way -- I can't use such material in this document.

Table of Contents

Glossary

Read and understand these terms. You will be lost and confused if you don't! Many of these are describe in much greater detail in other parts of this FAQ.

ATA or AT Attachment

ATA is the proper and correct name for what most people call IDE. In this document, ATA refers to all forms of ATA (ATA-1, ATA-2, etc, IDE, EIDE, etc). The ATA interface uses a single 40-conductor cable in most desktop systems.

ATA-1

ATA-1 is the common name of the original ATA (IDE) specification. ATA-1 is not an official standard yet. Final approval is pending.

ATA-2 or ATA Extensions

ATA-2 is the common name of the new ATA specification. ATA-2 is still in early draft form and has not been submitted for approval as an official standard.

ATA-3

! ATA-3 is the common name of a future version of the ATA specification. The ATA-3 working group has held several meeting but the only things adopted so far are a DMA version of the Identify command, a description of "device 1 only configurations" and a set of "security" commands.

! There is much discussion going on concerning merging ATA-3 with ATAPI. This will require some kind of "command overlap" capability. The details of this are consumming much of the meeting time.

ATAPI or ATA Packet Interface

ATAPI is a proposed new interface specification. Initially it will probably be used for CD-ROM and tape devices. It uses the ATA hardware interface at the physical level but uses a subset of the SCSI command set at the logical level. The ATAPI specification work is currently being done in the SFF committee.

! The ATAPI folks have delayed forwarding their CD-ROM specification from SFF to X3T10 so the X3T10 ATAPI working group has nothing to work on yet and have held no meetings.

! Block Mode

! Block mode is the name given to the use of the ATA Read Multiple and Write Multiple commands. These commands generate a single interrupt to the host system for each block of sectors transfered. The traditional Read Sectors and Write Sectors commands generate an interrupt to the host for each sector transfered.

CAM (Common Access Method) Committee

The Common Access Method committee, now disbanded, worked on two specifications: the CAM SCSI and CAM ATA specifications. Both specifications were forwarded to the X3T9 committee for further work years ago.

CHS or Cylinder/Head/Sector

CHS is the old and traditional way to address data sectors on a hard disk. This style of addressing relates a sector's address to the position of the read/write heads. In today's ATA devices, all sector addresses used by the host are logical and have nothing to do with the actual physical position of the sector on the media or the actual position of the read/write heads.

Command Block
Control Block

These are names given to the I/O register interface used by ATA devices. It refers to a set of I/O registers, or I/O ports and I/O port addresses used to program the device. These names replace the older term Task File.

DMA or Direct Memory Access

DMA is a method of data transfer between two devices that does not use the system's main processor as part of the data path. DMA requires lots of hardware: a DMA arbitration unit, a DMA

data transfer unit and host bus signals that enable the DMA controller to assume control of the host system's bus. When the DMA controller has control of the host system's bus, it moves data between the two devices by generating the appropriate bus read/write cycles. For the ATA READ DMA command this means generating an I/O read cycle and then a memory write cycle for each 16-bit word being transferred. For the ATA WRITE DMA command, a memory read cycle is followed by an I/O write cycle for each 16-bit word transferred.

EIDE or Enhanced IDE

EIDE is a marketing program started by Western Digital to promote certain ATA-2 features including ATAPI. WD has encouraged other product vendors to mark their products as "EIDE compatible" or "EIDE capable".

ESDI

See MFM.

Fast ATA

Fast ATA is a Seagate marketing program used to promote certain ATA-2 features in newer ATA devices. Seagate has encouraged other product vendors to mark their products as "Fast ATA compatible" or "Fast ATA capable".

Host or Host System

The computer system that the ATA device is attached to.

HBA or Host Bus Adapter or Host Adapter

The hardware that converts host bus signals to/from ATA interface signals. An ATA-1 host adapter is generally a very simple piece of hardware. An ATA-2 host adapter can be simple or complex.

IDE

IDE can mean any number of things: Imbedded Device (or Drive) Electronics (yes, you can spell embedded with an "i"), Intelligent Device (or Drive) Electronics, etc. The term IDE is the trademark of someone (Western Digital does not claim IDE as theirs but they do claim EIDE). Many hard disk vendors do not use IDE to describe their products to avoid any trademark conflicts.

LBA or Logical Block Addressing

LBA is a newer (for ATA it is newer) way to address data sectors on a hard disc. This style of addressing uses a 28-bit binary number to address a sector. LBA numbers start at zero. In today's ATA devices, all sector addresses used by the host are logical and have nothing to do with the actual physical location of the sector on the media.

Local Bus

Usually this refers to the processor's local bus in a high performance computer system. Usually the processor, the external processor instruction/data cache, the main memory controller and the bridge controller for the next low speed system bus, for example, a PCI bus, are located on the local bus. Lower speed local buses may have connectors that allow the attachment of other devices. For example, the VL-Bus is a local bus that can allow attachment of video, SCSI or ATA controllers. It is very difficult to attach other devices to high speed (say faster than 100MHz) local buses due to electrical restrictions that come into play at those higher speeds.

Master

ATA device 0. Device 0, the master, is the "master" of nothing. See Slave.

Megabyte or MB

Megabyte or MB is 1,000,000 bytes or 10^6 bytes. IT IS NOT 1,048,576 bytes or 2^20 bytes, repeat NOT!

MFM

In this document MFM refers to any of the older hard disk controller interfaces, MFM, RLL and ESDI. It is used to describe any hard disk controller that uses the Task File interface on the host side and the ST506/ST412 interface on the drive side.

OS

Operating System.

PC Card ATA
PCMCIA

We can thank the Personal Computer Memory Card International Association for the PC Card specification. The PCMCIA is a

nonprofit industry association. The PC Card ATA specification is another form of the ATA interface used by PCMCIA compatible ATA devices. This interface uses the PCMCIA 68-pin connector. Most 68-pin ATA devices are dual mode -- they can operate as either a PCMCIA PC Card ATA device or as a 68-pin ATA device.

PCI

We can thank Intel and the other members of the PCI committee for the PCI bus specification. PCI is intended to be the next high performance computer bus. PCI is not generally described as a processor local bus.

PIO or Programmed Input/Output

PIO is a method of data transfer between two devices that uses the system's main processor as part of the data path. On x86 systems, the REP INS and REP OUT instructions implement this data transfer method. INS reads and I/O port and writes the data into memory. OUTS reads data from memory and writes the data to an I/O port. Each time an INS or OUTS instruction is executed, the memory address is updated. The REP prefix causes the instructions to be repeated until a counter reaches zero.

RLL

See MFM.

Slave

ATA device 1. Device 1, the slave, is a "slave" to nothing. See Master.

Task File

This is the name given to the I/O register interface used by MFM controllers. It refers to a set of I/O registers, or I/O ports and I/O port addresses used to program the controller. In ATA, this name has been replaced by the terms Command Block and Control Block.

SCSI

See the SCSI FAQ.

SFF or Small Form Factor

The SFF committee is an ad hoc committee formed by most of the

major storage device and system vendors to set standards for the physical layout of hard disk and other devices. SFF has published many specifications that describe the physical mounting and connector specifications for hard disk devices, including ATA devices. During a brief period of time when the X3T9 committee was not doing much work on the ATA-1 interface, the SFF committee published several specifications that were not really part of the original SFF charter. Most, if not all, of these nonphysical specifications have now been incorporated into the latest X3T9 or X3T10 ATA specifications. ATAPI is currently an SFF specification.

ST506 and ST412

This is the common name for the hard disk controller to hard disk drive interface used by MFM, RLL and ESDI controllers and disk drives. ST stands for Seagate Technology. The ST506 and ST412 were the Seagate products that set the de facto standards for this interface many years ago. This interface is composed of two cables: a 34-conductor and a 20-conductor cable.

VESA and VL-Bus

We can thank the Video Electronics Standards Association for the VESA Local Bus or VL-Bus specification. The VL-Bus is one example of a local bus. VESA is a nonprofit industry association like the PCMCIA.

WG or Working Group

The actual work on various specifications and standards documents within the X3T9, X3T10 and SFF committees happens in working group meetings. Most WG meetings are held monthly.

X3T9 and X3T10

These are the names of the official standards committees that have worked on the ATA-1 and ATA-2 specifications. X3T9 was responsible for the SCSI and ATA-1 specifications and standards. X3T10 has replaced X3T9 and is now responsible for the current SCSI and ATA specifications and standards work.

528MB

This term is used in this document to describe the capacity boundary that exists in most x86 system software. This boundary limits the size of an ATA disk drive to 528MB. For cylinder/head/sector style addressing of disk data sectors, this number is computed as follows:

a) the number of cylinders are limited to 1024, numbered 0-1023.

b) the number of heads (per cylinder) are limited to 16, numbered 0-15,

c) the number of sectors (per track) are limited to 63, numbered 1-63.

d) a sector is 512 bytes,

e) 528MB means the following values:

(1024 * 16 * 63) or 1,032,192 data sectors

or

(1024 * 16 * 63 * 512) or 528,482,304 bytes.

68-pin ATA

This refers to a variation of the ATA interface that uses the PCMCIA 68-pin physical interface but does not use the PCMCIA electrical or logical interface. Most 68-pin ATA devices are dual mode -- they can operate as either a PCMCIA PC Card ATA device or as a 68-pin ATA device. This interface was developed within the SFF committee and is now included in ATA-2.

Basic Questions

Where did ATA come from?

What we now call the ATA-1 interface was developed for Compaq many years ago by Imprimus (then part of CDC, now part of Seagate) and Western Digital. The first ATA-1 hard disk drives were made by Imprimus but it was Conner that made the interface so popular.

How is ATA different from MFM?

From the host software standpoint, ATA is very much like the Task File interface used by MFM controllers. A properly written host software driver should not notice any difference between the MFM Task File interface and the ATA Command Block interface while doing basic commands such as Read/Write Sectors.

At the hardware level, ATA uses a single cable between a host bus adapter and the ATA device, where the MFM controller interface uses two cables between the controller and the drive.

In the MFM environment, the controller is one piece of hardware and the drive another piece of hardware. Most likely these two pieces of hardware are from different vendors. The MFM controller is dependent on the design of both the host bus and on the drive.

In the ATA environment, the host adapter is the one piece of hardware that is dependent on the host system bus design. The ATA interface is (mostly) system independent. All of the hard disk controller and drive logic is contained in the ATA device hardware. This gives the hard disk designer complete control over both the controller and drive functions.

Why is ATA so popular?

Two basic things make ATA so popular today: cost and hard disk drive technology. An ATA-1 host adapter is cheap, usually much less than $25US and it uses only one cable. On the technology side, current hard disk features, such as, defect handling, error recovery, zone recording, cache management and power management require that the controller be fully integrated with the read/write channel, the servo system and spindle hardware of the disk drive.

What are the basics of the ATA interface?

The ATA interface is a very simple interface based on an ISA bus I/O device architecture. The interface consists of two sets of I/O registers, mostly 8-bit, for passing command and status information. The registers are like a set of mail boxes with a door on front and back connected such that both doors can not be open at the same time. The front door is open when the Busy bit in the Status register is zero and the host can read and write the registers. The back door is open when the Busy bit in the Status register is one and the ATA device can read or write the registers.

The physical interface contains just enough signals for a 16 bit data bus, five register address bits, and a few control signals like read register, write register and reset.

ATA devices look like traditional hard disk drives even though some are not really a hard disc with rotating platters. User data is recorded in 512 byte sectors. Each sector has a sector address. There are two ways to

express sector addresses: by cylinder/head/sector (CHS) or by logical block address (LBA). CHS is standard, LBA is optional.

What is EIDE or Fast ATA?

Both are marketing programs used to promote various ATA-2 features, mostly the faster data transfer rates defined by ATA-2.

WD defines EIDE as:

* Support for drives larger than 528MB.

* Support for two connectors to allow up to four drives.

* Support for CD-ROM and tape peripherals.

* Support for 11.1/16.6 Mbytes/second, I/O Channel Ready PIO data transfers.

* Support for 13.3/16.6 Mbytes/second, DMA data transfers.

???Seagate defines Fast ATA as:

* Support for PIO mode 3 (11.1 MB/sec) and DMA mode 1(13.3 MB/sec).

* Support for Multi-sector [Read/Write Multiple] transfers.

* Support for >528 MB.

* Support for Identify Drive Extensions & Set Transfer Mode Extensions.

* Backward compatibility with ATA-1.

What does all of this mean to us?

Support for the ATA-2 high speed PIO and DMA data transfer modes is both a hardware and software issue.

Support for more than one hard disc controller (or ATA host adapter) requires the BIOS and/or the operating system to support more than one Task File or Command/Control Block

register set on the host bus.

The 528MB problem is due to the original design of the x86 BIOS which limits cylinders to 1024 and sectors to 63. The ATA interface allows up to 65,535 cylinders, 16 heads and 255 sectors -- that's about 136GB (137GB if is LBA is used). Support for devices over 528MB requires the BIOS and/or operating system to support some form of CHS translation. Note that LBA alone does not solve this problem (in fact, LBA may make things more complex).

Support for CD-ROM and tape will probably be done via the ATAPI interface which uses a different command structure than ATA. That makes ATAPI another host software issue.

What does an ATA-1 host adapter do?

An ATA-1 host adapter is a simple piece of logic whose main purpose is to reduce the system bus address lines from 12 (or more) down to 5. It may also buffer some signals giving some degree of electrical isolation between the host bus (usually an ISA or EISA bus) and the ATA bus. In ATA-1, the ATA interface is controlled directly by the host bus so that all timings are controlled by the host bus timing.

What does an ATA-2 host adapter do?

This answer is complex because it depends on how smart your ATA-2 host adapter is. First, an ATA-2 host adapter supports the ATA-2 higher speed data transfer rates. That requires that the host adapter is attached to something other than an ISA or EISA bus. Second, an ATA-2 host adapter may perform 32-bit wide transfers on the host bus. This requires FIFO logic and data buffers in the host adapter. Third, an ATA-2 host adapter may use a different data transfer protocol on the host side than is used on the ATA device side.

! ### Can I put an ATA-2 device on an ATA-1 host adapter?
! ### Can I put an ATA-1 device on an ATA-2 host adapter?

The answer to both questions is yes, as long as the electrical timing specifications of the device are not violated. In general it is impossible for an ATA-1 host adapter to violate the specifications of an ATA-2 device. It is possible for an ATA-2 host adapter to violate the timing specifications of an ATA-1 device but this is not common. Hoever, host adapter hardware design errors or software driver bugs can cause such a problem. The result will be corrupted data read or written to the ATA-1 device.

! ### I have an ATA-2 host adapter with an ATA-2 device. I want to
! ### add an ATA-1 device to this host adapter. Can I run the ATA-2
! ### device in its ATA-2 data transfer modes?

Sorry, *NO* you can *NOT* run the new drive in its faster data transfer modes. Be very careful, most of the ATA-2 host adapter vendors don't have anything in their setup documentation or software to prevent this sort of thing.

When you run the new drive at a data transfer speed that is faster than the older drive can support, you are violating the electrical interface setup and hold times on the older drive. There is no telling what the older drive will do about this, but you are asking for data corruption and other nasty problems on your older drive.

How many disk controllers and/or ATA host adapters and/or
SCSI host adapters can I put in my system?

From a hardware standpoint -- as many as you want as long as there are no I/O port address, memory address or interrupt request signal conflicts. From a software standpoint it is a whole different story.

First the simple x86 system hard disk controller configurations...

a) 1 ATA with 1 or 2 drives at I/O port addresses 1Fxh/3Fxh using interrupt request 14 (IRQ14).

b) 1 ATA with 1 drive at I/O port addresses 1Fxh/3Fxh using interrupt request 14 (IRQ14) plus a SCSI with 1 drive.

c) 1 SCSI with 1 or 2 drives.

Other configurations are possible but are most likely not supported in the system or SCSI host adapter BIOS. And if its not supported at the BIOS level, it is unlikely to be supported by an operating system, especially DOS. The primary reason the above configurations are so restrictive is that the original IBM x86 BIOS supported only one MFM controller with a maximum of 2 drives. This restriction was then coded into much x86 software including many early version of DOS. The configurations above work because they don't break this old rule.

Just remember this -- most systems will always boot from the first drive on the first controller. In a) that is ATA drive 0, in b) that is ATA drive 0, in c) that is SCSI drive 0.

And now the more complex configurations...

Once you go beyond the three configurations above all bets are off. Most likely you will need operating system device drivers in order to access any drives beyond the first two. And now your real problems start especially if you like to run more than one operating system!

If you do run more than one OS, then you need equivalent drivers for each system if you would like to access all the drives. Plus it would be nice if all the drivers configured the drives in the same manner and supported all the possible partitioning schemes and partition sizes. It would be especially nice if a driver would not destroy the data in a partition just because it did not understand the file system format in that partition.

One of the things EIDE promotes is BIOS support for up to four ATA devices -- 2 ATA host adapters each with 1 or 2 drives. The first would be at I/O port addresses 1Fxh/3Fxh using interrupt request 14 (IRQ14) and the second at I/O port addresses 17xh/37xh using interrupt request 15 (IRQ15). Acceptance of this configuration has not been spreading like wild fire through the BIOS world.

Lets look at a two complex configurations...

a) 1 ATA with 2 drives and 1 SCSI with 1 or more drives.

Nice configuration. The ATA drives would be supported by the system BIOS and the SCSI drives may be, could be, should be, supported by the SCSI host adapter BIOS but probably not. So in order to use the 2 SCSI drives you probably have to disable the BIOS on the SCSI card and then load a device driver in CONFIG.SYS. And because the SCSI BIOS is disabled, you then need a SCSI driver for that other OS you run.

b) 2 ATA with 1 or 2 drives on each.

Also a nice configuration. But because the system BIOS probably only supports the first controller address, you'll need a DOS device driver loaded in CONFIG.SYS in order to access the drives on the second controller. You'll need that driver even if there is only one drive on the first controller. You also need a similar driver to support the second controller in your other OS.

Note: I understand that OS/2 does support both MFM/ATA controller addresses and does allow up to four drives -- I

have not confirmed this for myself.

! ### Are disk drives the only ATA devices?

No. Over the years there have been ATA tape drives, ATA CD-ROMS and other strange devices. Most of these are expected to be added to an existing ATA host adapter as the second device (device 1) with an existing ATA disk drive (device 0). In general these require software drivers to operate with your OS.

Now, we have ATAPI CD-ROM and tape devices that can be placed on an ATA host adapter. And soon we should see system motherboard BIOS support for booting from an ATAPI CD-ROM device. The general idea is that an ATAPI device can coexist with an ATA device on the same cable.

! ### What can be done to improve ATA device performance?

A difficult question. But the first step is usually to reduce the number of interrupts that the host sees during a read or write command. ATA disk drives have three types of read/write commands:

* Read Sectors / Write Sectors -- These commands are the old traditional data transfer commands. These commands generate one interrupt to the host for each sector transfered. These are PIO data in and PIO data out commands which use the host processor to transfer the data.

* Read Multiple / Write Multiple -- These commands where defined in ATA-1 but were not used very much until recently. These commands generate one interrupt to the host for each block of sectors transfered. The number of sector per block is generally 4, 8 or 16. However, when 1 sector per block is used, these commands are the same as the Read/Write Sectors commands. These are PIO data in and PIO data out commands which use the host processor to transfer the data.

* Read DMA / Write DMA -- These commands where defined in ATA-1 but were not used very much until recently. The main reason for not using them was that x86 system DMA transfer rates are much slower than PIO. However, these command do generate a single interrupt at the completion of the command.

Now see the next question...

! ### What else can be done to improve ATA device performance?
! ### -or-
! ### What is PIO mode "x" ?

An even more difficult question. The second step is usually to increase the rate at which the host transfers data.

(Ahh... I can see the funny look on your face from here. You are saying to yourself: "the rate at which the host transfers data? doesn't this guy have things backwards?" Read on...)

The rate at which data is transferred to or from an ATA device is determined by only one thing: the PIO or DMA cycle time the host uses. No, the drive does not have much to do with this! The only requirement is that the host not exceed the minimum PIO or DMA cycle times that the device supports. For example, during a PIO read command when the device signals an interrupt to the host this means that the device is waiting for the host to read the next sector or block of sectors from the drive. The host must execute a REP INSW instruction to do transfer the data. The rate at which the host executes this instruction determines the PIO cycle time. Technically, for a read command, the cycle time is the time from the host assertion of the I/O Read signal to the next time the host asserts the I/O Read signal.

Be careful when looking at the table below -- the data rate is the data transfer rate achieved while transfering the sector or block or sectors. It is an "instantanous" data rate. The overall data transfer rate for a command includes many time consuming events such as the amount of time the host requires to process an interrupt. Note that on many fast ATA drives today, the time it takes the host to process an interrupt is frequently greater than the time required to transfer the sector of block of sectors for that interrupt! It is not uncommon for the host overhead to reduce the data rate to 1/2 or 1/3 of the instantanous rate shown here.

The ATA PIO modes are defined as follows:

PIO mode	min cycle time	data rate	comment
0	???ns	?MB	the rate at which a system running at 4.77MHZ could execute the REP INSW.
1	???ns	?MB	the rate at which a system running at 6MHz could execute the REP INSW.
2	240ns	8MB	the rate at which a system running at 8MHz could

execute the REP INSW.

3 180ns 13MB requires an ATA-2 host adapter.

4 120ns 16MB requires an ATA-2 host adapter.

The complete description of the PIO (and DMA modes is much more complex and will be cover in more detail later in this FAQ.

Do I need BIOS or OS drivers to support more than 528MB?

Warning: Read the previous question before reading this one.

Maybe, probably, yes. The answer to this *very* complex and will be discussed in detail in Part 2. Here is the brief answer...

A traditional x86 system BIOS supports only CHS mode addressing with cylinders limited to 1024, heads limited to 16 and sectors limited to 63. This allows addressing of drives up to 528MB. These limitations come from the INT 13 read/write calls that combine a 10-bit cylinder number with a 6-bit sector number into a 16-bit register.

