14 Diskettes And Hard Drives

The main purpose of the BIOS routines is to perform low-level functions on behalf of DOS. For example, BIOS routines can physically format the surface of a floppy diskette or access the sectors of a hard drive. DOS, however, remains the master controller for these processes. Most applications perform disk drive operations at the DOS level instead of the BIOS level. However, there are exceptions. For example, disk utilities, such as PC Tools or Norton Utilities, access the disk drive at the BIOS level. But generally these specialized programs are rare.

In this chapter we'll show you how to access the disk drives using the BIOS functions. Since all disk controllers aren't programmed identically, we won't be programming the disk controller directly. Most of the functions that you can perform at the disk controller level can also be performed at the BIOS level. It's worth using the BIOS functions to avoid hardware-dependent disk controller programming.

In addition to BIOS functions, we'll also discuss a few topics related to the hard drive. We'll explain the different types of hard drive controllers and see how hard drives record data.

We'll end the chapter with a discussion of hard drive partitioning, which lets the user divide the hard drive into several logical drives.

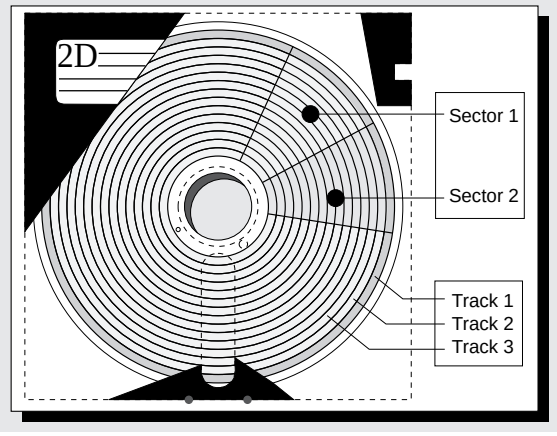
Floppy Diskette And Hard Drive Structure

Let's begin with a look at the common characteristics of floppy diskettes and hard drives. Floppy diskettes and hard drives have similar structures, which is indicated by the number of BIOS functions that apply to both. If you think of a floppy diskette as a two-dimensional version of a hard drive, the similarities are even more apparent.

Floppy diskette structure

A floppy diskette consists of individual tracks arranged as concentric circles at equal intervals over the surface of the diskette's magnetic media. These tracks are labeled from 0 to N; N represents the total number of sectors minus 1 and varies depending on the format. The outermost track is always numbered 0, the next track is numbered 1, etc. This process continues to the innermost track.

Structure of a 5.25-inch diskette



Each track is subdivided into a fixed number of sectors. Each sector holds the same amount of data. Sectors are numbered from 1 to N; N represents the number of sectors per track. The maximum number of sectors in a track depends on the type of floppy disk drive and the diskette's format. Each sector contains 512 bytes and is the smallest amount of data that a program can access. In other words, you must read or write a complete sector at a time. It isn't possible to read or write a single byte from the diskette.

The data in each sector is recorded using either an FM or MFM technique. These are the same recording methods used in hard drives (refer to the "Recording Information On The Hard Drive" section for more information). As a programmer, you don't have to worry about these details to access the data from the disk drive.

Use the following formula to calculate the capacity of a floppy diskette. Remember, this formula is for a single side of a diskette. If the floppy disk drive has two read/write heads (like most recent floppy drives), you must double this value. DOS refers to these sides of a diskette as side 0 and side 1.

```
Sectors * tracks_per_sector * 512 [bytes per sector]
```

The number of sectors per track also affects the data transfer rate. The data transfer rate is the speed of the floppy disk drive electronics and its controller. With a constant rotation speed of 300 revolutions a minute, the more bits per time unit that pass by the read/write head, and the more sectors can be written to a track.