Note that this is entirely a software problem: the ATA interface supports up to 65,535 cylinders, 16 heads and 255 sectors.

While the head number usually requires only 4-bits, up to 6 or 8 bits are available in the INT 13 interface. This fact has allowed the SCSI folks to support big drives by increasing the number of heads above 16. The SCSI host adapter BIOS converts this "fake" CHS address to a different CHS or an LBA when it issues a read/write command to the drive. The following table shows some approximate drives sizes and the "fake" CHS parameters that you may see from a SCSI BIOS:

cyl	head	sector	size
1024	16	63	512MB
1024	32	63	1GB
512	64	63	1GB
1024	64	63	2GB
1024	256	63	8GB

The last entry represents the largest possible drive that a traditional INT 13 BIOS can support.

The system BIOS folks *must* start supporting drives over 528MB in their BIOS by implementing some type of CHS translation. To date, few systems have such BIOS. And here is the bad part: Microsoft says that the BIOS *must* support it in order to use it in their OS. The algorithm is simple (but warning: this is not the complete algorithm!):

INT 13 input	action	ATA interface
cyl number	"multiply" by n	modified cyl number
head number	"divide" by n	modified head number
sector number	nothing	sector number

The value of n must be selected such that the modified head number is less than 16.

LBA addressing at the hard disk drive level or at the BIOS or driver level is another solution. This solution will probably not be popular for several more years. It requires that the BIOS people implement a new INT 13 interface, called the Microsoft/IBM Extensions and that the OS people start using this new BIOS interface. Few system BIOS support this alternative interface today. Without this new interface, LBA support at the hard disk drive level is not required.

So most of us have older systems without much possibility of getting a BIOS upgrade, so what do we do? Well we must obtain one of the many driver products that are on the market that live in one of the disk boot sectors and "take over" the system BIOS INT 13 with an INT 13 that supports the translation. The biggest problem with this is that the replacement INT 13 BIOS must live someplace in memory. For DOS based systems, it can usually live at the top of the 640K of memory and DOS is made to think that that part of memory, usually around 8K bytes, does not exist. But the protected mode OS's don't like this and usually wipe out the driver when they load their kernel. So if you plan to run multiple OS's on your system, buyer beware!

Then there is the Windows problem: the standard FastDisk driver in Windows does *not* support such translation schemes and can not be enabled. So make sure the driver you purchase also comes with a Windows FastDisk replacement.

Buyer beware!

Do I need BIOS or OS drivers to support the ATA-2 data
transfer rates?

Warning: Read the previous two questions before reading this one.

Maybe, probably, yes. The answer to this *very* complex and will be discussed in detail in Part 2. Here is the brief answer...

If you have a new ATA drive that supports the advanced PIO or DMA data transfer rates (ATA-2 PIO Mode 3 or 4, or, ATA-2 DMA Mode 1 or 2) then you also must have a new ATA host adapter that attaches to the VL-Bus or PCI bus or some other high speed bus (probably a 32-bit bus) in your system. That host adapter has I/O registers of its own that are used to control its advanced features. Controlling those advanced features requires software -- either in the system INT 13 BIOS or in a INT 13 BIOS on the host adapter card or in a driver loaded via the boot record or later by your OS.

Depending on how that host adapter works you may also need a Windows FastDisk replacement in order to use the high speed data transfer modes in Windows.

Buyer beware!

I just purchased a new high speed host adapter for my VL-Bus
(or PCI bus) system and a new 540MB hard disk. How do I get
full use out of all this new hardware?

Did you read the previous three questions?

You need BIOS or driver software and a Windows FastDisk replacement. These *must* support both CHS translation (because your drive is over 528MB) and the host adapter hardware (to use the high speed data transfer rates).

Some drivers on the market today use LBA addressing on the ATA interface to get over 528MB. This may make your disk partition(s) unreadable by another OS.

Check the hardware and software specifications of the product before you buy it! Ask lots of questions -- you probably get lots of incorrect or misleading answers -- be prepared for that! If you plan to run something other than DOS and Windows, especially if you plan a "dual boot" or "boot manager" environment, be real careful.

Buyer beware!

OPINION: I know of only one product that supports all of this new hardware, supports over 528MB *and* supports most of the

current OS's that are shipping including several in shipping
in beta form. The product is from a small two person company
that is trying to sell the product on an OEM basis and not in
the retail market. - Hale Landis <landis@sugs.tware.com>

/end part 1/

How It Works -- CHS Translation

Plus BIOS Types, LBA and Other Good Stuff

Part 1 of 2

Version 4a

by Hale Landis (landis@sugs.tware.com)

THE "HOW IT WORKS" SERIES

This is one of several How It Works documents. The series currently includes the following:

* How It Works -- CHS Translation
* How It Works -- Master Boot Record
* How It Works -- DOS Floppy Boot Sector
* How It Works -- OS2 Boot Sector
* How It Works -- Partition Tables

Introduction (READ THIS!)

This is very technical. Please read carefully. There is lots of information here that can sound confusing the first time you read it.

Why is an understanding of how a BIOS works so important? The basic reason is that the information returned by INT 13H AH=08H is used by FDISK, it is used in the partition table entries within a partition record (like the Master Boot Record) that are created by FDISK, and it is used by the small boot program that FDISK places into the Master Boot Record. The information returned by INT 13H AH=08H is in cylinder/head/sector (CHS) format -- it is not in LBA format. The boot processing done by your computer's BIOS (INT 19H and INT 13H) is all CHS based.

Read this so that you are not confused by all the false information going around that says "LBA solves the >528MB problem".

Read this so that you understand the possible data integrity problem that a WD EIDE type BIOS creates. Any BIOS that has a "LBA mode" in the BIOS setup could be a WD EIDE BIOS. Be very careful and NEVER chage the "LBA mode" setting after you have partitioned and installed your software.

History

Changes between this version and the preceeding version are
marked by "!" at left margin of the first line of a changed
or new paragraph.

Version 4 -- BIOS Types 8 and 10 updated.

Version 3 -- New BIOS types found and added to this list. More
detailed information is listed for each BIOS type. A section
describing CHS translation was added.

Version 2 -- A rewrite of version 1 adding BIOS types not
included in version 1.

Version 1 -- First attempt to classify the BIOS types and
describe what each does or does not do.

Definitions
---

* 528MB - The maximun drive capacity that is supported by 1024
cylinders, 16 heads and 63 sectors (1024x16x63x512). This
is the limit for CHS addressing in the original IBM PC/XT
and IBM PC/AT INT 13H BIOS.

* 8GB - The maximum drive capacity that can be supported by 1024
cylinders, 256 heads and 63 sectors (1024x256x63x512). This
is the limit for the BIOS INT 13H AH=0xH calls.

* ATA - AT Attachment -- The real name of what is widely known
as IDE.

* CE Cylinder - Customer Engineering cylinder. This is the
last cylinder in P-CHS mode. IBM has always reserved this
cylinder for use of disk diagnostic programs. Many BIOS do
not account for it correctly. It is of questionable value
these days and probably should be considered obsolete.
However, since there is no industry wide agreement, beware.
There is no CE Cylinder reserved in the L-CHS address. Also
beware of diagnostic programs that don't realize they are
operating in L-CHS mode and think that the last L-CHS cylinder
is the CE Cylinder.

* CHS - Cylinder/Head/Sector. This is the traditional way to
address sectors on a disk. There are at least two types
of CHS addressing: the CHS that is used at the INT 13H
interface and the CHS that is used at the ATA device
interface. In the MFM/RLL/ESDI and early ATA days the CHS
used at the INT 13H interface was the same as the CHS used at

the device interface.

Today we have CHS translating BIOS types that can use one CHS at the INT 13H interface and a different CHS at the device interface. These two types of CHS will be called the logical CHS or L-CHS and the physical CHS or P-CHS in this document. L-CHS is the CHS used at the INT 13H interface and P-CHS is the CHS used at the device interface.

The L-CHS used at the INT 13 interface allows up to 256 heads, up to 1024 cylinders and up to 63 sectors. This allows support of up to 8GB drives. This scheme started with either ESDI or SCSI adapters many years ago.

The P-CHS used at the device interface allows up to 16 heads up to 65535 cylinders, and up to 63 sectors. This allows access to 2^28 sectors (136GB) on an ATA device. When a P-CHS is used at the INT 13H interface it is limited to 1024 cylinders, 16 heads and 63 sectors. This is where the old 528MB limit originated.

ATA devices may also support LBA at the device interface. LBA allows access to approximately 2^28 sectors (137GB) on an ATA device.

A SCSI host adapter can convert a L-CHS directly to an LBA used in the SCSI read/write commands. On a PC today, SCSI is also limited to 8GB when CHS addressing is used at the INT 13H interface.

* EDPT - Enhanced fixed Disk Parameter Table -- This table returns additional information for BIOS drive numbers 80H and 81H. The EDPT for BIOS drive 80H is pointed to by INT 41H. The EDPT for BIOS drive 81H is pointed to by INT 46H. The EDPT is a fixed disk parameter table with an AxH signature byte. This table format returns two sets of CHS information. One set is the L-CHS and is probably the same as returned by INT 13H AH=08H. The other set is the P-CHS used at the drive interface. This type of table allows drives with >1024 cylinders or drives >528MB to be supported. The translated CHS will have <=1024 cylinders and (probably) >16 heads. The CHS used at the drive interface will have >1024 cylinders and <=16 heads. It is unclear how the IBM defined CE cylinder is accounted for in such a table. Compaq probably gets the credit for the original definition of this type of table.

* FDPT - Fixed Disk Parameter Table - This table returns additional information for BIOS drive numbers 80H and 81H. The FDPT for BIOS drive 80H is pointed to by INT 41H. The FDPT for BIOS drive 81H is pointed to by INT 46H. A FDPT does

not have a AxH signature byte. This table format returns a single set of CHS information. The L-CHS information returned by this table is probably the same as the P-CHS and is also probably the same as the L-CHS returned by INT 13H AH=08H. However, not all BIOS properly account for the IBM defined CE cylinder and this can cause a one or two cylinder difference between the number of cylinders returned in the AH=08H data and the FDPT data. IBM gets the credit for the original definition of this type of table.

* LBA - Logical Block Address. Another way of addressing sectors that uses a simple numbering scheme starting with zero as the address of the first sector on a device. The ATA standard requires that cylinder 0, head 0, sector 1 address the same sector as addressed by LBA 0. LBA addressing can be used at the ATA interface if the ATA device supports it. LBA addressing is also used at the INT 13H interface by the AH=4xH read/write calls.

* L-CHS -- Logical CHS. The CHS used at the INT 13H interface by the AH=0xH calls. See CHS above.

* MBR - Master Boot Record (also known as a partition table) - The sector located at cylinder 0 head 0 sector 1 (or LBA 0). This sector is created by an "FDISK" utility program. The MBR may be the only partition table sector or the MBR can be the first of multiple partition table sectors that form a linked list. A partition table entry can describe the starting and ending sector addresses of a partition (also known as a logical volume or a logical drive) in both L-CHS and LBA form. Partition table entries use the L-CHS returned by INT 13H AH=08H. Older FDISK programs may not compute valid LBA values.

* OS - Operating System.

* P-CHS -- Physical CHS. The CHS used at the ATA device interface. This CHS is also used at the INT 13H interface by older BIOS's that do not support >1024 cylinders or >528MB. See CHS above.

Background and Assumptions

First, please note that this is written with the OS implementor in mind and that I am talking about the possible BIOS types as seen by an OS during its hardware configuration search.

It is very important that you not be confused by all the misinformation going around these days. All OS's that want to be

co-resident with another OS (and that is all of the PC based OS's that I know of) MUST use INT 13H to determine the capacity of a hard disk. And that capacity information MUST be determined in L-CHS mode. Why is this? Because: 1) FDISK and the partition tables are really L-CHS based, and 2) MS/PC DOS uses INT 13H AH=02H and AH=03H to read and write the disk and these BIOS calls are L-CHS based. The boot processing done by the BIOS is all L-CHS based. During the boot processing, all of the disk read accesses are done in L-CHS mode via INT 13H and this includes loading the first of the OS's kernel code or boot manager's code.

Second, because there can be multiple BIOS types in any one system, each drive may be under the control of a different type of BIOS. For example, drive 80H (the first hard drive) could be controlled by the original system BIOS, drive 81H (the second drive) could be controlled by a option ROM BIOS and drive 82H (the third drive) could be controlled by a software driver. Also, be aware that each drive could be a different type, for example, drive 80H could be an MFM drive, drive 81H could be an ATA drive, drive 82H could be a SCSI drive.

Third, not all OS's understand or use BIOS drive numbers greater than 81H. Even if there is INT 13H support for drives 82H or greater, the OS may not use that support.

Fourth, the BIOS INT 13H configuration calls are:

* AH=08H, Get Drive Parameters -- This call is restricted to drives up to 528MB without CHS translation and to drives up to 8GB with CHS translation. For older BIOS with no support for >1024 cylinders or >528MB, this call returns the same CHS as is used at the ATA interface (the P-CHS). For newer BIOS's that do support >1024 cylinders or >528MB, this call returns a translated CHS (the L-CHS). The CHS returned by this call is used by FDISK to build partition records.

* AH=41H, Get BIOS Extensions Support -- This call is used to determine if the IBM/Microsoft Extensions or if the Phoenix Enhanced INT 13H calls are supported for the BIOS drive number.

* AH=48H, Extended Get Drive Parameters -- This call is used to determine the CHS geometries, LBA information and other data about the BIOS drive number.

* the FDPT or EDPT -- While not actually a call, but instead a data area, the FDPT or EDPT can return additional information about a drive.

* other tables -- The IBM/Microsoft extensions provide a pointer

to a drive parameter table via INT 13H AH=48H. The Phoenix enhancement provides a pointer to a drive parameter table extension via INT 13H AH=48H. These tables are NOT the same as the FDPT or EDPT.

Note: The INT 13H AH=4xH calls duplicate the older AH=0xH calls but use a different parameter passing structure. This new structure allows support of drives with up to 2^64 sectors (really BIG drives). While at the INT 13H interface the AH=4xH calls are LBA based, these calls do NOT require that the drive support LBA addressing.

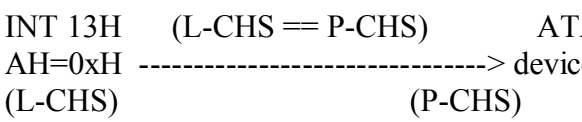
CHS Translation Algorithms

NOTE: Before you send me email about this, read this entire section. Thanks!

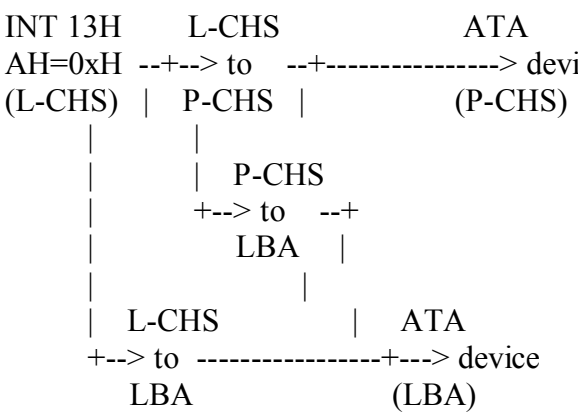
As you read this, don't forget that all of the boot processing done by the system BIOS via INT 19H and INT 13H use only the INT 13H AH=0xH calls and that all of this processing is done in CHS mode.

First, lets review all the different ways a BIOS can be called to perform read/write operations and the conversions that a BIOS must support.

! * An old BIOS (like BIOS type 1 below) does no CHS translation and does not use LBA. It only supports the AH=0xH calls:

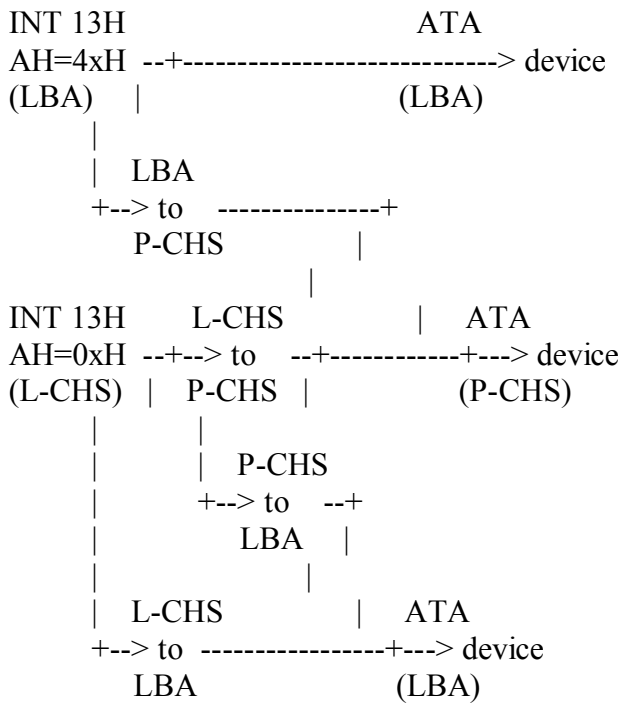


* A newer BIOS may support CHS translation and it may support LBA at the ATA interface:



* A really new BIOS may also support the AH=4xH in addtion to

the older AH\0xH calls. This BIOS must support all possible combinations of CHS and LBA at both the INT 13H and ATA interfaces:



You would think there is only one L-CHS to P-CHS translation algorithm, only one L-CHS to LBA translation algorithm and only one P-CHS to LBA translation algorithm. But this is not so. Why? Because there is no document that standardizes such an algorithm. You can not rely on all BIOS's and OS's to do these translations the same way.

The following explains what is widely accepted as the "correct" algorithms.

An ATA disk must implement both CHS and LBA addressing and must at any given time support only one P-CHS at the device interface. And, the drive must maintain a strick relationship between the sector addressing in CHS mode and LBA mode. Quoting the ATA-2 document:

```
LBA = ( (cylinder * heads_per_cylinder + heads )
      * sectors_per_track ) + sector - 1
```

where heads_per_cylinder and sectors_per_track are the current translation mode values.

This algorithm can also be used by a BIOS or an OS to convert a L-CHS to an LBA as we'll see below.

This algorithm can be reversed such that an LBA can be

converted to a CHS:

```
   cylinder = LBA / (heads_per_cylinder * sectors_per_track)
      temp = LBA % (heads_per_cylinder * sectors_per_track)
      head = temp / sectors_per_track
    sector = temp % sectors_per_track + 1
```

While most OS's compute disk addresses in an LBA scheme, an OS like DOS must convert that LBA to a CHS in order to call INT 13H.

Technically an INT 13H should follow this process when converting an L-CHS to a P-CHS:

1) convert the L-CHS to an LBA,
2) convert the LBA to a P-CHS,

If an LBA is required at the ATA interface, then this third step is needed:

3) convert the P-CHS to an LBA.

All of these conversions are done by normal arithmetic.

However, while this is the technically correct way to do things, certain short cuts can be taken. It is possible to convert an L-CHS directly to a P-CHS using bit a bit shifting algorithm. This combines steps 1 and 2. And, if the ATA device being used supports LBA, steps 2 and 3 are not needed. The LBA value produced in step 1 is the same as the LBA value produced in step 3.

AN EXAMPLE

Lets look at an example. Lets say that the L-CHS is 1000 cylinders 10 heads and 50 sectors, the P-CHS is 2000 cylinders, 5 heads and 50 sectors. Lets say we want to access the sector at L-CHS 2,4,3.

* step 1 converts the L-CHS to an LBA,

```
lba = 1202 = ( ( 2 * 10 + 4 ) * 50 ) + 3 - 1
```

* step 2 converts the LBA to the P-CHS,

```
cylinder =   4 = ( 1202 / ( 5 * 50 )
   temp = 202 = ( 1202 % ( 5 * 50 ) )
   head =   4 = ( 202 / 50 )
 sector =   3 = ( 202 % 50 ) + 1
```

* step 3 converts the P-CHS to an LBA,

lba = 1202 = ((4 * 5 + 4) * 50) + 3 - 1

Most BIOS (or OS) software is not going to do all of this to convert an address. Most will use some other algorithm. There are many such algorithms.

BIT SHIFTING INSTEAD

If the L-CHS is produced from the P-CHS by 1) dividing the P-CHS cylinders by N, and 2) multiplying the P-CHS heads by N, where N is 2, 4, 8, ..., then this bit shifting algorithm can be used and N becomes a bit shift value. This is the most common way to make the P-CHS geometry of a >528MB drive fit the INT 13H L-CHS rules. Plus this algorithm maintains the same sector ordering as the more complex algorithm above. Note the following:

Lcylinder = L-CHS cylinder being accessed
Lhead = L-CHS head being accessed
Lsector = L-CHS sector being accessed

Pcylinder = the P-CHS cylinder being accessed
Phead = the P-CHS head being accessed
Psector = P-CHS sector being accessed

NPH = is the number of heads in the P-CHS
N = 2, 4, 8, ..., the bit shift value

The algorithm, which can be implemented using bit shifting instead of multiply and divide operations is:

Pcylinder = (Lcylinder * N) + (Lhead / NPH);
Phead = (Lhead % NPH);
Psector = Lsector;

A BIT SHIFTING EXAMPLE

Lets apply this to our example above (L-CHS = 1000,10,50 and P-CHS = 2000, 5, 50) and access the same sector at at L-CHS 2,4,3.

Pcylinder = 4 = (2 * 2) + (4 / 5)
Phead = 4 = (4 % 5)
Psector = 3 = 3

As you can see, this produces the same P-CHS as the more complex method above.

SO WHAT IS THE PROBLEM?

The basic problem is that there is no requirement that a CHS translating BIOS followed these rules. There are many other algorithms that can be implemented to perform a similar function. Today, there are at least two popular implementions: the Phoenix implementation (described above) and the non-Phoenix implementations.

SO WHY IS THIS A PROBLEM IF IT IS HIDDEN INSIDE THE BIOS?

Because a protected mode OS that does not want to use INT 13H must implement the same CHS translation algorithm. If it doesn't, your data gets scrambled.

WHY USE CHS AT ALL?

In the perfect world of tomorrow, maybe only LBA will be used. But today we are faced with the following problems:

* Some drives >528MB don't implement LBA.

* Some drives are optimized for CHS and may have lower performance when given commands in LBA mode. Don't forget that LBA is something new for the ATA disk designers who have worked very hard for many years to optimize CHS address handling. And not all drive designs require the use of LBA internally.

* The L-CHS to LBA conversion is more complex and slower than the bit shifting L-CHS to P-CHS conversion.

* DOS, FDISK and the MBR are still CHS based -- they use the CHS returned by INT 13H AH=08H. Any OS that can be installed on the same disk with DOS must understand CHS addressing.

* The BIOS boot processing and loading of the first OS kernel code is done in CHS mode -- the CHS returned by INT 13H AH=08H is used.

* Microsoft has said that their OS's will not use any disk capacity that can not also be accessed by INT 13H AH=0xH.

These are difficult problems to overcome in today's industry environment. The result: chaos.

DANGER TO YOUR DATA!

See the description of BIOS Type 7 below to understand why a WD EIDE BIOS is so dangerous to your data.

/end of part 1 of 2 -- continued in part 2 of 2/

How It Works -- CHS Translation

Plus BIOS Types, LBA and Other Good Stuff

Part 2 of 2

Version 4a

by Hale Landis (landis@sugs.tware.com)

/continued from part 1 of 2/

The BIOS Types

I assume the following:

a) All BIOS INT 13H support has been installed by the time the OS starts its boot processing. I'm don't plan to cover what could happen to INT 13H once the OS starts loading its own device drivers.

b) Drives supported by INT 13H are numbered sequentially starting with drive number 80H (80H-FFH are hard drives, 00-7FH are floppy drives).

And remember, any time a P-CHS exists it may or may not account for the CE Cylinder properly.

I have identified the following types of BIOS INT 13H support as seen by an OS during its boot time hardware configuration determination:

BIOS Type 1

Origin: Original IBM PC/XT.

BIOS call support: INT 13H AH=0xH and FDPT for BIOS drives 80H and 81H. There is no CHS translation. INT 13H AH=08H returns the P-CHS. The FDPT should contain the same P-CHS.

Description: Supports up to 528MB from a table of drive descriptions in BIOS ROM. No support for >1024 cylinders or drives >528MB or LBA.

Support issues: For >1024 cylinders or >528MB support, either an option ROM with an INT 13H replacement (see BIOS types 4-7) -or- a software driver (see BIOS type 8) must be added to the system.

BIOS Type 2

Origin: Unknown, but first appeared on systems having BIOS drive type table entries defining >1024 cylinders. Rumored to have originated at the request of Novell or SCO.

BIOS call support: INT 13H AH=0xH and FDPT for BIOS drives 80H and 81H. INT 13H AH=08H should return a L-CHS with the cylinder value limited to 1024. Beware, many BIOS perform a logical AND on the cylinder value. A correct BIOS will limit the cylinder value as follows:

```
cylinder = cylinder > 1024 ? 1024 : cylinder;
```

An incorrect BIOS will limit the cylinder value as follows (this implementation turns a 540MB drive into a 12MB drive!):

```
cylinder = cylinder & 0x03ff;
```

The FDPT will return a P-CHS that has the full cylinder value.

Description: For BIOS drive numbers 80H and 81H, this BIOS type supports >1024 cylinders or >528MB without using a translated CHS in the FDPT. INT 13H AH=08H truncates cylinders to 1024 (beware of buggy implementations). The FDPT can show >1024 cylinders thereby allowing an OS to support drives >528MB. May convert the L-CHS or P-CHS directly to an LBA if the ATA device supports LBA.

Support issues: Actual support of >1024 cylinders is OS specific -- some OS's may be able to place OS specific partitions spanning or beyond cylinder 1024. Usually all OS boot code must be within first 1024 cylinders. The FDISK program of an OS that supports such partitions uses an OS specific partition table entry format to identify these paritions. There does not appear to be a standard (de facto or otherwise) for this unusual partition table entry. Apparently one method is to place -1 into the CHS fields and use the LBA fields to describe the location of the partition. This DOES NOT require the drive to support LBA addressing. Using an LBA in the partition table entry is just a trick to get around the CHS limits in the partition table entry. It is unclear if such a partition table entry will be ignored by an OS that does not understand what it is. For an OS that does not support such partitions, either an option ROM with an INT 13H replacement (see BIOS types 4-7) -or- a software driver (see BIOS type 8) must be added to the system.

Note: OS/2 can place HPFS partitions and Linux can place

Linux partitions beyond or spanning cylinder 1024. (Anyone know of other systems that can do the same?)

BIOS Type 3

Origin: Unknown, but first appeared on systems having BIOS drive type table entires defining >1024 cylinders. Rumored to have originated at the request of Novell or SCO.

BIOS call support: INT 13H AH=0xH and FDPT for BIOS drives 80H and 81H. INT 13H AH=08H can return an L-CHS with more than 1024 cylinders.

Description: This BIOS is like type 2 above but it allows up to 4096 cylinders (12 cylinder bits). It does this in the INT 13H AH=0xH calls by placing two most significant cylinder bits (bits 11 and 10) into the upper two bits of the head number (bits 7 and 6).

Support issues: Identification of such a BIOS is difficult. As long as the drive(s) supported by this type of BIOS have <1024 cylinders this BIOS looks like a type 2 BIOS because INT 13H AH=08H should return zero data in bits 7 and 6 of the head information. If INT 13H AH=08H returns non zero data in bits 7 and 6 of the head information, perhaps it can be assumed that this is a type 3 BIOS. For more normal support of >1024 cylinders or >528MB, either an option ROM with an INT 13H replacement (see BIOS types 4-7) -or- a software driver (see BIOS type 8) must be added to the system.

Note: Apparently this BIOS type is no longer produced by any BIOS vendor.

BIOS Type 4

Origin: Compaq. Probably first appeared in systems with ESDI drives having >1024 cylinders.

BIOS call support: INT 13H AH=0xH and EDPT for BIOS drives 80H and 81H. If the drive has <1024 cylinders, INT 13H AH=08H returns the P-CHS and a FDPT is built. If the drive has >1024 cylinders, INT 13H AH=08H returns an L-CHS and an EDPT is built.

Description: Looks like a type 2 BIOS when an FDPT is built. Uses CHS translation when an EDPT is used. May convert the L-CHS directly to an LBA if the ATA device supports LBA.

Support issues: This BIOS type may support up to four drives with a EDPT (or FDPT) for BIOS drive numbers 82H and 83H

located in memory following the EDPT (or FDPT) for drive 80H. Different CHS translation algorithms may be used by the BIOS and an OS.

BIOS Type 5

Origin: The IBM/Microsoft BIOS Extensions document. For many years this document was marked "confidential" so it did not get wide spread distribution.

BIOS call support: INT 13H AH=0xH, AH=4xH and EDPT for BIOS drives 80H and 81H. INT 13H AH=08H returns an L-CHS. INT 13H AH=41H and AH=48H should be used to get P-CHS configuration. The FDPT/EDPT should not be used. In some implementations the FDPT/EDPT may not exist.

Description: A BIOS that supports very large drives (>1024 cylinders, >528MB, actually >8GB), and supports the INT 13H AH=4xH read/write functions. The AH=4xH calls use LBA addressing and support drives with up to 2^64 sectors. These calls do NOT require that a drive support LBA at the drive interface. INT 13H AH=48H describes the L-CHS used at the INT 13 interface and the P-CHS or LBA used at the drive interface. This BIOS supports the INT 13 AH=0xH calls the same as a BIOS type 4.

Support issues: While the INT 13H AH=4xH calls are well defined, they are not implemented in many systems shipping today. Currently undefined is how such a BIOS should respond to INT 13H AH=08H calls for a drive that is >8GB. Different CHS translation algorithms may be used by the BIOS and an OS.

Note: Support of LBA at the drive interface may be automatic or may be under user control via a BIOS setup option. Use of LBA at the drive interface does not change the operation of the INT 13 interface.

BIOS Type 6

Origin: The Phoenix Enhanced Disk Drive Specification.

BIOS call support: INT 13H AH=0xH, AH=4xH and EDPT for BIOS drives 80H and 81H. INT 13H AH=08H returns an L-CHS. INT 13H AH=41H and AH=48H should be used to get P-CHS configuration. INT 13H AH=48H returns the address of the Phoenix defined "FDPT Extension" table.

Description: A BIOS that supports very large drives (>1024 cylinders, >528MB, actually >8GB), and supports the INT 13H AH=4xH read/write functions. The AH=4xH calls use LBA

addressing and support drives with up to 2^64 sectors. These calls do NOT require that a drive support LBA at the drive interface. INT 13H AH=48H describes the L-CHS used at the INT 13 interface and the P-CHS or LBA used at the drive interface. This BIOS supports the INT 13 AH=0xH calls the same as a BIOS type 4. The INT 13H AH=48H call returns additional information such as host adapter addresses, DMA support, LBA support, etc, in the Phoenix defined "FDPT Extension" table.

Phoenix says this this BIOS need not support the INT 13H AH=4xH read/write calls but this BIOS is really an extension/enhancement of the original IBM/MS BIOS so most implementations will probably support the full set of INT 13H AH=4xH calls.

Support issues: Currently undefined is how such a BIOS should respond to INT 13H AH=08H calls for a drive that is >8GB. Different CHS translation algorithms may be used by the BIOS and an OS.

Note: Support of LBA at the drive interface may be automatic or may be under user control via a BIOS setup option. Use of LBA at the drive interface does not change the operation of the INT 13 interface.

BIOS Type 7

Origin: Described in the Western Digital Enhanced IDE Implementation Guide.

BIOS call support: INT 13H AH=0xH and FDPT or EDPT for BIOS drives 80H and 81H. An EDPT with a L-CHS of 16 heads and 63 sectors is built when "LBA mode" is enabled. An FDPT is built when "LBA mode" is disabled.

Description: Supports >1024 cylinders or >528MB using a EDPT with a translated CHS *** BUT ONLY IF *** the user requests "LBA mode" in the BIOS setup *** AND *** the drive supports LBA. As long as "LBA mode" is enabled, CHS translation is enabled using a L-CHS with <=1024 cylinders, 16, 32, 64, ..., heads and 63 sectors. Disk read/write commands are issued in LBA mode at the ATA interface but other commands are issued in P-CHS mode. Because the L-CHS is determined by table lookup based on total drive capacity and not by a multiply/divide of the P-CHS cylinder and head values, it may not be possible to use the simple (and faster) bit shifting L-CHS to P-CHS algorithms.

When "LBA mode" is disabled, this BIOS looks like a BIOS type 2 with an FDPT. The L-CHS used is taken either from the BIOS

drive type table or from the device's Identify Device data.
This L-CHS can be very different from the L-CHS returned when
"LBA mode" is enabled.

This BIOS may support FDPT/EDPT for up to four drives in the
same manner as described in BIOS type 4.

The basic problem with this BIOS is that the CHS returned by
INT 13H AH=08H changes because of a change in the "LBA mode"
setting in the BIOS setup. This should not happen. This use
or non-use of LBA at the ATA interface should have no effect
on the CHS returned by INT 13H AH=08H. This is the only BIOS
type know to have this problem.

Support issues: If the user changes the "LBA mode" setting in
BIOS setup, INT 13H AH=08H and the FDPT/EDPT change
which may cause *** DATA CORRUPTION ***. The user should be
warned to not change the "LBA mode" setting in BIOS setup once
the drive has been partitioned and software installed.
Different CHS translation algorithms may be used by the BIOS
and an OS.

BIOS Type 8

Origin: Unknown. Perhaps Ontrack's Disk Manager was the
first of these software drivers. Another example of such a
driver is Micro House's EZ Drive.

BIOS call support: Unknown (anyone care to help out here?).
Mostly likely only INT 13H AH=0xH are support. Probably a
FDPT or EDPT exists for drives 80H and 81H.

! Description: A software driver that "hides" in the MBR such
that it is loaded into system memory before any OS boot
processing starts. These drivers can have up to three parts:
a part that hides in the MBR, a part that hides in the
remaining sectors of cylinder 0, head 0, and an OS device
driver. The part in the MBR loads the second part of the
driver from cylinder 0 head 0. The second part provides a
replacement for INT 13H that enables CHS translation for at
least the boot drive. Usually the boot drive is defined in
CMOS setup as a type 1 or 2 (5MB or 10MB drive). Once the
second part of the driver is loaded, this definition is
changed to describe the true capacity of the drive and INT 13H
is replaced by the driver's version of INT 13H that does CHS
translation. In some cases the third part, an OS specific
device driver, must be loaded to enable CHS translation for
devices other than the boot device.

! I don't know the details of how these drivers respond to INT

13H AH=08H or how they set up drive parameter tables (anyone care to help out here?). Some of these drivers convert the L-CHS to an LBA, then they add a small number to the LBA and finally they convert the LBA to a P-CHS. This in effect skips over some sectors at the front of the disk.

Support issues: Several identified -- Some OS installation programs will remove or overlay these drivers; some of these drivers do not perform CHS translation using the same algorithms used by the other BIOS types; special OS device drivers may be required in order to use these software drivers For example, under MS Windows the standard FastDisk driver (the 32-bit disk access driver) must be replaced by a driver that understands the Ontrack, Micro House, etc, version of INT 13H. Different CHS translation algorithms may be used by the driver and an OS.

! The hard disk vendors have been shipping these drivers with their drives over 528MB during the last year and they have been ignoring the statements of Microsoft and IBM that these drivers would not be supported in future OS's. Now it appears that both Microsoft and IBM are in a panic trying to figure out how to support some of these drivers in WinNT, Win95 and OS/2. It is unclear what the outcome of this will be at this time.

! NOTE: THIS IS NOT A PRODUCT ENDORSEMENT! An alternate solution for an older ISA system is one of the BIOS replacement cards. This cards have a BIOS option ROM. AMI makes such a card called the "Disk Extender". This card replaces the motherboard's INT 13H BIOS with a INT 13H BIOS that does some form of CHS translation. Another solution for older VL-Bus systems is an ATA-2 (EIDE) type host adapter card that provides a option ROM with an INT 13H replacement.

BIOS Type 9

Origin: SCSI host adapters.

BIOS call support: Probably INT 13H AH=0xH and FDPT for BIOS drives 80H and 81H, perhaps INT 13H AH=4xH.

Description: Most SCSI host adapters contain an option ROM that enables INT 13 support for the attached SCSI hard drives. It is possible to have more than one SCSI host adapter, each with its own option ROM. The CHS used at the INT 13H interface is converted to the LBA that is used in the SCSI commands. INT 13H AH=08H returns a CHS. This CHS will have <=1024 cylinders, <=256 heads and <=63 sectors. The FDPT probably will exist for SCSI drives with BIOS drive numbers of

80H and 81H and probably indicates the same CHS as that returned by INT 13H AH=08H. Even though the CHS used at the INT 13H interface looks like a translated CHS, there is no need to use a EDPT since there is no CHS-to-CHS translation used. Other BIOS calls (most likely host adapter specific) must be used to determine other information about the host adapter or the drives.

The INT 13H AH=4xH calls can be used to get beyond 8GB but since there is little support for these calls in today's OS's, there are probably few SCSI host adapters that support these newer INT 13H calls.

Support issues: Some SCSI host adapters will not install their option ROM if there are two INT 13H devices previously installed by another INT 13H BIOS (for example, two MFM/RLL/ESDI/ATA drives). Other SCSI adapters will install their option ROM and use BIOS drive numbers greater than 81H. Some older OS's don't understand or use BIOS drive numbers greater than 81H. SCSI adapters are currently faced with the >8GB drive problem.

BIOS Type 10

Origin: A european system vendor.

BIOS call support: INT 13H AH=0xH and FDPT for BIOS drives 80H and 81H.

Description: This BIOS supports drives >528MB but it does not support CHS translation. It supports only ATA drives with LBA capability. INT 13H AH=08H returns an L-CHS. The L-CHS is converted directly to an LBA. The BIOS sets the ATA drive to a P-CHS of 16 heads and 63 sectors using the Initialize Drive Parameters command but it does not use this P-CHS at the ATA interface.

! Support issues: OS/2 will probably work with this BIOS as long as the drive's power on default P-CHS mode uses 16 heads and 63 sectors. Because there is no EDPT, OS/2 uses the ATA Identify Device power on default P-CHS, described in Identify Device words 1, 3 and 6 as the current P-CHS for the drive. However, this may not represent the correct P-CHS. A newer drive will have the its current P-CHS information in Identify Device words 53-58 but for some reason OS/2 does not use this information.

/end of part 2 of 2/

How It Works -- DOS Floppy Disk Boot Sector

Version 1a

by Hale Landis (landis@sugs.tware.com)

THE "HOW IT WORKS" SERIES

This is one of several How It Works documents. The series currently includes the following:

* How It Works -- CHS Translation
* How It Works -- Master Boot Record
* How It Works -- DOS Floppy Boot Sector
* How It Works -- OS2 Boot Sector
* How It Works -- Partition Tables

DOS FLOPPY DISK BOOT SECTOR

This article is a disassembly of a floppy disk boot sector for a DOS floppy. The boot sector of a floppy disk is located at cylinder 0, head 0, sector 1. This sector is created by a floppy disk formating program, such as the DOS FORMAT program. The boot sector of a FAT hard disk partition has a similar layout and function. Basically a bootable FAT hard disk partition looks like a big floppy during the early stages of the system's boot processing.

At the completion of your system's Power On Self Test (POST), INT 19 is called. Usually INT 19 tries to read a boot sector from the first floppy drive. If a boot sector is found on the floppy disk, the that boot sector is read into memory at location 0000:7C00 and INT 19 jumps to memory location 0000:7C00. However, if no boot sector is found on the first floppy drive, INT 19 tries to read the MBR from the first hard drive. If an MBR is found it is read into memory at location 0000:7c00 and INT 19 jumps to memory location 0000:7c00. The small program in the MBR will attempt to locate an active (bootable) partition in its partition table. If such a partition is found, the boot sector of that partition is read into memory at location 0000:7C00 and the MBR program jumps to memory location 0000:7C00. Each operating system has its own boot sector format. The small program in the boot sector must locate the first part of the operating system's kernel loader program (or perhaps the kernel itself or perhaps a "boot manager program") and read that into memory.

INT 19 is also called when the CTRL-ALT-DEL keys are used. On

most systems, CTRL-ALT-DEL causes an short version of the POST to be executed before INT 19 is called.

=====

Where stuff is:

The BIOS Parameter Block (BPB) starts at offset 0.
The boot sector program starts at offset 3e.
The messages issued by this program start at offset 19e.
The DOS hidden file names start at offset 1e6.
The boot sector signature is at offset 1fe.

Here is a summary of what this thing does:

1) Copy Diskette Parameter Table which is pointed to by INT 1E.
2) Alter the copy of the Diskette Parameter Table.
3) Alter INT 1E to point to altered Diskette Parameter Table.
4) Do INT 13 AH=00, disk reset call.
5) Compute sector address of root directory.
6) Read first sector of root directory into 0000:0500.
7) Confirm that first two directory entries are for IO.SYS and MSDOS.SYS.
8) Read first 3 sectors of IO.SYS into 0000:0700 (or 0070:0000).
9) Leave some information in the registers and jump to IO.SYS at 0070:0000.

NOTE:

This program uses the CHS based INT 13H AH=02 to read the FAT root directory and to read the IO.SYS file. If the drive is >528MB, this CHS must be a translated CHS (or L-CHS, see my BIOS TYPES document). Except for internal computations no addresses in LBA form are used, another reason why LBA doesn't solve the >528MB problem.

=====

Here is the entire sector in hex and ascii.

```
OFFSET 0 1 2 3  4 5 6 7  8 9 A B  C D E F  *0123456789ABCDEF*
000000 eb3c904d 53444f53 352e3000 02010100 *.<.MSDOS5.0.....*
000010 02e00040 0bf00900 12000200 00000000 *...@............*
000020 00000000 0000295a 5418264e 4f204e41 *......)ZT.&NO NA*
000030 4d452020 20204641 54313220 2020fa33 *ME    FAT12   .3*
000040 c08ed0bc 007c1607 bb780036 c5371e56 *.....|...x.6.7.V*
000050 1653bf3e 7cb90b00 fcf3a406 1fc645fe *.S.>|.........E.*
000060 0f8b0e18 7c884df9 894702c7 073e7cfb *....|.M..G...>|.*
000070 cd137279 33c03906 137c7408 8b0e137c *..ry3.9..|t....|*
000080 890e207c a0107cf7 26167c03 061c7c13 *.. |..|.&.|...|.*
```

```
000090 161e7c03 060e7c83 d200a350 7c891652 *..|...|....P|..R*
0000a0 7ca3497c 89164b7c b82000f7 26117c8b *|.I|..K|. ..&.|.*
0000b0 1e0b7c03 c348f7f3 0106497c 83164b7c *..|..H....I|..K|*
0000c0 00bb0005 8b16527c a1507ce8 9200721d *......R|.P|...r.*
0000d0 b001e8ac 0072168b fbb90b00 bee67df3 *.....r.....}.*
0000e0 a6750a8d 7f20b90b 00f3a674 18be9e7d *.u... .....t...}*
0000f0 e85f0033 c0cd165e 1f8f048f 4402cd19 *._.3...^....D...*
000100 585858eb e88b471a 48488a1e 0d7c32ff *XXX...G.HH...|2.*
000110 f7e30306 497c1316 4b7cbb00 07b90300 *....I|..K|......*
000120 505251e8 3a0072d8 b001e854 00595a58 *PRQ.:.r....T.YZX*
000130 72bb0501 0083d200 031e0b7c e2e28a2e *r.........|....*
000140 157c8a16 247c8b1e 497ca14b 7cea0000 *.|..$|..I|.K|...*
000150 7000ac0a c07429b4 0ebb0700 cd10ebf2 *p....t).........*
000160 3b16187c 7319f736 187cfec2 88164f7c *;..|s..6.|....O|*
000170 33d2f736 1a7c8816 257ca34d 7cf8c3f9 *3..6.|..%|.M|...*
000180 c3b4028b 164d7cb1 06d2e60a 364f7c8b *.....M|.....6O|.*
000190 ca86e98a 16247c8a 36257ccd 13c30d0a *.....$|.6%|.....*
0001a0 4e6f6e2d 53797374 656d2064 69736b20 *Non-System disk *
0001b0 6f722064 69736b20 6572726f 720d0a52 *or disk error..R*
0001c0 65706c61 63652061 6e642070 72657373 *eplace and press*
0001d0 20616e79 206b6579 20776865 6e207265 * any key when re*
0001e0 6164790d 0a00494f 20202020 20205359 *ady...IO      SY*
0001f0 534d5344 4f532020 20535953 000055aa *SMSDOS   SYS..U.*
```

=====

The first 62 bytes of a boot sector are known as the BIOS Parameter Block (BPB). Here is the layout of the BPB fields and the values they are assigned in this boot sector:

```
  db JMP instruction      at 7c00 size  2 = eb3c
  db NOP instruction        7c02       1   90
  db OEMname                7c03       8   'MSDOS5.0'
  dw bytesPerSector         7c0b       2   0200
  db sectPerCluster         7c0d       1   01
  dw reservedSectors        7c0e       2   0001
  db numFAT                 7c10       1   02
  dw numRootDirEntries      7c11       2   00e0
  dw numSectors             7c13       2   0b40 (ignore numSectorsHuge)
  db mediaType              7c15       1   f0
  dw numFATsectors          7c16       2   0009
  dw sectorsPerTrack        7c18       2   0012
  dw numHeads               7c1a       2   0002
  dd numHiddenSectors       7c1c       4   00000000
  dd numSectorsHuge         7c20       4   00000000
  db driveNum               7c24       1   00
  db reserved               7c25       1   00
  db signature              7c26       1   29
  dd volumeID               7c27       4   5a541826
  db volumeLabel            7c2b      11   'NO NAME   '
```

```
  db fileSysType            7c36      8   'FAT12   '
```

=====

Here is the boot sector...

The first 3 bytes of the BPB are JMP and NOP instructions.

```
0000:7C00 EB3C       JMP     START
0000:7C02 90         NOP
```

Here is the rest of the BPB.

```
0000:7C00 .....4d 53444f53 352e3000 02010100 *   MSDOS5.0.....*
0000:7C10 02e00040 0bf00900 12000200 00000000 *...@............*
0000:7C20 00000000 0000295a 5418264e 4f204e41 *......)ZT.&NO NA*
0000:7C30 4d452020 20204641 54313220 2020.... *ME    FAT12     *
```

Now pay attention here...

The 11 bytes starting at 0000:7c3e are immediately overlaid by information copied from another part of memory. That information is the Diskette Parameter Table. This data is pointed to by INT 1E. This data is:

7c3e = Step rate and head unload time.
7c3f = Head load time and DMA mode flag.
7c40 = Delay for motor turn off.
7c41 = Bytes per sector.
7c42 = Sectors per track.
7c43 = Intersector gap length.
7c44 = Data length.
7c45 = Intersector gap length during format.
7c46 = Format byte value.
7c47 = Head settling time.
7c48 = Delay until motor at normal speed.

The 11 bytes starting at 0000:7c49 are also overlaid by the following data:

7c49 - 7c4c = diskette sector address (as LBA)
of the data area.
7c4d - 7c4e = cylinder number to read from.
7c4f - 7c4f = sector number to read from.
7c50 - 7c53 = diskette sector address (as LBA)
of the root directory.

```
          START:                START OF BOOT SECTOR PROGRAM

0000:7C3E FA        CLI                 interrupts off
```

```
0000:7C3F 33C0       XOR    AX,AX            set AX to zero
0000:7C41 8ED0       MOV    SS,AX            SS is now zero
0000:7C43 BC007C     MOV    SP,7C00          SP is now 7c00
0000:7C46 16         PUSH   SS               also set ES
0000:7C47 07         POP    ES                to zero

                  The INT 1E vector is at 0000:0078.
                  Get the address that the vector points to
                  into the DS:SI registers.

0000:7C48 BB7800     MOV    BX,0078          BX is now 78
0000:7C4B 36         SS:
0000:7C4C C537       LDS    SI,[BX]          DS:SI is now [0:78]
0000:7C4E 1E         PUSH   DS               save DS:SI --
0000:7C4F 56         PUSH   SI               saves param tbl addr
0000:7C50 16         PUSH   SS               save SS:BX --
0000:7C51 53         PUSH   BX               saves INT 1E address

                  Move the diskette param table to 0000:7c3e.

0000:7C52 BF3E7C     MOV    DI,7C3E          DI is address of START
0000:7C55 B90B00     MOV    CX,000B          count is 11
0000:7C58 FC         CLD                     clear direction
0000:7C59 F3         REPZ                    move the diskette param
0000:7C5A A4         MOVSB                   table to 0000:7c3e
0000:7C5B 06         PUSH   ES               also set DS
0000:7C5C 1F         POP    DS               to zero

                  Alter some of the diskette param table data.

0000:7C5D C645FE0F   MOV    BYTE PTR [DI-02],0F  change head settle time
                                                 at 0000:7c47
0000:7C61 8B0E187C   MOV    CX,[7C18]        sectors per track
0000:7C65 884DF9     MOV    [DI-07],CL       save at 0000:7c42

                  Change INT 1E so that it points to the
                  altered Diskette param table at 0000:7c3e.

0000:7C68 894702     MOV    [BX+02],AX       change INT 1E segment
0000:7C6B C7073E7C   MOV    WORD PTR [BX],7C3E   change INT 1E offset

                  Call INT 13 with AX=0000, disk reset, so
                  that the new diskette param table is used.

0000:7C6F FB         STI                     interrupts on
0000:7C70 CD13       INT    13               do diskette reset call
0000:7C72 7279       JB     TALK             jmp if any error

                  Detemine the starting sector address of
                  the root directory as an LBA.
```

```
0000:7C74 33C0       XOR     AX,AX              AX is now zero
0000:7C76 3906137C   CMP     [7C13],AX          number sectros zero?
0000:7C7A 7408       JZ      SMALL_DISK         yes
0000:7C7C 8B0E137C   MOV     CX,[7C13]          number of sectors
0000:7C80 890E207C   MOV     [7C20],CX          save in huge num sects

      SMALL_DISK:

0000:7C84 A0107C     MOV     AL,[7C10]          number of FAT tables
0000:7C87 F726167C   MUL     WORD PTR [7C16]    number of fat sectors
0000:7C8B 03061C7C   ADD     AX,[7C1C]          number of hidden sectors
0000:7C8F 13161E7C   ADC     DX,[7C1E]          number of hidden sectors
0000:7C93 03060E7C   ADD     AX,[7C0E]          number of reserved sectors
0000:7C97 83D200     ADC     DX,+00             number of reserved sectors
0000:7C9A A3507C     MOV     [7C50],AX          save start addr
0000:7C9D 8916527C   MOV     [7C52],DX          of root dir (as LBA)
0000:7CA1 A3497C     MOV     [7C49],AX          save start addr
0000:7CA4 89164B7C   MOV     [7C4B],DX          of root dir (as LBA)

                  Determine sector address of first sector
                  in the data area as an LBA.

0000:7CA8 B82000     MOV     AX,0020            size of a dir entry (32)
0000:7CAB F726117C   MUL     WORD PTR [7C11]    number of root dir entries
0000:7CAF 8B1E0B7C   MOV     BX,[7C0B]          bytes per sector
0000:7CB3 03C3       ADD     AX,BX
0000:7CB5 48         DEC     AX
0000:7CB6 F7F3       DIV     BX
0000:7CB8 0106497C   ADD     [7C49],AX          add to start addr
0000:7CBC 83164B7C00 ADC     WORD PTR [7C4B],+00  of root dir (as LBA)

                  Read the first root dir sector into 0000:0500.

0000:7CC1 BB0005     MOV     BX,0500            addr to read into
0000:7CC4 8B16527C   MOV     DX,[7C52]          get start of address
0000:7CC8 A1507C     MOV     AX,[7C50]          of root dir (as LBA)
0000:7CCB E89200     CALL    CONVERT            call conversion routine
0000:7CCE 721D       JB      TALK               jmp is any error
0000:7CD0 B001       MOV     AL,01              read 1 sector
0000:7CD2 E8AC00     CALL    READ_SECTORS       read 1st root dir sector
0000:7CD5 7216       JB      TALK               jmp if any error
0000:7CD7 8BFB       MOV     DI,BX              addr of 1st dir entry
0000:7CD9 B90B00     MOV     CX,000B            count is 11
0000:7CDC BEE67D     MOV     SI,7DE6            addr of file names
0000:7CDF F3         REPZ                       is this "IO.SYS"?
0000:7CE0 A6         CMPSB
0000:7CE1 750A       JNZ     TALK               no
0000:7CE3 8D7F20     LEA     DI,[BX+20]         addr of next dir entry
0000:7CE6 B90B00     MOV     CX,000B            count is 11
```

```
0000:7CE9 F3        REPZ                    is this "MSDOS.SYS"?
0000:7CEA A6        CMPSB
0000:7CEB 7418      JZ     FOUND_FILES      they are equal

          TALK:

                    Display "Non-System disk..." message,
                    wait for user to hit a key, restore
                    the INT 1E vector and then
                    call INT 19 to start boot processing
                    all over again.

0000:7CED BE9E7D    MOV    SI,7D9E          "Non-System disk..."
0000:7CF0 E85F00    CALL   MSG_LOOP         display message
0000:7CF3 33C0      XOR    AX,AX            INT 16 function
0000:7CF5 CD16      INT    16               read keyboard
0000:7CF7 5E        POP    SI               get INT 1E vector's
0000:7CF8 1F        POP    DS                 address
0000:7CF9 8F04      POP    [SI]             restore the INT 1E
0000:7CFB 8F4402    POP    [SI+02]            vector's data
0000:7CFE CD19      INT    19               CALL INT 19 to try again

        SETUP_TALK:

0000:7D00 58        POP    AX               pop junk off stack
0000:7D01 58        POP    AX               pop junk off stack
0000:7D02 58        POP    AX               pop junk off stack
0000:7D03 EBE8      JMP    TALK               now talk to the user

       FOUND_FILES:

                    Compute the sector address of the first
                    sector of IO.SYS.

0000:7D05 8B471A    MOV    AX,[BX+1A]         get starting cluster num
0000:7D08 48        DEC    AX                 subtract 1
0000:7D09 48        DEC    AX                 subtract 1
0000:7D0A 8A1E0D7C  MOV    BL,[7C0D]          sectors per cluster
0000:7D0E 32FF      XOR    BH,BH
0000:7D10 F7E3      MUL    BX                 multiply
0000:7D12 0306497C  ADD    AX,[7C49]          add start addr of
0000:7D16 13164B7C  ADC    DX,[7C4B]            root dir (as LBA)

                    Read IO.SYS into memory at 0000:0700.  IO.SYS
                    is 3 sectors long.

0000:7D1A BB0007    MOV    BX,0700            address to read into
0000:7D1D B90300    MOV    CX,0003            read 3 sectors

         READ_LOOP:
```

```
                    Read the first 3 sectors of IO.SYS
                    (IO.SYS is much longer than 3 sectors).

0000:7D20 50         PUSH    AX                save AX
0000:7D21 52         PUSH    DX                save DX
0000:7D22 51         PUSH    CX                save CX
0000:7D23 E83A00      CALL    CONVERT             call conversion routine
0000:7D26 72D8       JB      SETUP_TALK          jmp if error
0000:7D28 B001       MOV     AL,01               read one sector
0000:7D2A E85400      CALL    READ_SECTORS        read one sector
0000:7D2D 59         POP     CX                restore CX
0000:7D2E 5A         POP     DX                restore DX
0000:7D2F 58         POP     AX                restore AX
0000:7D30 72BB       JB      TALK                jmp if any INT 13 error
0000:7D32 050100      ADD     AX,0001             add one to the sector addr
0000:7D35 83D200      ADC     DX,+00              add one to the sector addr
0000:7D38 031E0B7C    ADD     BX,[7C0B]           incr mem addr by sect size
0000:7D3C E2E2       LOOP    READ_LOOP           read next sector

                    Leave information in the AX, BX, CX and DX
                    registers for IO.SYS to use.  Finally,
                    jump to IO.SYS at 0070:0000.

0000:7D3E 8A2E157C    MOV     CH,[7C15]           media type
0000:7D42 8A16247C    MOV     DL,[7C24]           drive number
0000:7D46 8B1E497C    MOV     BX,[7C49]           get start addr of
0000:7D4A A14B7C      MOV     AX,[7C4B]             root dir (as LBA)
0000:7D4D EA00007000  JMP     0070:0000           JUMP TO 0070:0000

          MSG_LOOP:

                    This routine displays a message using
                    INT 10 one character at a time.
                    The message address is in DS:SI.

0000:7D52 AC         LODSB                     get message character
0000:7D53 0AC0        OR      AL,AL               end of message?
0000:7D55 7429       JZ      RETURN              jmp if yes
0000:7D57 B40E       MOV     AH,0E               display one character
0000:7D59 BB0700      MOV     BX,0007             video attrbiutes
0000:7D5C CD10       INT     10                display one character
0000:7D5E EBF2       JMP     MSG_LOOP            do again

          CONVERT:
                    This routine
                    converts a sector address (an LBA) to
                    a CHS address.  The LBA is in DX:AX.

0000:7D60 3B16187C    CMP     DX,[7C18]           hi part of LBA > sectPerTrk?
```

```
0000:7D64 7319        JNB     SET_CARRY           jmp if yes
0000:7D66 F736187C    DIV     WORD PTR [7C18]     div by sectors per track
0000:7D6A FEC2        INC     DL                  add 1 to sector number
0000:7D6C 88164F7C    MOV     [7C4F],DL           save sector number
0000:7D70 33D2        XOR     DX,DX               zero DX
0000:7D72 F7361A7C    DIV     WORD PTR [7C1A]     div number of heads
0000:7D76 8816257C    MOV     [7C25],DL           save head number
0000:7D7A A34D7C      MOV     [7C4D],AX           save cylinder number
0000:7D7D F8          CLC                         clear carry
0000:7D7E C3          RET                         return

        SET_CARRY:

0000:7D7F F9          STC                         set carry

          RETURN:

0000:7D80 C3          RET                         return

      READ_SECTORS:

                        The caller of this routine supplies:
                          AL = number of sectors to read
                          ES:BX = memory location to read into
                          and CHS address to read from in
                          memory locations 7c25 and 7C4d-7c4f.

0000:7D81 B402        MOV     AH,02               INT 13 read sectors
0000:7D83 8B164D7C    MOV     DX,[7C4D]           get cylinder number
0000:7D87 B106        MOV     CL,06               shift count
0000:7D89 D2E6        SHL     DH,CL               shift upper cyl left 6 bits
0000:7D8B 0A364F7C    OR      DH,[7C4F]           or in sector number
0000:7D8F 8BCA        MOV     CX,DX               move to CX
0000:7D91 86E9        XCHG    CH,CL               CH=cyl lo, CL=cyl hi + sect
0000:7D93 8A16247C    MOV     DL,[7C24]           drive number
0000:7D97 8A36257C    MOV     DH,[7C25]           head number
0000:7D9B CD13        INT     13                  read sectors
0000:7D9D C3          RET                         return
```

Data not used.

```
0000:7D90 ca86e98a 16247c8a 36257ccd 13c3.... *.....$|.6%|...  *
```

Messages here.

```
0000:7D90 ........ ........ ........ ....0d0a *               ..*
0000:7Da0 4e6f6e2d 53797374 656d2064 69736b20 *Non-System disk *
0000:7Db0 6f722064 69736b20 6572726f 720d0a52 *or disk error..R*
0000:7Dc0 65706c61 63652061 6e642070 72657373 *eplace and press*
0000:7Dd0 20616e79 206b6579 20776865 6e207265 * any key when re*
```

0000:7De0 6164790d 0a00.... .... *ady...   *

MS DOS hidden file names (first two root directory entries).

0000:7De0 ....494f 20202020 20205359 *   IO   SY*
0000:7Df0 534d5344 4f532020 20535953 000055aa *SMSDOS   SYS..U.*

The last two bytes contain a 55AAH signature.

0000:7Df0 ....55aa *   U.*

/end/

Facts and Fiction (FnF)

Volume 1

by Hale Landis <landis@sugs.tware.com>

FnF Volume 1 contains articles that were written during 1995. Most of these articles were posted to the Usenet newsgroup comp.sys.ibm.pc.hardware.storage. Some were answers to private email.

FnF Volume 1 Index:

Logical Block Addressing (LBA)

LBA DOESN'T SOLVE THE >528MB PROBLEM!

There continues to be a large amount of very false information floating around in this newsgroup concerning LBA addressing on ATA (IDE) drives.

Please, lets get this correct...

* LBA has NOTHING to do with fixing the >528MB problem.

* LBA does NOT solve the >528MB problem.

* It is not true that LBA is needed to support a drive >528MB.

An INT 13H BIOS that does CHS translation is the ONLY way to support a drive >528MB today. CHS translation enables the use of >16 heads with <=1024 cylinders at the INT 13H interface while using <=16 heads and >1024 cylinders (or maybe LBA) at the drive interface.

FDISK uses CHS. It does not use LBA. It does not know (or care) if LBA is being used at the device interface. FDISK can not address any part of a disk that can not be accessed in CHS mode.

Microsoft operating systems WILL NOT use any part of a disk that can not be accessed in CHS mode. Read the Microsoft documentation carefully. OS/2 and Linux do have a special FDISK "hack" that allows partitions to span or be beyond the 528MB boundary even on systems that don't have a CHS translating BIOS and even on drives that don't support LBA. Be very careful when using this feature of OS/2 or Linux.

LBA provides only a slight performance improvement for some protected mode operating systems, HOWEVER, those systems MUST still boot in CHS mode and use CHS mode to understand the drive configuration data returned by INT 13H AH=08H and to understand the partition layout of the drive. If an OS decides to use LBA addressing it MUST not attempt to access any sectors that are beyond those that can be accessed by the CHS mode returned by INT 13H AH=08H. Therefore, the size of a drive in LBA mode is the same as the size in CHS mode. If the INT 13H BIOS does not support drives >528MB then LBA mode can not be used make the drive look >528MB either.

So, once again, LBA does not solve the >528MB problem. Don't be confused by the the BIOS vendors that claim LBA solves the >528MB problem. They are confusing you by using LBA to explain that their BIOS is really doing CHS translation. It is very unfortunate that they are using LBA so incorrectly. We can probably thank Western Digital's EIDE Implementation Guide for starting this confusion.

NOTE: If you are about to send me email to tell me how wrong I am, please see my "How It Works" series of documents that describe how the MBR and boot sectors work, or, disassemble the master boot record on your system and look at how your system boots. You will find that it does not use LBA at all. If after doing this you still think I'm wrong, then send me email. Thanks!

More About LBA

Question: Which is better: LARGE or LBA?

Hmmm... The truth is this: The LBA option is not stable and is not the "standard" that will survive in the future. I've posted several articles in the past about this but the misleading

information comes faster than I and others can keep up.

The truth is that the IBM/MS/Phoenix extended/enhanced BIOS specification *is* the future "standard" and it is being widely implemented. This "standard" does not require the use of LBA at the drive interface. Using LBA at the device interface never did and never will solve the >528MB (>1024 cylinder) problem. Only CHS translation at the INT 13 interface or LBA at the INT 13 interface solves this problem.

The truth is that the LBA BIOS option is a poorly designed and very poorly documented Western Digital idea that should have never been adopted by the PC industry. Even WD doesn't actively support it any more.

Maybe this table will help everyone understand this problem...

drive size	INT 13 interface	IDE/EIDE drive interface
<528MB and <1024 cyl	any old BIOS works -- no CHS translation is required.	CHS used here is the same as the CHS used at the INT 13 interface.
>528MB or >1024 cyl but less than 8GB	CHS translation required -- keep cylinders under 1024 and use more than 16 heads implementation #1 is described by IBM/MS/Phoenix documents. implementation #2 is described by the WD EIDE guide.	two possible implementations exist: #1 the "standard" or LARGE implementation uses CHS at the drive interface and generally gives better performance. #2 the WD EIDE BIOS or LBA implementation uses LBA at the drive interface (for no good reason). This is not the "standard" of the future and it can confuse some OS device drivers -- this is why so many versions of the Windows FastDisk (32-bit disk access) driver exist today. This BIOS implementation does not work for drives of 8GB or bigger.
8GB	IBM/MS/Phoenix	either CHS or LBA can be

or bigger "standard" is the used at the device interface.
only thing that
works. LBA is used
at the INT 13
interface!

Note: The IBM/MS/Phoenix "standard" works for all drive sizes including drives with up to 2^64 sectors (really big drives!). The WD LBA BIOS implemetation fails when drives get to 8GB and it also doesn't include any of the Plug-and-Play stuff that future operating systems will require.

I really think that someone at WD got confused about how LBA should be used. It appears to me that this WD person (or persons) thought that using LBA at the device interface would solve the big disk problem. What they overlooked was this: LBA at the INT 13 interface is what really solves the big disk problems. LBA at the device interface isn't needed to solve any big disk problem (even for 100GB drives)!

As I've said before, those of you that are using the LBA option in your BIOS setup *may* find that some day you'll have to back up all your data and switch to the LARGE option in order to remain compatible with the real BIOS "standard".

Even More About LBA

Question: What does LBA do for me?

Some people in this message thread seem to be confused here about the use of LBA to:

1) get around the 528MB problem,
2) reformatting the disk because of LBA,
3) performance increase due to LBA.

I'm not sure were all this started but, lets take a brief look at each of these...

1) LBA does not solve the >528MB problem. Only a BIOS with an INT 13 that does CHS translation solves this problem. How the BIOS sends sector addresses to the disk drive, as CHS or as LBA, should not change the CHS that is used at the INT 13 interface.

   Yes, there is at least one BIOS type out there that has a problem with this: any BIOS based on the Western Digital EIDE Implementation Guide may change the CHS used at the INT 13

interface when the use of LBA at the device interface is turned on or off. This is a very flawed implementation and should be avoided OR if you have such a BIOS, NOT do change the LBA setting in the CMOS setup once you have partitioned your disk and installed your software.

2) As long as the CHS at the INT 13 interface does not change, it should make not difference if CHS or LBA is being used at the drive's interface. CHS and LBA are completely interchangeable at the drive interface on a command by command basis.

However, beware of these WD EIDE BIOS types. These BIOS incorrectly change the CHS used at the INT 13 interface when LBA use at the drive interface is changed. This should not be. This is the major flaw in the WD EIDE BIOS type. It represents a serious threat to your data!

3) LBA use will not give any great performance increase. In fact, on some drives it may decrease performance. The disk drive industry has had many years to optimize the use of CHS at the drive's interface. Many drives do NOT convert the CHS at the interface to an LBA internally. In these drives, using LBA at the drive's interface just causes the drive more overhead as it must convert the LBA to a CHS before it can proceed with the command.

Some more LBA info...

Q: Is LBA slower?

In article <...> mdschus@ibm.net writes:
>..., no it is not slower, just a different geometry translation,
>and at the same speed as long as the system supports LBA,
>and is set up correctly.

Beware... LBA *can* be slower for two reasons:

a) Conversion of INT 13 CHS input to an LBA at the device interface (when LBA is used in BIOS setup) usually requires more instructions than conversion of INT 13 CHS input to another CHS at the device interface (when CHS translation is used in BIOS setup, aka LARGE);

b) LBA processing in many drives is slower than CHS processing (this will change as drive vendors optimize their hardware and firmware for LBA in future drives). Many people can measure a slight difference in performance between CHS and LBA today.

Low Level Formating

Question: What does low-level format do on an ATA (IDE) drive?

Depends on how old the ATA drive is and who made it.

Lets talk about what low-level formatting is...

> In the old days of MFM, RLL and ESDI, a new hard disk drive was just like a new unformatted floppy -- there was nothing on it. The drive had to be connected to a controller and the controller had to be told how many sectors per track to lay down on each track -- this is the same thing you do when you format a floppy these days. Formatting of each track creates the inter sector gaps, the sector ID fields and the sector data fields. Most MFM/RLL/ESDI controllers include a small low-level formatting program in ROM. You generally use DOS DEBUG to enter this program.

Lets talk about what high-level formatting is...

> High level formatting creates a file system within a partition of a hard disk or on a floppy. This process writes the initial version of the boot record, root directory and file allocation table (FAT).

Here is a bit of confusion -- when you use the DOS FORMAT command to format a floppy, you are doing BOTH a low-level and high level format at the same time. When you use the DOS FDISK and FORMAT commands to format a hard disk, all you are doing is the high level format.

Back to hard disks...

MFM/RLL/ESDI and some older ATA drives use the same controller command code, known as Format Track, to do the low-level format of a track. This command is issued once for each track on the hard disk. This command writes the inter sector gaps, the sector ID fields and the sector data fields. Each sector on a track has an ID -- an 8-bit binary number usually starting at 1. The order is which the sector IDs are written determines the interleave. If the sector IDs are written as 1, 2, 3, ..., n, this is 1-to-1 interleave. When written as 1, n, 2, n+1, 3, ..., n-1, you have 2-to-1 interleave. MFM controllers usually used 2-to-1 or 3-to-1 interleave with 17 sectors per track. RLL usually used 26 sectors per track.

MFM/RLL/ESDI also would allow "marking" a sector "bad" as the sector ID field was being created. This would cause a very special kind of error on any read or write command that attempted

to access this sector -- a Bad Block error. When you run the DOS FORMAT program, it reads every sector in the partition looking for "bad" sectors and sectors with uncorrectable data errors. Such sectors then cause clusters of sectors in the FAT to be marked bad and DOS will never use them. Other systems, such as OS/2 HPFS, Unix, etc, keep bad block lists as part of their file system data and they also do not access these "bad" sectors.

MFM/RLL/ESDI drive technology was generally based on using a stepper motor to position the read/write heads. Over time the bearings in the drive would wear and read/write errors would appear because the stepper motor was no longer positioning the read/write heads in the right place. The solution was to do a low-level format again.

New drives don't use stepper motors and instead use embedded servo bursts in between the sectors. These burst are used to locate tracks and sectors and to keep the read/write heads properly positioned even when the drive's bearing are worn. More important is keeping the read/write heads properly positioned under a wide range of temperatures. Thermal expansion can change the length of the arm the read/write head is attached to by as much as 5 or 10 tracks! This is the reason you hear new drives doing a lot of thermal calibration (that strange seeking noise you hear when there should be no disk activity).

The servo bursts are written at the factory in a very controlled environment using some very expensive equipment. The drive alone can not recreate these servo data bursts. Likewise, because most drives are now zone recorded (they have different number of sectors per track at different locations on the media), the inter sector gaps, sector ID fields and sector data fields are also written at the factory and can not be recreated later. Some drives may soon do away with the sector ID fields and some of the gaps in order to increase data storage capacity.

Now for the history the ATA Format Track command...

Early ATA drives did not really implement the Format Track command -- it was thought to be obsolete (and it was). What was implemented was a simple writing of some data pattern into each sector of each track formatted. Most drives did not support marking sectors bad. However, the disk drive industry is driven by features and slowly, one-by-one, the hard disk vendors started implementing the command such that it did the same thing as in MFM/RLL/ESDI. Then someone implemented the ability to "reassign" a sector's address to a different physical sector on the disk. Never mind that there were *no* programs then and few now that use the Format Track command to do this.

The original ATA specification (now at rev 4.0c, also known as ATA-1) documents the "full" function Format Track command but leaves it to the drive vendor to decide what a drive will really do. It recommends a minimum action of writing binary zero into the data field of each sector formatted. The ATA-2 specification says that the function of the command is "vendor specific" -- it doesn't even recommend the minimum action of writing binary zero data -- a major step towards (finally) making the command obsolete.

So what does low-level formatting do on a modern ATA drive?

Assuming that you can find a program that really does issue the ATA Format Track command your ATA drive probably doesn't do anything other than write some data pattern, maybe binary zeroes, into the data field of every sector on the drive. That is no different than just using the normal write command and writing data into every sector on the drive. In general, low-level formatting of an ATA drive is just a big waste of time.

So several things have gotten us to this time and place:

1) No stepper motors.
2) Very close track spacing.
3) Embedded servo data.
4) Zone recording.
5) Few programs that use the Format Track command.
6) The Format Track command DOUBLES the amount of firmware in a drive.
7) And then there is the failure rate problem... There are two major components in a disk drive -- the Head/Disk Assembly (HDA, contains the media and read/write heads) and the printed circuit board with the electronics -- which fails most frequently?

THE ELECTRONICS!

Question: Wait a minute... Are you trying to say the electonics are more unreliable than the HDA (media and heads)?

In general the failure rate for the electronics is slightly higher than the failure rate of the media and heads. High temperatures and heating/cooling cycles cause failure of the little gold wires inside the chips. This could be a good reason to make sure your drive has adequate cooling and leave it on most of the time. But heat is the big problem here -- it is just a matter of time before the HDA or the electronic will fail. Of course rough handling (like dropping the drive) will probably not damage the electronics but will damage heads or media.

Zone Bit Recording (ZBR)

ZBR maintains a constant data bit density across the disk surface. This done by placing more sectors on tracks at the outside of the disk and fewer sectors at the inside edge of the disk. Typically, drives have >60 sectors on the outside tracks and <40 on the inside tracks. This does cause the disk's raw data rate to change. The data rate is higher on the outside than on the inside. Most ZBR drives have at least three zones. Some my have 30 or more. All of the tracks within a zone have the same number of sectors per track.

ZBR and embedded servo data are the two major reasons you can't low level format drives anymore.

Dual Channel ATA Host Adapters and Data Corruption

WARNING! POSSIBLE SYSTEM HANGS AND DATA CORRUPTION PROBLEMS!

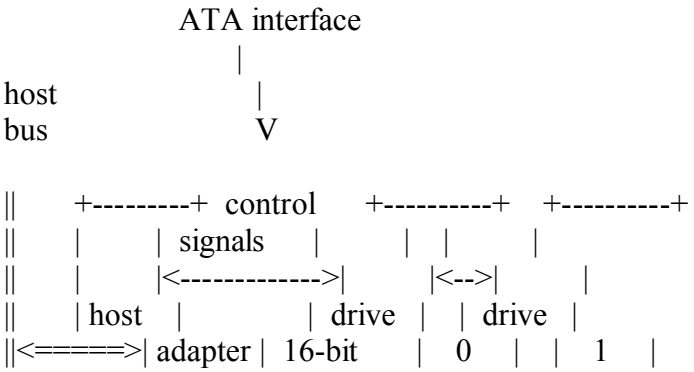
If you have a dual channel ATA (IDE) host adapter (or you are thinking of buying one) either an add in card or on a new motherboard, read this.

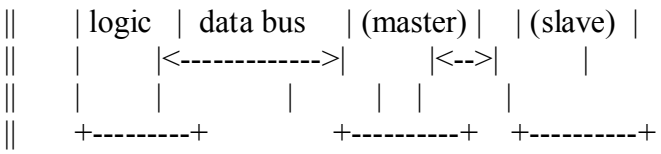
First, ATA is the real name for IDE or EIDE.

Second, dual channel ATA host adapters are two ATA host adapters in one package. These are add in cards with two ATA connectors or a motherboard with two ATA connectors. You can attach up to two ATA devices per host adapter (up to two devices per cable).

Normally one host adapter will be assigned to the "primary" I/O addresses (1F0-1F7H and 3F6H) and the other host adapter will be assigned to the "secondary" I/O addresses (1790177H and 376H).

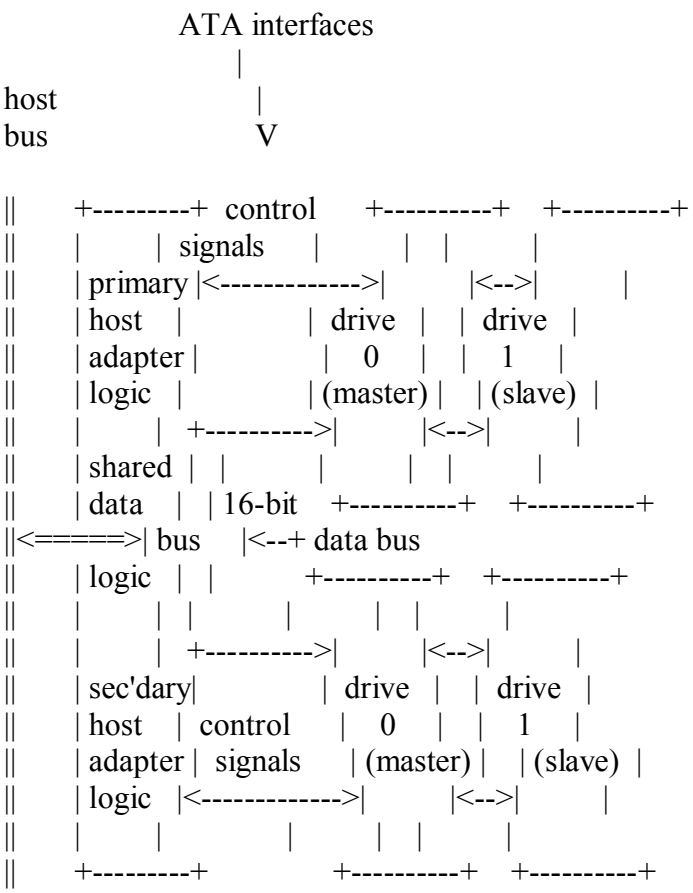
A single ATA host adapter looks like this internally:





A dual channel host adapter should have two complete ATA host adapters with no shared logic, no shared signals, no shared functions of any kind.

According to some estimates, up to 30 percent of all dual channel host adapters now on the market (as boards or as on motherboard designs) have a serious flaw. This flaw can result in data corruption. The flaw is that these dual channel host adapters "share" the data bus between the two ATA cables. This results in a host adapter that looks like this internally:



Data corruption can come from two sources in this design:

1) the minor reason is that the data bus exceeds the ATA 18 inch maximum length. This design makes the two cables look like one 36 inch cable.

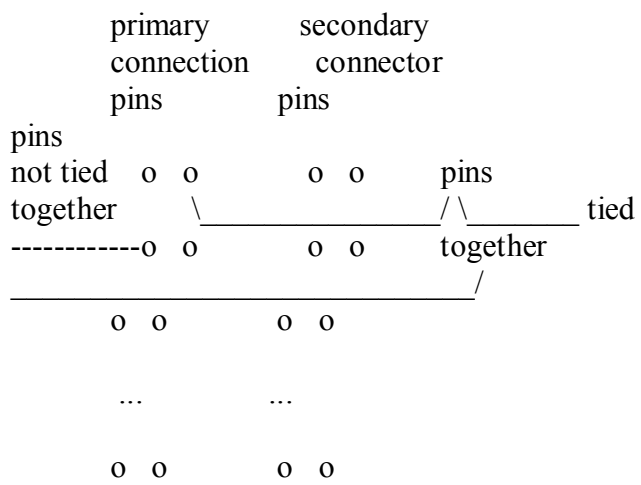
2) the major reason is that a multitasking operating system expects to be able to perform an I/O operation at the same time on both the primary and secondary host adapters. This

design will corrupt data (or cause hang conditions) because of the shared data bus.

How can you identify a flawed dual channel host adapter?

One way is to look at the printed circuit board and count the number of direct connections between the pins of the primary host adapter connector and secondary host adapter connector. This may be difficult on multilayer printed circuit boards. It is normal for a few signals (such as ground signals) to be tied together directly. However, if you see a large number of pins (more than 16) tied directly together by copper traces on the circuit board, you are looking at one of these flawed host adapters.

For example:



However, beware, this visual check is not foolproof! Pins that don't appear to be tied together in the area of the connectors could be tied together at some other location.

My advice: Talk to the host adapter (or motherboard) technical support people and ask them if the two host adapters share any signals, logic or functions. Any sharing of data bus signals, host adapter logic or functions (especially data transfer logic) would indicated a flawed design that could corrupt your data.

BUYER BEWARE!

If you run Linux, check the latest Linux IDE driver information for a version of the IDE driver that will serialize all I/O to the two host adapters in order to prevent strange things.

It is unclear at this time (May95) how WinNT, Win95 or OS/2 will deal with these host adapters.

BEWARE OF INTEL TRITON DUAL CHANNEL HOST ADAPTERS

The Triton is even worse than first look gave. True, it does not corrupt data or do strange things with interrupts, but it can have what I consider to be serious performance problems. Now that Intel has made the chip spec public, we can all find out that not only does the chip share the two ATA data buses but that combined bus is also shared with the ISA address/data bus function that is also in the chip. Intel claims "fair round robin" sharing of all the uses of the single bus. So if you have your serial or parallel ports on the Triton ISA bus and you have any COM or LPT activity going on this will be multiplexed with your two ATA interfaces on the same set of signals comming out of the Triton chip. So much for high performance.

Standard(?) I/O Port Addresses and Interrupt Numbers

The following table shows the most commonly supported I/O port addresses and IRQ numbers used for PC ATA (IDE/EIDE) host adapters.

Interface Number	CS0- Decode	CS1- Decode	IRQ number
1	01F0h-01F7h	03F6h-03F7h	14
2	0170h-0177h	0376h-0377h	15
3	01E8h-01EFh	03EEh-03EFh	12 or 11
4	0168h-016Fh	036Eh-036Fh	10 or 9

Now some history and notes...

a) Interface number 1 -- The Primary

The addresses and IRQ number for the first interface have been well established since the first IBM PC/AT system in 1984. This is the address and IRQ used by the MFM controller in the original IBM PC/AT. All PC software (BIOS and OS) support the primary host adapter (because they support an MFM, RLL or ESDI controller using this configuration). Remember that ATA (IDE/EIDE) is designed to operated just like an old MFM controller (of course there are new features in ATA that MFM, RLL and ESDI did not support).

b) Interface number 2 -- The Secondary

During 1995 support for second interface has become very well established. Some implementations of this interface number have used IRQ's other than 15 in the past. Today, IRQ 15 has become the accepted standard for the secondary interface.

c) The other interfaces

The addresses and IRQ's used by the 3rd and 4th interfaces is not very standard. Usually it is the ATA (IDE) interface on some other type of card, such as a sound card, that is used for the 3rd or 4th ATA host adapter in a system. There is no standardized BIOS or OS support for such configurations and usually an OS device driver is required to access these ATA host adapter addresses.

Why Disk Drives Break

Question: Why do hard drives fail?

There are three major (and these really are the only three real reasons hard disk drives fail):

1) they get to hot (your system cooling fan quits working).

2) they are mishandled (dropped or banged around).

Drives that have been over heated or mishandled can develop bad sectors as time goes on. Usually you will see one bad sector and then a few more and then a bunch more until the drive is basicly not worth using any more.

A drive that really gets too hot may refuse to spin up because the heads get "glued" to the media by the high heat.

3) an electronics failure.

Electronics failures are usually sudden and without warning. A common time for a drive's electronics to fail are on Monday morning after the drive has been powered off all weekend. It is the heating/cooling cycles that cause breaks in the printed circuit board or breaks in the little gold wires inside the chips on the printed circuit board.

You can also zap the electronics by handling a drive when you are not properly grounded.

What can I do?

Make that your drive is properly cooled, don't drop it and be grounded before you touch a drive!

OK, every once in awhile a disk vendor will make a bunch of drives that have some kind of dirt or chemical inside that should not be there. Drives with this kind of problem usually don't last very long and usually fail during the warranty period. This is a very rare thing to have happen these days but the failure mode is usually the same as described above for over heating.

Plug and Play (PnP)

As many people have found out PnP is a joke. It is a cute marketing thing to make you think that the computer hardware and software vendors have solved all the hardware and software configuration problems for you. Just plug in a new device and the system will recognize it and allow you to use it. Ha Ha!

What is PnP really? It is a bunch of uncoordinated and proprietary solutions to specific hardware or software configuration problems. Many of the so called PnP specifications were developed by individual companies or groups of companies (aka, private clubs) to enhanced their image with the computer buying public. Most of these specifications are short sighted and don't address all of the issues.

There are so many PnP specifications floating around the industry that no one has a complete list of them. New ones appear monthly. It is a major mess.

A specific example is the PnP mess related to ATA (IDE/EIDE) devices. Several years ago various hardware specificaion groups (VESA and PCI and others) attempted to defined PnP for various type of host adapters, including ATA (IDE/EIDE) host adapters. But these groups addressed only the hardware aspects of the host adapter configuration. They forgot to address the BIOS issues!

Now there is a Compaq/Phoenix/Intel "BIOS Boot Specification" (http://www.ptltd.com or ftp://www.ptltd.com in file BBSINFO.ZIP). But this specification doesn't talk about all the issues with FDISK and the rules for creating partitions on a hard disk, especially bootable partitions.

So the PnP mess continues on and grows bigger each day.

Thermal Calibration and "AV" Drives

In article <...> cvanderb@toronto.cbc.ca writes:
>The way I see it, there are (at least) two common
>myths floating around about SCSI and EIDE drives.
>
>Myth #1: SCSI has better sustained throughput than EIDE.
>
>As I understand it, the SCSI 3 bus is capable of higher
>bandwidth than EIDE on a PCI bus, but that is irrelevent,
>because the hard drives themselves don't come anywhere
>near using the available bandwidth. As far as I can tell,
>the high end of SCSI drives (single drives, anyway) are
>capable of approx 7 MB/sec sustained throughput. And
>the high-end EIDE drives (PIO Mode 4) are capable of
>about the same sustained throughput. True or false?

Nearly always FALSE (due to IDE/EIDE's extremely low command
overhead) except that there are real high capacity and real high
performance drives that come only with the SCSI interface. This
will probably change in the future.

Plus there is the old question of "how many reads are from the
drive's cache?". A cache read is just a memory (in the drive) to
memory (in the system) transfer and can, in theory, be done at
extremely high speeds (much faster than the drive's actually
media tranfer rate).

Today, for the same transfer rate, EIDE is cheaper (especially
when you include the cost of host adapters and cables).

Who knows what tomorrow will bring. There are many people in the
disk drive industry working on faster EIDE and faster SCSI data
transfer protocols. And they probably are also working on the
interface that will replace both EIDE and SCSI (for disk drives).
Neither EIDE or SCSI can keep up with the disk technology that is
just over the hill now and moving towards us very rapidly.
Expect an entirely new hard disk interface in a few years as
drives approach data transfer rates of 100+MB/second.

>Myth #2: Only drives sold as "AV" drives are capable of
>uninterrupted sustained throughput.
>
>This is based on a problem which no longer exists.
>Older drives used to pause every so often to recalibrate
>their head positions, the so-called "thermal recal".
>As I understand it, just about all drives now use
>a system of so-called servo tracks which are written
>onto the platter surfaces. Thus, as the drives get
>hotter and the material expands, the servo tracks
>expand right along with them, and so there's no need

>for thermal recal anymore. True or false?

Well, sort of TRUE and sort of FALSE. ALL drives MUST do thermal calibration every so often.

Until a few years ago most drives had "dedicated servo" systems (all servo data on one surface and only read by one head) that required frequent calibration especially if the drives internal temperature was changing rapidly (like right after power on). The so called AV versions of this type of drive attempted to reduce the chance that the host would notice this drive activity. This activity can distrub the smooth flow of video or sound data during disk read/write commands.

Today most all drives use an "embedded servo" system (servo data mixed in with user data on every data track, every head reads servo data while reading user data). While thermal calibration is still required, its impact on the smooth flow of data can be kept to an absolute mimimum. My guess is that some disk vendors will still use the AV label just because it's a good marketing tool. You may even see an AV label on an EIDE drive.

However, there is another side of this AV thing that you didn't bring up (myth#3?): reduced error recovery.

A true AV drive will implement a set of reduced error recovery read commands based on the theory that your eye will notice a pause in the flow of the video data while the drive attempts its normal full error recovery (the picture will "pause" or "jump") but your eye will probably ignore a few bad bits of video data flashing across the screen (video snow). On course you don't want your OS to use this reduced error recovery read command when it is reading real data (like a directory or you current tax return's data!).

The bottom line 99% of the time: Unless you are buying a drive that will become part of the disk array that is used to store video data, you probably don't need an AV drive. AV drives are really built for this application.

Just Who Is This Hale Landis Character Anyway?

(Ahhh... A very good question...)

Hale Landis is an ATA/ATAPI interface consultant. Mr. Landis is available to teach classes about these interfaces or help design hardware or software for these interfaces. Hale has 25+ years experience creating diagnostic test software for disk storage

subsystems and devices. He is very active in the ATA and ATAPI standards committee efforts (the bum even attends the meeting in person now and then).

Contact Hale Landis by email at

landis@sugs.tware.com

or by phone at

303-548-0567

(...but is any of this fact or fiction?)

And here is something that Hale really hates...

Someone once wrote:

> HD manufacturers think 1MB = 1e6 bytes, not 1048576 bytes.

And Hale said:

> The HD manufacturers are right: 1MB is 1000000 bytes -- it IS NOT 1048576 bytes! Mega means 1000000 (just like kilo means 1000).
>
> Is a 1MHz a frequency of 1048576 cycles/second? NO! It is 1000000 cycles/second. Is a kilometer 1024 meters? NO! It is 1000 meters.
>
> Mega, Kilo, etc are ISO defined terms that predate the computer industry by many decades!

/end FnF Volume 1/

How It Works -- Master Boot Record

Version 1a

by Hale Landis (landis@sugs.tware.com)

THE "HOW IT WORKS" SERIES

This is one of several How It Works documents. The series currently includes the following:

* How It Works -- CHS Translation
* How It Works -- Master Boot Record
* How It Works -- DOS Floppy Boot Sector
* How It Works -- OS2 Boot Sector
* How It Works -- Partition Tables

MASTER BOOT RECORD

This article is a disassembly of a Master Boot Record (MBR). The MBR is the sector at cylinder 0, head 0, sector 1 of a hard disk. An MBR is created by the FDISK program. The FDISK program of all operating systems must create a functionally similar MBR. The MBR is first of what could be many partition sectors, each one containing a four entry partition table.

At the completion of your system's Power On Self Test (POST), INT 19 is called. Usually INT 19 tries to read a boot sector from the first floppy drive. If a boot sector is found on the floppy disk, the that boot sector is read into memory at location 0000:7C00 and INT 19 jumps to memory location 0000:7C00. However, if no boot sector is found on the first floppy drive, INT 19 tries to read the MBR from the first hard drive. If an MBR is found it is read into memory at location 0000:7c00 and INT 19 jumps to memory location 0000:7c00. The small program in the MBR will attempt to locate an active (bootable) partition in its partition table. If such a partition is found, the boot sector of that partition is read into memory at location 0000:7C00 and the MBR program jumps to memory location 0000:7C00. Each operating system has its own boot sector format. The small program in the boot sector must locate the first part of the operating system's kernel loader program (or perhaps the kernel itself or perhaps a "boot manager program") and read that into memory.

INT 19 is also called when the CTRL-ALT-DEL keys are used. On most systems, CTRL-ALT-DEL causes an short version of the POST to be executed before INT 19 is called.

=====

Where stuff is:

The MBR program code starts at offset 0000.
The MBR messages start at offset 008b.
The partition table starts at offset 00be.
The signature is at offset 00fe.

Here is a summary of what this thing does:

If an active partition is found, that partition's boot record
is read into 0000:7c00 and the MBR code jumps to 0000:7c00
with SI pointing to the partition table entry that describes
the partition being booted. The boot record program uses this
data to determine the drive being booted from and the location
of the partition on the disk.

If no active partition table enty is found, ROM BASIC is
entered via INT 18. All other errors cause a system hang, see
label HANG.

NOTES (VERY IMPORTANT):

1) The first byte of an active partition table entry is 80.
This byte is loaded into the DL register before INT 13 is
called to read the boot sector. When INT 13 is called, DL is
the BIOS device number. Because of this, the boot sector read
by this MBR program can only be read from BIOS device number
80 (the first hard disk). This is one of the reasons why it
is usually not possible to boot from any other hard disk.

2) The MBR program uses the CHS based INT 13H AH=02H call to
read the boot sector of the active partition. The location of
the active partition's boot sector is in the partition table
entry in CHS format. If the drive is >528MB, this CHS must be
a translated CHS (or L-CHS, see my BIOS TYPES document).
No addresses in LBA form are used (another reason why LBA
doesn't solve the >528MB problem).

=====

Here is the entire MBR record (hex dump and ascii).

```
OFFSET 0 1 2 3  4 5 6 7  8 9 A B  C D E F  *0123456789ABCDEF*
000000 fa33c08e d0bc007c 8bf45007 501ffbfc *.3.....|..P.P...*
000010 bf0006b9 0001f2a5 ea1d0600 00bebe07 *................*
000020 b304803c 80740e80 3c00751c 83c610fe *...<.t..<.u.....*
000030 cb75efcd 188b148b 4c028bee 83c610fe *.u......L.......*
```

```
000040 cb741a80 3c0074f4 be8b06ac 3c00740b *.t..<.t.....<.t.*
000050 56bb0700 b40ecd10 5eebf0eb febf0500 *V.......^.......*
000060 bb007cb8 010257cd 135f730c 33c0cd13 *..|...W.._s.3...*
000070 4f75edbe a306ebd3 bec206bf fe7d813d *Ou.........}.=*
000080 55aa75c7 8bf5ea00 7c000049 6e76616c *U.u.....|..Inval*
000090 69642070 61727469 74696f6e 20746162 *id partition tab*
0000a0 6c650045 72726f72 206c6f61 64696e67 *le.Error loading*
0000b0 206f7065 72617469 6e672073 79737465 * operating syste*
0000c0 6d004d69 7373696e 67206f70 65726174 *m.Missing operat*
0000d0 696e6720 73797374 656d0000 00000000 *ing system......*
0000e0 00000000 00000000 00000000 00000000 *................*
0000f0 TO 0001af SAME AS ABOVE
0001b0 00000000 00000000 00000000 00008001 *................*
0001c0 0100060d fef83e00 00000678 0d000000 *......>....x....*
0001d0 00000000 00000000 00000000 00000000 *................*
0001e0 00000000 00000000 00000000 00000000 *................*
0001f0 00000000 00000000 00000000 000055aa *..............U.*
```

=====

Here is the disassembly of the MBR...

This sector is initially loaded into memory at 0000:7c00 but
it immediately relocates itself to 0000:0600.

```
            BEGIN:                  NOW AT 0000:7C00, RELOCATE

0000:7C00 FA          CLI                 disable int's
0000:7C01 33C0        XOR    AX,AX        set stack seg to 0000
0000:7C03 8ED0        MOV    SS,AX
0000:7C05 BC007C      MOV    SP,7C00      set stack ptr to 7c00
0000:7C08 8BF4        MOV    SI,SP        SI now 7c00
0000:7C0A 50          PUSH   AX
0000:7C0B 07          POP    ES           ES now 0000:7c00
0000:7C0C 50          PUSH   AX
0000:7C0D 1F          POP    DS           DS now 0000:7c00
0000:7C0E FB          STI                 allow int's
0000:7C0F FC          CLD                 clear direction
0000:7C10 BF0006      MOV    DI,0600      DI now 0600
0000:7C13 B90001      MOV    CX,0100      move 256 words (512 bytes)
0000:7C16 F2          REPNZ               move MBR from 0000:7c00
0000:7C17 A5          MOVSW                  to 0000:0600
0000:7C18 EA1D060000  JMP    0000:061D    jmp to NEW_LOCATION

      NEW_LOCATION:                 NOW AT 0000:0600

0000:061D BEBE07      MOV    SI,07BE      point to first table entry
0000:0620 B304        MOV    BL,04        there are 4 table entries

      SEARCH_LOOP1:                 SEARCH FOR AN ACTIVE ENTRY
```

```
0000:0622 803C80     CMP    BYTE PTR [SI],80  is this the active entry?
0000:0625 740E       JZ     FOUND_ACTIVE      yes
0000:0627 803C00     CMP    BYTE PTR [SI],00  is this an inactive entry?
0000:062A 751C       JNZ    NOT_ACTIVE        no
0000:062C 83C610     ADD    SI,+10            incr table ptr by 16
0000:062F FECB       DEC    BL                decr count
0000:0631 75EF       JNZ    SEARCH_LOOP1      jmp if not end of table
0000:0633 CD18       INT    18                GO TO ROM BASIC

      FOUND_ACTIVE:                  FOUND THE ACTIVE ENTRY

0000:0635 8B14       MOV    DX,[SI]           set DH/DL for INT 13 call
0000:0637 8B4C02     MOV    CX,[SI+02]        set CH/CL for INT 13 call
0000:063A 8BEE       MOV    BP,SI             save table ptr

      SEARCH_LOOP2:                  MAKE SURE ONLY ONE ACTIVE ENTRY

0000:063C 83C610     ADD    SI,+10            incr table ptr by 16
0000:063F FECB       DEC    BL                decr count
0000:0641 741A       JZ     READ_BOOT         jmp if end of table
0000:0643 803C00     CMP    BYTE PTR [SI],00  is this an inactive entry?
0000:0646 74F4       JZ     SEARCH_LOOP2      yes

       NOT_ACTIVE:                   MORE THAN ONE ACTIVE ENTRY FOUND

0000:0648 BE8B06     MOV    SI,068B           display "Invld prttn tbl"

      DISPLAY_MSG:                   DISPLAY MESSAGE LOOP

0000:064B AC         LODSB                    get char of message
0000:064C 3C00       CMP    AL,00             end of message
0000:064E 740B       JZ     HANG              yes
0000:0650 56         PUSH   SI                save SI
0000:0651 BB0700     MOV    BX,0007           screen attributes
0000:0654 B40E       MOV    AH,0E             output 1 char of message
0000:0656 CD10       INT    10                to the display
0000:0658 5E         POP    SI                restore SI
0000:0659 EBF0       JMP    DISPLAY_MSG       do it again

           HANG:                     HANG THE SYSTEM LOOP

0000:065B EBFE       JMP    HANG              sit and stay!

       READ_BOOT:                    READ ACTIVE PARITION BOOT RECORD

0000:065D BF0500     MOV    DI,0005           INT 13 retry count

       INT13RTRY:                    INT 13 RETRY LOOP
```

```
0000:0660 BB007C      MOV     BX,7C00
0000:0663 B80102      MOV     AX,0201          read 1 sector
0000:0666 57          PUSH    DI               save DI
0000:0667 CD13        INT     13               read sector into 0000:7c00
0000:0669 5F          POP     DI               restore DI
0000:066A 730C        JNB     INT13OK          jmp if no INT 13
0000:066C 33C0        XOR     AX,AX            call INT 13 and
0000:066E CD13        INT     13               do disk reset
0000:0670 4F          DEC     DI               decr DI
0000:0671 75ED        JNZ     INT13RTRY        if not zero, try again
0000:0673 BEA306      MOV     SI,06A3          display "Errr ldng systm"
0000:0676 EBD3        JMP     DISPLAY_MSG      jmp to display loop

          INT13OK:                    INT 13 ERROR

0000:0678 BEC206      MOV     SI,06C2          "missing op sys"
0000:067B BFFE7D      MOV     DI,7DFE          point to signature
0000:067E 813D55AA    CMP     WORD PTR [DI],AA55   is signature correct?
0000:0682 75C7        JNZ     DISPLAY_MSG      no
0000:0684 8BF5        MOV     SI,BP            set SI
0000:0686 EA007C0000  JMP     0000:7C00        JUMP TO THE BOOT SECTOR
                                               WITH SI POINTING TO
                                               PART TABLE ENTRY
```

Messages here.

```
0000:0680 ........ ........ ......49 6e76616c *           Inval*
0000:0690 69642070 61727469 74696f6e 20746162 *id partition tab*
0000:06a0 6c650045 72726f72 206c6f61 64696e67 *le.Error loading*
0000:06b0 206f7065 72617469 6e672073 79737465 * operating syste*
0000:06c0 6d004d69 7373696e 67206f70 65726174 *m.Missing operat*
0000:06d0 696e6720 73797374 656d00.. ........ *ing system.      *
```

Data not used.

```
0000:06d0 ........ ........ ......00 00000000 *           .....*
0000:06e0 00000000 00000000 00000000 00000000 *................*
0000:06f0 00000000 00000000 00000000 00000000 *................*
0000:0700 00000000 00000000 00000000 00000000 *................*
0000:0710 00000000 00000000 00000000 00000000 *................*
0000:0720 00000000 00000000 00000000 00000000 *................*
0000:0730 00000000 00000000 00000000 00000000 *................*
0000:0740 00000000 00000000 00000000 00000000 *................*
0000:0750 00000000 00000000 00000000 00000000 *................*
0000:0760 00000000 00000000 00000000 00000000 *................*
0000:0770 00000000 00000000 00000000 00000000 *................*
0000:0780 00000000 00000000 00000000 00000000 *................*
0000:0790 00000000 00000000 00000000 00000000 *................*
0000:07a0 00000000 00000000 00000000 00000000 *................*
0000:07b0 00000000 00000000 00000000 0000.... *............    *
```

The partition table starts at 0000:07be. Each partition table entry is 16 bytes. This table defines a single primary partition which is also an active (bootable) partition.

```
0000:07b0 ........ ........ ........ ....8001 *           ....*
0000:07c0 0100060d fef83e00 00000678 0d000000 *......>....x....*
0000:07d0 00000000 00000000 00000000 00000000 *................*
0000:07e0 00000000 00000000 00000000 00000000 *................*
0000:07f0 00000000 00000000 00000000 0000.... *............    *
```

The last two bytes contain a 55AAH signature.

```
0000:07f0 ........ ........ ........ ....55aa *..............U.*
```

/end/

How It Works -- OS2 Boot Sector

Version 1a

by Hale Landis (landis@sugs.tware.com)

THE "HOW IT WORKS" SERIES

This is one of several How It Works documents. The series currently includes the following:

* How It Works -- CHS Translation
* How It Works -- Master Boot Record
* How It Works -- DOS Floppy Boot Sector
* How It Works -- OS2 Boot Sector
* How It Works -- Partition Tables

OS2 BOOT SECTOR

Note: I'll leave it to someone else to provide you with a disassembly of an OS/2 HPFS boot sector, or a Linux boot sector, or a WinNT boot sector, etc.

This article is a disassembly of a floppy or hard disk boot sector for OS/2. Apparently OS/2 uses the same boot sector for both environments. Basically a bootable FAT hard disk partition looks like a big floppy during the early stages of the system's boot processing. This sector is at cylinder 0, head 0, sector 1 of a floppy or it is the first sector within a FAT hard disk partition. OS/2 floppy disk and hard disk boot sectors are created by the OS/2 FORMAT program.

At the completion of your system's Power On Self Test (POST), INT 19 is called. Usually INT 19 tries to read a boot sector from the first floppy drive. If a boot sector is found on the floppy disk, the that boot sector is read into memory at location 0000:7C00 and INT 19 jumps to memory location 0000:7C00. However, if no boot sector is found on the first floppy drive, INT 19 tries to read the MBR from the first hard drive. If an MBR is found it is read into memory at location 0000:7c00 and INT 19 jumps to memory location 0000:7c00. The small program in the MBR will attempt to locate an active (bootable) partition in its partition table. If such a partition is found, the boot sector of that partition is read into memory at location 0000:7C00 and the MBR program jumps to memory location 0000:7C00. Each operating system has its own boot sector format. The small program in the boot sector must locate the first part of the operating system's kernel loader program (or perhaps the kernel

itself or perhaps a "boot manager program") and read that into memory.

INT 19 is also called when the CTRL-ALT-DEL keys are used. On most systems, CTRL-ALT-DEL causes an short version of the POST to be executed before INT 19 is called.

=====

Where stuff is:

The BIOS Parameter Block (BPB) starts at offset 0.
The boot sector program starts at offset 46.
The messages issued by this program start at offset 198.
The OS/2 boot loader file name starts at offset 1d5.
The boot sector signature is at offset 1fe.

Here is a summary of what this thing does:

1) If booting from a hard disk partition, skip to step 6.
2) Copy Diskette Parameter Table which is pointed to by INT 1E to the top of memory.
3) Alter the copy of the Diskette Parameter Table.
4) Alter INT 1E to point to altered Diskette Parameter Table at the top of memory.
5) Do INT 13 AH=00, disk reset call so that the altered Diskette Parameter Table is used.
6) Compute sector address of the root directory.
7) Read the entire root directory into memory starting at location 1000:0000.
8) Search the root directory entires for the file OS2BOOT.
9) Read the OS2BOOT file into memory at 0800:0000.
10) Do a far return to enter the OS2BOOT program at 0800:0000.

NOTES:

This program uses the CHS based INT 13H AH=02 to read the FAT root directory and to read the OS2BOOT file. If the drive is >528MB, this CHS must be a translated CHS (or L-CHS, see my BIOS TYPES document). Except for internal computations no addresses in LBA form are used, another reason why LBA doesn't solve the >528MB problem.

=====

Here is the entire sector in hex and ascii.

```
OFFSET 0 1 2 3  4 5 6 7  8 9 A B  C D E F  *0123456789ABCDEF*
000000 eb449049 424d2032 302e3000 02100100 *.D.IBM 20.0.....*
000010 02000200 00f8d800 3e000e00 3e000000 *........>...>...*
```

```
000020 06780d00 80002900 1c0c234e 4f204e41 *.x....)...#NO NA*
000030 4d452020 20204641 54202020 20200000 *ME    FAT     ..*
000040 00100000 0000fa33 db8ed3bc ff7bfbba *.......3.....{..*
000050 c0078eda 803e2400 00753d1e b840008e *.....>$..u=..@..*
000060 c026ff0e 1300cd12 c1e0068e c033ff33 *.&...........3.3*
000070 c08ed8c5 367800fc b90b00f3 a41fa118 *....6x.........*
000080 0026a204 001e33c0 8ed8a378 008c067a *.&....3....x...z*
000090 001f8a16 2400cd13 a0100098 f7261600 *....$........&..*
0000a0 03060e00 5091b820 00f72611 008b1e0b *....P.. ..&.....*
0000b0 0003c348 f7f35003 c1a33e00 b800108e *...H..P...>.....*
0000c0 c033ff59 890e4400 58a34200 33d2e873 *.3.Y..D.X.B.3..s*
0000d0 0033db8b 0e11008b fb51b90b 00bed501 *.3.......Q......*
0000e0 f3a65974 0583c320 e2ede335 268b471c *..Yt... ...5&.G.*
0000f0 268b571e f7360b00 fec08ac8 268b571a *&.W..6......&.W.*
000100 4a4aa00d 0032e4f7 e203063e 0083d200 *JJ...2.....>....*
000110 bb00088e c333ff06 57e82800 8d360b00 *.....3..W.(..6..*
000120 cbbe9801 eb03bead 01e80900 bec201e8 *................*
000130 0300fbeb feac0ac0 7409b40e bb0700cd *........t.......*
000140 10ebf2c3 50525103 061c0013 161e00f7 *....PRQ.........*
000150 361800fe c28ada33 d2f7361a 008afa8b *6......3..6.....*
000160 d0a11800 2ac34050 b402b106 d2e60af3 *....*.@P........*
000170 8bca86e9 8a162400 8af78bdf cd1372a6 *......$.......r.*
000180 5b598bc3 f7260b00 03f85a58 03c383d2 *[Y...&....ZX....*
000190 002acb7f afc31200 4f532f32 20212120 *.*......OS/2 !! *
0001a0 53595330 31343735 0d0a0012 004f532f *SYS01475.....OS/*
0001b0 32202121 20535953 30323032 350d0a00 *2 !! SYS02025...*
0001c0 12004f53 2f322021 21205359 53303230 *..OS/2 !! SYS020*
0001d0 32370d0a 004f5332 424f4f54 20202020 *27...OS2BOOT    *
0001e0 00000000 00000000 00000000 00000000 *................*
0001f0 00000000 00000000 00000000 000055aa *..............U.*
```

=====

The first 62 bytes of a boot sector are known as the BIOS Parameter Block (BPB). Here is the layout of the BPB fields and the values they are assigned in this boot sector:

```
db JMP instruction      at 7c00 size  2 = eb44
db NOP instruction        7c02        1   90
db OEMname                7c03        8   'IBM 20.0'
dw bytesPerSector         7c0b        2   0200
db sectPerCluster         7c0d        1   01
dw reservedSectors        7c0e        2   0001
db numFAT                 7c10        1   02
dw numRootDirEntries      7c11        2   0200
dw numSectors             7c13        2   0000 (use numSectorsHuge)
db mediaType              7c15        1   f8
dw numFATsectors          7c16        2   00d8
dw sectorsPerTrack        7c18        2   003e
dw numHeads               7c1a        2   000e
```

```
  dd numHiddenSectors       7c1c      4   00000000
  dd numSectorsHuge         7c20      4   000d7806
  db driveNum               7c24      1   80
  db reserved               7c25      1   00
  db signature              7c26      1   29
  dd volumeID               7c27      4   001c0c23
  db volumeLabel            7c2b      11  'NO NAME   '
  db fileSysType            7c36      8   'FAT     '
```

=====

Here is the boot sector...

The first 3 bytes of the BPB are JMP and NOP instructions.

```
0000:7C00 EB44       JMP     START
0000:7C02 90         NOP
```

Here is the rest of the BPB.

```
0000:7C00 eb449049 424d2032 302e3000 02100100 *.D.IBM 20.0.....*
0000:7C10 02000200 00f8d800 3e000e00 3e000000 *........>...>...*
0000:7C20 06780d00 80002900 1c0c234e 4f204e41 *.x....)...#NO NA*
0000:7C30 4d452020 20204641 54202020 20200000 *ME    FAT     ..*
```

Additional data areas.

```
0000:7C30 ........ ........ ........ ....0000 *             ..*
0000:7C40 00100000 0000.... ........ ........ *......         *
```

Note:

0000:7c3e (DS:003e) = number of sectors in the FATs and root dir.
0000:7c42 (DS:0042) = number of sectors in the FAT.
0000:7c44 (DS:0044) = number of sectors in the root dir.

START: START OF BOOT SECTOR PROGRAM

```
0000:7C46 FA         CLI                    interrupts off
0000:7C47 33DB       XOR     BX,BX          zero BX
0000:7C49 8ED3       MOV     SS,BX          SS now zero
0000:7C4B BCFF7B     MOV     SP,7BFF        SP now 7bff
0000:7C4E FB         STI                    interrupts on
0000:7C4F BAC007     MOV     DX,07C0        set DX to
0000:7C52 8EDA       MOV     DS,DX          07c0
```

Are we booting from a floppy or a
hard disk partition?

```
0000:7C54 803E240000  CMP     BYTE PTR [0024],00   is driveNum in BPB 00?
```

```
0000:7C59 753D       JNZ    NOT_FLOPPY          jmp if not zero
```

We are booting from a floppy. The
Diskette Parameter Table must be
copied and altered...

Diskette Parameter Table is pointed to by INT 1E. This program moves this table to high memory, alters the table, and changes INT 1E to point to the altered table.

This table contains the following data:

????:0000 = Step rate and head unload time.
????:0001 = Head load time and DMA mode flag.
????:0002 = Delay for motor turn off.
????:0003 = Bytes per sector.
????:0004 = Sectors per track.
????:0005 = Intersector gap length.
????:0006 = Data length.
????:0007 = Intersector gap length during format.
????:0008 = Format byte value.
????:0009 = Head settling time.
????:000a = Delay until motor at normal speed.

Compute a valid high memory address.

```
0000:7C5B 1E         PUSH   DS                  save DS
0000:7C5C B84000     MOV    AX,0040             set ES
0000:7C5F 8EC0       MOV    ES,AX               to 0040 (BIOS data area)
0000:7C61 26         ES:                        reduce system memory
0000:7C62 FF0E1300   DEC    WORD PTR [0013]     size by 1024
0000:7C66 CD12       INT    12                  get system memory size
0000:7C68 C1E06      SHL    AX,06               shift AX (mult by 64)
0000:7C6B 8EC0       MOV    ES,AX               move to ES
0000:7C6D 33FF       XOR    DI,DI               zero DI
```

Move the diskette param table to high memory.

```
0000:7C6F 33C0       XOR    AX,AX               zero AX
0000:7C71 8ED8       MOV    DS,AX               DS now zero
0000:7C73 C5367800   LDS    SI,[0078]           DS:SI = INT 1E vector
0000:7C77 FC         CLD                        clear direction
0000:7C78 B90B00     MOV    CX,000B             count is 11
0000:7C7B F3         REPZ                       copy diskette param table
0000:7C7C A4         MOVSB                      to top of memory
```

Alter the number of sectors per track
in the diskette param table in high memory.

```
0000:7C7D 1F         POP    DS                  restore DS
```

```
0000:7C7E A11800      MOV     AX,[0018]          get sectorsPerTrack from BPB
0000:7C81 26          ES:                        alter sectors per track
0000:7C82 A20400      MOV     [0004],AL          in diskette param table

                      Change INT 1E to point to altered diskette
                      param table and do a INT 13 disk reset call.

0000:7C85 1E          PUSH    DS                 save DS
0000:7C86 33C0        XOR     AX,AX              AX now zero
0000:7C88 8ED8        MOV     DS,AX              DS no zero
0000:7C8A A37800      MOV     [0078],AX          alter INT 1E vector
0000:7C8D 8C067A00    MOV     [007A],ES          to point to altered
                                                 diskette param table
0000:7C91 1F          POP     DS                 restore DS
0000:7C92 8A162400    MOV     DL,[0024]          driveNum from BPB
0000:7C96 CD13        INT     13                 diskette reset

      NOT_FLOPPY:

                      Compute the location and the size of
                      the root directory.  Read the entire
                      root directory into memory.

0000:7C98 A01000      MOV     AL,[0010]          get numFAT
0000:7C9B 98          CBW                        make into a word
0000:7C9C F7261600    MUL     WORD PTR [0016]    mult by numFatSectors
0000:7CA0 03060E00    ADD     AX,[000E]          add reservedSectors
0000:7CA4 50          PUSH    AX                 save
0000:7CA5 91          XCHG    CX,AX              move to CX
0000:7CA6 B82000      MOV     AX,0020            dir entry size
0000:7CA9 F7261100    MUL     WORD PTR [0011]    mult by numRootDirEntries
0000:7CAD 8B1E0B00    MOV     BX,[000B]          get bytesPerSector
0000:7CB1 03C3        ADD     AX,BX              add
0000:7CB3 48          DEC     AX                 subtract 1
0000:7CB4 F7F3        DIV     BX                 div by bytesPerSector
0000:7CB6 50          PUSH    AX                 save number of dir sectors
0000:7CB7 03C1        ADD     AX,CX              add number of fat sectors
0000:7CB9 A33E00      MOV     [003E],AX          save
0000:7CBC B80010      MOV     AX,1000            AX is now 1000
0000:7CBF 8EC0        MOV     ES,AX              ES is now 1000
0000:7CC1 33FF        XOR     DI,DI              DI is now zero
0000:7CC3 59          POP     CX                 get number dir sectors
0000:7CC4 890E4400    MOV     [0044],CX          save
0000:7CC8 58          POP     AX                 get number fat sectors
0000:7CC9 A34200      MOV     [0042],AX          save
0000:7CCC 33D2        XOR     DX,DX              DX now zero
0000:7CCE E87300      CALL    READ_SECTOR        read 1st sect of root dir
0000:7CD1 33DB        XOR     BX,BX              BX is now zero
0000:7CD3 8B0E1100    MOV     CX,[0011]          number of root dir entries
```

```
DIR_SEARCH:                 SEARCH FOR OS2BOOT.

                Search the root directory for the file
                name OS2BOOT.

0000:7CD7 8BFB      MOV   DI,BX           DI is dir entry addr
0000:7CD9 51        PUSH  CX             save CX
0000:7CDA B90B00    MOV   CX,000B        count is 11
0000:7CDD BED501    MOV   SI,01D5        addr of "OS2BOOT"
0000:7CE0 F3        REPZ                 is 1st dir entry
0000:7CE1 A6        CMPSB                for "OS2BOOT"?
0000:7CE2 59        POP   CX             restore CX
0000:7CE3 7405      JZ    FOUND_OS2BOOT  jmp if OS2BOOT
0000:7CE5 83C320    ADD   BX,+20         incr to next dir entry
0000:7CE8 E2ED      LOOP  DIR_SEARCH     try again

FOUND_OS2BOOT:              FOUND OS2BOOT.

                OS2BOOT was found.  Get the starting
                cluster number and convert to a sector
                address.  Read OS2BOOT into memory and
                finally do a far return to enter
                the OS2BOOT program.

0000:7CEA E335      JCXZ  FAILED1        JMP if CX zero.
0000:7CEC 26        ES:                  get the szie of
0000:7CED 8B471C    MOV   AX,[BX+1C]     the OS2BOOT file
0000:7CF0 26        ES:                  from the OS2BOOT
0000:7CF1 8B571E    MOV   DX,[BX+1E]     directory entry
0000:7CF4 F7360B00  DIV   WORD PTR [000B]  div by bytesPerSect
0000:7CF8 FEC0      INC   AL             add 1
0000:7CFA 8AC8      MOV   CL,AL          num sectors OS2BOOT
0000:7CFC 26        ES:                  get the starting
0000:7CFD 8B571A    MOV   DX,[BX+1A]     cluster number
0000:7D00 4A        DEC   DX             subtract 1
0000:7D01 4A        DEC   DX             subtract 1
0000:7D02 A00D00    MOV   AL,[000D]      sectorsPerCluster
0000:7D05 32E4      XOR   AH,AH          mutiply
0000:7D07 F7E2      MUL   DX             to get LBA
0000:7D09 03063E00  ADD   AX,[003E]      add number of FAT sectors
0000:7D0D 83D200    ADC   DX,+00         to LBA
0000:7D10 BB0008    MOV   BX,0800        set ES
0000:7D13 8EC3      MOV   ES,BX          to 0800
0000:7D15 33FF      XOR   DI,DI          set ES:DI to entry point
0000:7D17 06        PUSH  ES             address of
0000:7D18 57        PUSH  DI             OS2BOOT
0000:7D19 E82800    CALL  READ_SECTOR    read OS2BOOT into memory
0000:7D1C 8D360B00  LEA   SI,[000B]      set DS:SI
0000:7D20 CB        RETF                 "far return" to OS2BOOT
```

```
        FAILED1:                    OS2BOOT WAS NOT FOUND.

0000:7D21 BE9801     MOV     SI,0198          "SYS01475" message
0000:7D24 EB03       JMP     FAILED3

        FAILED2:                    ERROR FROM INT 13.

0000:7D26 BEAD01     MOV     SI,01AD          "SYS02025" message

        FAILED3:                    OUTPUT ERROR MESSAGES.

0000:7D29 E80900     CALL    MSG_LOOP         display message
0000:7D2C BEC201     MOV     SI,01C2          "SYS02027" message
0000:7D2F E80300     CALL    MSG_LOOP         display message
0000:7D32 FB         STI                      interrupts on

          HANG:                 HANG THE SYSTEM!

0000:7D33 EBFE       JMP     HANG             sit and stay!

        MSG_LOOP:               DISPLAY AN ERROR MESSAGE.

                    Routine to display the message
                    text pointed to by SI.

0000:7D35 AC         LODSB                    get next char of message
0000:7D36 0AC0       OR      AL,AL            end of message?
0000:7D38 7409       JZ      RETURN           jmp if yes
0000:7D3A B40E       MOV     AH,0E            write 1 char
0000:7D3C BB0700     MOV     BX,0007          video attributes
0000:7D3F CD10       INT     10               INT 10 to write 1 char
0000:7D41 EBF2       JMP     MSG_LOOP         do again

          RETURN:

0000:7D43 C3         RET                      return

     READ_SECTOR:               ROUTINE TO READ SECTORS.

                    Read sectors into memory.  Read multiple
                    sectors but don't read across a track
                    boundary.

                    The caller supplies the following:
                      DX:AX = sector address to read (as LBA)
                        CX = number of sectors to read
                      ES:DI = memory address to read into

0000:7D44 50         PUSH    AX               save lower part of LBA
0000:7D45 52         PUSH    DX               save upper part of LBA
```

```
0000:7D46 51          PUSH    CX                  save number of sect to read
0000:7D47 03061C00    ADD     AX,[001C]           add numHiddenSectors
0000:7D4B 13161E00    ADC     DX,[001E]             to LBA
0000:7D4F F7361800    DIV     WORD PTR [0018]     div by sectorsPerTrack
0000:7D53 FEC2        INC     DL                  add 1 to sector number
0000:7D55 8ADA        MOV     BL,DL               save sector number
0000:7D57 33D2        XOR     DX,DX               zero upper part of LBA
0000:7D59 F7361A00    DIV     WORD PTR [001A]     div by numHeads
0000:7D5D 8AFA        MOV     BH,DL               save head number
0000:7D5F 8BD0        MOV     DX,AX               save cylinder number
0000:7D61 A11800      MOV     AX,[0018]           sectorsPerTrack
0000:7D64 2AC3        SUB     AL,BL               sub sector number
0000:7D66 40          INC     AX                  add 1
0000:7D67 50          PUSH    AX                  save number of sector to read
0000:7D68 B402        MOV     AH,02               INT 13 read sectors
0000:7D6A B106        MOV     CL,06               shift count
0000:7D6C D2E6        SHL     DH,CL               shift high cyl left
0000:7D6E 0AF3        OR      DH,BL               or in sector number
0000:7D70 8BCA        MOV     CX,DX               move cyl/sect to CX
0000:7D72 86E9        XCHG    CH,CL               swap cyl/sect
0000:7D74 8A162400    MOV     DL,[0024]           driveNum
0000:7D78 8AF7        MOV     DH,BH               head number
0000:7D7A 8BDF        MOV     BX,DI               memory addr to read into
0000:7D7C CD13        INT     13                  INT 13 read sectors call
0000:7D7E 72A6        JB      FAILED2             jmp if any error
0000:7D80 5B          POP     BX                  get number of sectors read
0000:7D81 59          POP     CX                  restore CX
0000:7D82 8BC3        MOV     AX,BX               number of sector to AX
0000:7D84 F7260B00    MUL     WORD PTR [000B]     multiply by sector size
0000:7D88 03F8        ADD     DI,AX               add to memory address
0000:7D8A 5A          POP     DX                  restore upper part of LBA
0000:7D8B 58          POP     AX                  resotre lower part of LBA
0000:7D8C 03C3        ADD     AX,BX               add number of sector just
0000:7D8E 83D200      ADC     DX,+00                read to LBA
0000:7D91 2ACB        SUB     CL,BL               decr requested num of sect
0000:7D93 7FAF        JG      READ_SECTOR         jmp if not zero
0000:7D95 C3          RET                         return
```

Data not used.

```
0000:7D90 ........ ....1200 ........ ........ *       ..        *
```

Messages here.

```
0000:7D90 ........ ........ 4f532f32 20212120 *        OS/2 !! *
0000:7Da0 53595330 31343735 0d0a0012 004f532f *SYS01475.....OS/*
0000:7Db0 32202121 20535953 30323032 350d0a00 *2 !! SYS02025...*
0000:7Dc0 12004f53 2f322021 21205359 53303230 *..OS/2 !! SYS020*
0000:7Dd0 32370d0a 00...... ........ ........ *27...           *
```

OS/2 loader file name.

0000:7Dd0 ....4f5332 424f4f54 20202020 * OS2BOOT *

Data not used.

0000:7De0 00000000 00000000 00000000 00000000 *.....*
0000:7Df0 00000000 00000000 00000000 0000.... *..... *

The last two bytes contain a 55AAH signature.

0000:7Df0 ....55aa * U.*

/end/

How it Works -- Partition Tables
Part 1 of 2

Version 1e

by Hale Landis <landis@sugs.tware.com>

THE "HOW IT WORKS" SERIES

This is one of several How It Works documents. The series currently includes the following:

* How It Works -- CHS Translation
* How It Works -- Master Boot Record
* How It Works -- DOS Floppy Boot Sector
* How It Works -- OS2 Boot Sector
* How It Works -- Partition Tables

NOTE: A "!" at the left margin of the first line of a paragraph indicates that some change was made in that paragraph.

PARTITION SECTOR/RECORD/TABLE BASICS

FDISK creates all partition records (sectors). The primary purpose of a partition record is to hold a partition table. The rules for how FDISK works are unwritten but so far most FDISK programs (DOS, OS/2, WinNT, etc) seem to follow the same basic idea.

First, all partition table records (sectors) have the same format. This includes the partition table record at cylinder 0, head 0, sector 1 -- what is known as the Master Boot Record (MBR). The last 66 bytes of a partition table record contain a partition table and a 2 byte signature. The first 446 bytes of these sectors usually contain a program but only the program in the MBR is ever executed (so extended partition table records could contain something other than a program in the first 466 bytes). See "How It Works -- The Master Boot Record".

Second, extended partitions are "nested" inside one another and extended partition table records form a "linked list". I will attempt to show this in a diagram below.

PARTITION TABLE ENTRY FORMAT

Each partition table entry is 16 bytes and contains things like the start and end location of a partition in CHS, the start in

LBA, the size in sectors, the partition "type" and the "active" flag. Warning: older versions of FDISK may compute incorrect LBA or size values. And note: When your computer boots itself, only the CHS fields of the partition table entries are used (another reason LBA doesn't solve the >528MB problem). The CHS fields in the partition tables are in L-CHS format -- see "How It Works -- CHS Translation".

! There is no central clearing house to assign the codes used in the one byte "type" field, however, there is at least one person at both Microsoft and IBM that attempt to keep track of the type codes. Type codes are used to define most every type of file system that anyone has ever implemented on the x86 PC: 12-bit FAT, 16-bit FAT, HPFS, NTFS, etc. Plus, an extended partition also has a unique type code. The complete list of known partition type codes is contained in part 2 of this document.

The 16 bytes of a partition table entry are used as follows:

```
   +--- Bit 7 is the active partition flag, bits 6-0 are zero
   |        (when not zero this byte is also the drive
   |        number of the drive to boot so the active
   |        partition is always found on drive 80H,
   |        the first hard disk).
   |
   |   +--- Starting CHS in INT 13 call format.
   |   |
   |   |     +--- Partition type byte.
   |   |     |
   |   |     |   +--- Ending CHS in INT 13 call format.
   |   |     |   |
   |   |     |   |       +-- Starting LBA.
   |   |     |   |       |
   |   |     |   |       |      +-- Size in sectors.
   |   |     |   |       |      |
   v <--+---> v <--+-->    v      v

   0  1  2  3  4  5  6  7  8 9 A B  C D E F
   DL DH CL CH TB DH CL CH LBA..... SIZE....

   80 01 01 00 06 0e be 94 3e000000 0c610900  1st entry

   00 00 81 95 05 0e fe 7d 4a610900 724e0300  2nd entry

   00 00 00 00 00 00 00 00 00000000 00000000  3rd entry

   00 00 00 00 00 00 00 00 00000000 00000000  4th entry
```

Bytes 0-3 are used by the small program in the Master Boot Record to read the first sector of an active partition into memory. The

DH, DL, CH and CL above show which x86 register is loaded when the MBR program calls INT 13H AH=02H to read the active partition's boot sector. See "How It Works -- Master Boot Record".

These entries define the following partitions:

1) The first partition, a primary partition DOS FAT, starts at CHS 0H,1H,1H (LBA 3EH) and ends at CHS 294H,EH,3EH with a size of 9610CH sectors.

2) The second partition, an extended partition, starts at CHS 295H,0H,1H (LBA 9614AH) and ends at CHS 37DH,EH,3EH with a size of 34E72H sectors.

3) The third and fourth table entries are unused.

PARTITION TABLE RULES

Keep in mind that there are NO written rules and NO industry standards on how FDISK should work but here are some basic rules that seem to be followed by most versions of FDISK:

1) In the MBR there can be 0-4 "primary" partitions, OR, 0-3 primary partitions and 0-1 extended partition entry.

2) In an extended partition there can be 0-1 "secondary" partition entries and 0-1 extended partition entries.

3) Only 1 primary partition in the MBR can be marked "active" at any given time.

4) In most versions of FDISK, the first sector of a partition will be aligned such that it is at head 0, sector 1 of a cylinder. This means that there may be unused sectors on the track(s) prior to the first sector of a partition and that there may be unused sectors following a partition table sector.

   For example, most new versions of FDISK start the first partition (primary or extended) at cylinder 0, head 1, sector 1. This leaves the sectors at cylinder 0, head 0, sectors 2...n as unused sectors. This same layout may be seen on the first track of an extended partition. See example 2 below.

   Also note that software drivers like Ontrack's Disk Manager depend on these unused sectors because these drivers will "hide" their code there (in cylinder 0, head 0, sectors 2...n). This is also a good place for boot sector virus programs to hang out.

5) The partition table entries (slots) can be used in any order. Some versions of FDISK fill the table from the bottom up and some versions of FDISK fill the table from the top down. Deleting a partition can leave an unused entry (slot) in the middle of a table.

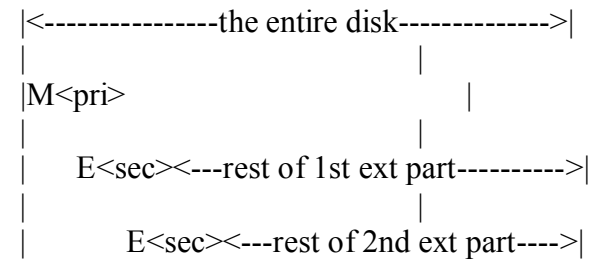
6) And then there is the "hack" that some newer OS's (OS/2 and Linux) use in order to place a partition spanning or passed cylinder 1024 on a system that does not have a CHS translating BIOS. These systems create a partition table entry with the partition's starting and ending CHS information set to all FFH. The starting and ending LBA information is used to describe the location of the partition. The LBA can be converted back to a CHS -- most likely a CHS with more than 1024 cylinders. Since such a CHS can't be used by the system BIOS, these partitions can not be booted or accessed until the OS's kernel and hard disk device drivers are loaded. It is not known if the systems using this "hack" follow the same rules for the creation of these type of partitions.

There are NO written rules as to how an OS scans the partition table entries so each OS can have a different method. For DOS, this means that different versions could assign different drive letters to the same FAT file system partitions.

PARTITION NESTING

What do I mean when I say the partitions are "nested" within each other? Lets look at this example:

M = Master Boot Record (and any unused sectors on the same track)
E = Extended partition record (and any unused sectors on the same track)
pri = a primary partition (first sector is a "boot" sector)
sec = a secondary partition (first sector is a "boot" sector)



The first extended partition is described in the MBR and it occupies the entire disk following the primary partition. The

second extended partition is described in the first extended partition record and it occupies the entire disk following the first secondary partition.

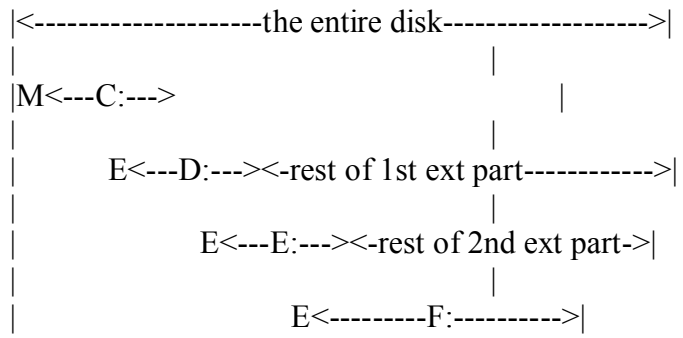
PARTITION TABLE LINKING

What do I mean when I say the partition records (tables) form a "linked" list? This means that the MBR has an entry that describes (points to) the first extended partition, the first extended partition table has an entry that describes (points to) the second extended partition table, and so on. There is, in theory, no limited to out long this linked list is. When you ask FDISK to show the DOS "logical drives" it scans the linked list looking for all of the DOS FAT type partitions that may exist. Remember that in an extended partition table, only two entries of the four can be used (rule 2 above).

And one more thing... Within a partition, the layout of the file system data varies greatly. However, the first sector of a partition is expected to be a "boot" sector. A DOS FAT file system has: a boot sector, first FAT sectors, second FAT sectors, root directory sectors and finally the file data area. See "How It Works -- OS2 Boot Sector".

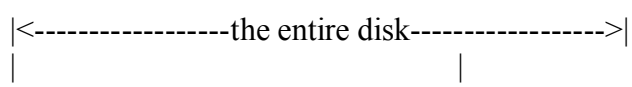
EXAMPLE 1

A disk containing four DOS FAT partitions (C, D, E and F):



EXAMPLE 2

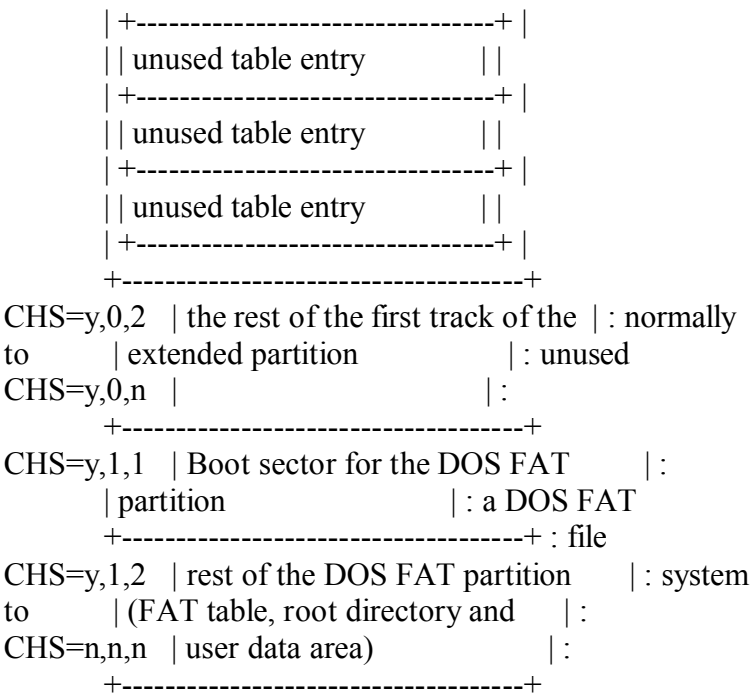
So here is an example of a disk with two primary partitions, one DOS FAT and one OS/2 HPFS, plus an extended partition with another DOS FAT:



```
|M<pri 1 - DOS FAT>                          |
|                                  |
|              <pri 2 - OS/2 HPFS>            |
|                                  |
|                          E<sec - DOS FAT>|
```

Or in more detail ('n' is the highest cylinder, head or sector number number allowed in the indicated field of the CHS)...

```
          +-----------------------------------+
 CHS=0,0,1  | Master Boot Record containing     |
          | partition table search program and |
          | a partition table                 |
          | +-------------------------------+ |
          | | DOS FAT partition description | | points to CHS=0,1,1
          | +-------------------------------+ | points to CHS=a
          | | OS/2 HPFS partition description | |
          | +-------------------------------+ |
          | | unused table entry            | |
          | +-------------------------------+ |
          | | extended partition entry      | | points to CHS=b
          | +-------------------------------+ |
          +-----------------------------------+
CHS=0,0,2   | the rest of "track 0" -- this is  | :
to          | where the software drivers such as | : normally
CHS=0,0,n   | Ontrack's Disk Manager or Micro   | : unused
          | House's EZ Drive are located.     | :
          +-----------------------------------+
CHS=0,1,1   | Boot sector for the DOS FAT       | :
          | partition                         | : a DOS FAT
          +-----------------------------------+ : file
CHS=0,1,2   | rest of the DOS FAT partition     | : system
to          | (FAT table, root directory and    | :
CHS=x-1,n,n | user data area)                   | :
          +-----------------------------------+
CHS=x,0,1   | Boot sector for the OS/2 HPFS     | :
          | file system partition             | : an OS/2
          +-----------------------------------+ : HPFS file
CHS=x,0,2   | rest of the OS/2 HPFS file system | : system
to          | partition                         | :
CHS=y-1,n,n |                                   | :
          +-----------------------------------+
CHS=y,0,1   | Partition record for the extended |
          | partition containing a partition  |
          | record program (never executed) and |
          | a partition table                 |
          | +-------------------------------+ |
          | | DOS FAT partition description | | points to CHS=b+1
```



EXAMPLE 3

Here is a partition record from an extended partition (the first sector of an extended partition). Note that it contains no program code. It contains only the partition table and the signature data.

```
OFFSET 0 1 2 3  4 5 6 7  8 9 A B  C D E F  *0123456789ABCDEF*
000000 00000000 00000000 00000000 00000000 *................*
000010 TO 0001af SAME AS ABOVE
0001b0 00000000 00000000 00000000 00000001 *................*
0001c0 8195060e fe7d3e00 0000344e 03000000 *.....}>...4N....*
0001d0 00000000 00000000 00000000 00000000 *................*
0001e0 00000000 00000000 00000000 00000000 *................*
0001f0 00000000 00000000 00000000 000055aa *..............U.*
```

NOTES

Thanks to yue@heron.Stanford.EDU (Kenneth C. Yue) for pointing out that in V0 of this document I did not properly describe the unused sectors normally found around the partition table sectors.

Thanks to Marcus.Better@abc.se (Marcus Better) for pointing out that in V1a-c of this document I did not properly describe the x86 registers that the partition table entry data is loaded into when INT 13 is called.

/end part 1 of 2/

How it Works -- Partition Tables
Part 2 of 2

Version 1g

by Hale Landis <landis@sugs.tware.com>

PARTITION TYPE CODES

The following table of partition type codes was compiled from many sources including information from kind people at IBM and Microsoft plus Ralf Brown's <ralf@telerama.lm.com> list.

Both IBM and Microsoft keep the partition type code lists and both have until recently assigned type codes. However, there apparently is no formal agreement between these two companies and their lists are not always in sync or up to date. It should be noted that the lists I obtained from IBM and Microsoft are fairly short and show most of type type codes as "available" or "reserved". My guess is that many type codes have been used without the knowledge of the IBM or Microsoft. This is probably the case since neither IBM or Microsoft have published a phone number or email address to contact if you wanted to request a type code. Chaos is the keyword here.

It now appears that neither company is assigning new type codes. I have also been told that it is now recommended that anyone defining a new partition type, or more correctly stated, anyone defining a new file system type, should use partition type 07 and use the first block(s) of the partition to fully define the file system type. Of course, the appearance of a new type code could cause problems for older versions of FDISK, various older operating system device drivers and disk utility programs. (This new use of type code 07 doesn't explain the recent "assignment" of type codes 0E and 0F.)

[If you know of a type code assignment that is not listed here please let me know about it ASAP. Thanks, Hale]

Note that several type codes have multiple uses (for example, see code 08). Also note that there is some question about the use of some codes as denoted by a '?' in the description.

Code	Description
00	Unused partition table entry
01	DOS, 12-bit FAT
02	XENIX root

03 XENIX user
04 DOS, 16-bit FAT
05 Extended partition (includes other partition types)
06 DOS, >32MB support, up to 64K Allocation unit
07 See partition boot record(s) for file system type:
could be QNX, OS/2 HPFS, Windows NT NTFS, Unix, ...
08 OS/2 (thru Version 1.3 only)
08 DELL partition spanning multiple drives (array)
08 Commodore DOS
08 AIX boot? or file system?
09 AIX boot? or file system?
09 Coherent swap
0A OS/2 Boot Manager
0A OPUS
0A Coherent swap
0B FAT32
0C FAT32 LBA (SEE NOTE BELOW)
0D ? (perhaps a type 07 LBA, SEE NOTE BELOW)
0E FAT16 LBA (SEE NOTE BELOW)
0F Extended partition LBA (SEE NOTE BELOW)

NOTE: Partitions types 0C, 0E and 0F (perhaps 0D too) REQUIRE that the system's INT 13 BIOS support the IBM/Microsoft/Phoenix extended/enhanced functions calls (AH=4x). In these partition table entries the CHS fields are NOT used and are generally set to maximum values (all 1 bits) in each CHS field. What this means is that some day there will be an LBA type partition for all the other partition types that are listed here and still in use by some system. This will certainly use up many of the currently unused type codes!

10 OPUS
11 OS/2 Boot Manager: Inactive type 01
12 Compaq diagnostics
13 Available for assignment
14 OS/2 Boot Manager: Inactive type 04
14 Novell DOS 7.0 FDISK (result of bug in FDISK?)
15 Available for assignment
16 OS/2 Boot Manager: Inactive type 06
17 OS/2 Boot Manager: Inactive type 07
18 AST Windows swap file
19 - 1F Available for assignment

20 Available for assignment
21 Reserved
22 Available for assignment
23 Reserved
24 NEC version of MS-DOS
25 Available for assignment

26 Reserved
27 - 2F Available for assignment

30 Available for assignment
31 Reserved
32 Available for assignment
33 Reserved
34 Reserved
35 Available for assignment
36 Reserved
37 - 3B Available for assignment
3C PowerQuest PartitionMagic recovery partition
3D - 3F Available for assignment

40 VENIX :Venix 80286
41 Personal RISC Boot
42 Secure File System (Peter Gutmann)
43 - 4F Available for assignment

50 OnTrack Disk Manager (read-only)
51 OnTrack Disk Manager (write-only)
51 Novell
51 OnTrack Disk Manager (read-only)
52 CP/M
52 Microport
53 OnTrack Disk Manager (write-only)
54 OnTrack Disk Manager (DDO)
55 Available for assignment
56 GoldenBow VFeature
57 - 5F Available for assignment

60 Available for assignment
61 SpeedStor
62 Available for assignment
63 UNIX System V/386
63 Mach, MtXinu BSD 4.3 on Mach
63 GNU HURD
64 Speedstore
64 Novell
65 Novell 286 Netware
66 Novell 386 Netware
67 Novell
68 Novell
69 Novell
6A - 6F Available for assignment

70 DiskSecure Multi-Boot
71 Reserved
72 Available for assignment
73 - 74 Reserved

75 PC/IX
76 Reserved
77 - 79 Available for assignment
7A - 7F ? (probably "available for assignment", these codes
are not shown in the IBM or Microsoft lists!)

80 Minix (ver. 1.4a and earlier)
81 Minix (ver. 1.4b and later)
81 Mitac Advanced Disk Manager
81 Linux
82 Prime
82 Linux swap
83 Linux ext2fs
84 OS/2 hiding a type 04
85 Available for assignment
86 Reserved
87 HPFS FT mirrored partition
88 - 8F Available for assignment

90 - 92 Available for assignment
93 Ameba file system
94 Ameba bad block table
95 - 98 Available for assignment
99 Mylex EISA SCSI
9A - 9F Available for assignment

A0 Available for assignment
A1 Reserved
A2 Available for assignment
A3 - A4 Reserved
A5 FreeBSD
A6 Reserved
A7 - AF Available for assignment

B0 Available for assignment
B1 Reserved
B2 Available for assignment
B3 - B4 Reserved
B5 Available for assignment
B6 Reserved
B7 BSDI file system or secondarily swap
B8 BSDI swap or secondarily file system
B9 - BF Available for assignment

C0 Available for assignment
C1 DR-DOS LOGIN.EXE-secured 12-bit FAT
C2 - C3 Available for assignment
C4 DR-DOS LOGIN.EXE-secured 16-bit FAT
C5 Available for assignment
C6 DR-DOS LOGIN.EXE-secured Huge

C7 HPFS FT disabled mirrored partition
C7 Cyrnix Boot
C8 - CF Available for assignment

D0 - D7 Available for assignment
D8 CP/M 86
D9 - DA Available for assignment
DB Concurrent DOS, CP/M and CTOS
DC - DF Available for assignment

E0 Available for assignment
E1 Speedstore
E2 Available for assignment
E3 Storage Dimensions
E4 Speedstore
E5 - E6 Reserved
E7 - EF Available for assignment

F0 Available for assignment
F1 Storage Dimensions
F2 DOS 3.3+ second partition
F3 Reserved
F4 Speedstore
F4 Storage Dimensions
F5 Available for assignment
F6 Reserved
F7 - FD Available for assignment
FE IBM PS/2 IML, LANstep
FF Xenix(?) Bad Block Tables

Note: Thanks to Christian Carey (ccarey@CapAccess.ORG) for
telling me about partition type 99.

/end part 2 of 2/

!!!!!!!!! begin the WHAT'S NEW BULLETIN

Updated 09 Nov 96

Thanks for your interest in the How It Works documents. Here is some late breaking news about these documents, the ATA (IDE/EIDE) disk drive industry and other things...

---
* NEW NEW NEW *
ATADEMO PROGRAM

If you really want to learn how the ATA (IDE/EIDE) and/or ATAPI interfaces work, then you need a copy of this program. This program lets you issue commands to your ATA/ATAPI devices, bypassing all BIOS and OS drivers, and it reports the results of those commands in several levels of detail. If you are having partition sector or partition table problems you may want to check out the SHOWPT command of this program. Watch for new versions of this program that will support the ATAPI Packet command (command code A0H). Follow the installation and operation instructions carefully!

GET HIW BY FTP

You may find it faster to get the How It Works (HIW) documents by FTP...

ftp://fission.dt.wdc.com/pub/otherdocs/pc_systems/how_it_works

Only the ZIP ALL file (ALLHIW.ZIP) is available at this FTP site. (The ATADEMO.ZIP file will be available there soon.)

THE HIW MINI DOCUMENTS:
FACTS and FICTION (FnF)

A new set of mini HIW documents is now available. These documents are called the Facts and Fiction (FnF) series. They are mostly small articles that were written to answer specific questions. Check it out -- request the FnF Volume 1 today: FNF1 (only available as an ASCII file).

---
* UPDATED *
STANDARDS COMMITTEE NEWS

Find out what the ATA and ATAPI standards committees are doing. They don't do much so progress is generally very slow. Request the Standards News today: ATANEWS (only available as an ASCII file).

OK, I have not updated this document for about a year and lots has happened. Watch for a major update with all the latest ATA/ATAPI/etc news!

LOOKING FOR THE EIDE FAQ?

If you are looking for the very good EIDE FAQ, check out these sites:

This FAQ is the work of John Wehman (jwehman@got.net) and Peter Herweijer (pieterh@sci.kun.nl). The most recent version is available by FTP from

- <ftp://ftp.netcom.com/pub/cl/clau/ide_ata/>
- <ftp://ftp.rahul.net/pub/lps/hard-disk/ya-ata.faq>
- <ftp://ftp.wi.leidenuniv.nl/pub/faqs/>
- <ftp://rtfm.mit.edu/pub/usenet/news.answers/pc-hardware-faq/enhanced-IDE/>

and finally on the World Wide Web from

- <http://www.wi.leidenuniv.nl/ata/>

You can also get the absolutely latest version by e-mail from pieterh@sci.kun.nl by sending a message with "EIDE FAQ text" in the Subject: header. The body of the message will be ignored. You can replace "text" by "postscript" or "html" if you want something more fancy than plain text. Anything else will probably break the mail server and cause it to send you an uuencoded coredump :-)

LOOKING FOR BIOS INT 13 DOCUMENTATION?

The most current documentation for a PC BIOS INT 13 services can be found at

http://www.ptltd.com (or ftp://www.ptltd.com)

The document is the "BIOS Enhanced Disk Drive Specification" (EDD). Look for a ZIP file with the name "EDDxxx.ZIP".

This document describes the IBM/Microsoft extensions to INT 13 and the Phoenix enhancements to some of the extended INT 13 functions.

The original IBM PC INT 13 supports functions calls 0x (AH=0xH). The IBM/Microsoft extensions add the 4x (AH=4xH) calls. Phoenix has enhanced a few of the 4x calls to return additional hardware and BIOS configuration data.

Don't be fooled by all the misleading statements you may see that say OS/2 or WinNT or Win95 or Linux doesn't use the BIOS when accessing a hard disk. These statements are false. All PC based OS's must use the BIOS to boot (load the OS kernel and some device drivers) and all OS's must use the BIOS to determine the hardware configuration of the system (including the CHS geometry of the hard disks).

Get this document and learn how all this works including some more information on why LBA is not required to break the 528MB barrier.

Note: The Phoenix EDD document does not describe the older INT 13 0x function calls. Documentation of these can be found in many books in your local computer book store, for example, "Undocumented PC" by Frank van Gilluwe. However, beware that many of these books, including "Undocumented PC", talked about a version of INT 13 that can use 12-bit cylinder numbers. Such BIOS were never widely implemented and were never supported by any mainstream OS.

!!!!!!!!! end the WHAT'S NEW BULLETIN

!!!!!!!!! begin the HIW Help document

HIW Document Server Help
Updated 09 Nov 96

This is the help document for the How It Works document server.
This document is sent as a plain ASCII text file.

HOW TO USE THE HIW DOCUMENT SERVER

The HIW document server will mail back to you one or more of the HIW documents. Follow these simple steps to use this server:

Send an email message to

hiw@sugs.tware.com

with one or more document names, or, one or more server commands, in the message body. Place one document name or server command per line in the message body. The subject line of the message is ignored.

The server will process each line of the message body until one of the following happens:

a) A line with the word END is found.

b) A line with data other than alphnumeric characters is found.

You will receive one or more email messages from the server for each document requested. The server will respond to document requests approximately once every 24 hours.

Additional rules and notes:

a) Document names and server commands can be in upper or lower case.

b) Additional data to the right of a document name or server command is ignored.

c) Blank lines are ignored.

SERVER COMMANDS

The following server commands are available:

Name	Description
ASCII	Send the following documents as plain ASCII text files. This is the default.
END	Should be used at the end of the list of documents and commands to prevent the server from processing signature lines.
HELP	Returns this help document. Help is sent as a plain ASCII text file even if HTML or ZIP is in effect.
HTML	Send the following documents as HTML files (HTML files are also plain ASCII text files).
INDEX	Returns an index of all the documents showing the current version number and approximate size. The index is sent as a plain ASCII text file even if HTML or ZIP is in effect.
ZIP	Send the following documents as uuencode'd ZIP files.

ASCII DOCUMENTS

The following ASCII documents are currently available (use the ASCII server command before requesting any of these documents):

Name	Description
ALL	Sends all of these ASCII documents, except the ATA-2 standard, each in a separate email message.
ATANEWS	News from the ATA and ATAPI committees and other industry information. Updated about once a month depending on what is going on.
BASICS	Basic information about the IDE/EIDE/ATA/ ATA-2/ATAPI interfaces.

CHSTRANS A detailed description of how CHS and LBA addressing is handled by the system BIOS and operating systems.

DOSBOOT A description and disassembly of an DOS boot record (sector).

FNF1 Facts and Fiction Volume 1. A collection of Mini HIW articles.

MBR A description and disassembly of a Master Boot Record (sector).

OS2BOOT A description and disassembly of an OS/2 boot record (sector).

PARTTABLE A description of partition records and the rules an FDISK program must follow when creating this data plus a complete list of all known partition type codes.

WHATSNEW The What's New Bulletin. Also sent anytime the HELP document is requested.

ATA2 The ATA-2 Revision 3 document. This is the *real* IDE/EIDE standard! Sent in six parts.

HTML DOCUMENTS

The following HTML documents are currently available (use the HTML server command before requesting any of these documents):

Name	Description

*** NO DOCUMENTS IN HTML FORMAT ARE AVAILABLE AT THIS TIME ***

ZIP DOCUMENTS

The following ZIP documents are currently available (use the ZIP server command before requesting any of these documents):

Name	Description

ALL uuencode'd ZIP file containing all of the ASCII documents listed above except the ATA-2 standard document. This file (allhiw.zip) is also at the WD FTP site -- see below.

ATA2 uuencode'd ZIP file containing the ATA-2 Revision 3 document. This is the *real* IDE/EIDE standard!

ATADEMO uuencoded'd ZIP file containing the ATADEMO program. If really want to learn how the ATA/ATAPI interfaces work, you need this program (you also need a copy of the ATA-2 document).

EXAMPLES

Here are some examples (only the message body lines are shown):

1) Request this help document.

```
help
END
```

2) Request the HIW Master Boot Record document in ASCII.

```
ascii
MBR
EnD
```

3) Request the HIW Partition Tables, Master Boot Record and DOS Boot Record document.

```
parttable
mbr
DOSboot
eNd
```

4) Request the ZIP file containing all the ASCII HIW documents.

```
ZIP
ALL
```

NOTES and NOTICES
---

1) The file allhiw.zip can be found at:

ftp://fission.dt.wdc.com/pub/otherdocs/pc_systems/how_it_works

2) HTML and ZIP'ed versions of these documents may be available in the future. Other related documents may be may available if there is interest and the authors of those documents give approval.

3) If you have problems or suggestions about this document server, send a message to

Hale Landis -- landis@sugs.tware.com

4) Hale Landis reserves the right to restrict access to this server at any time and for any reason without prior notice.

!!!!!!!!!! end the HIW Help document

.

Reprinted 20 march 2004, Sopot.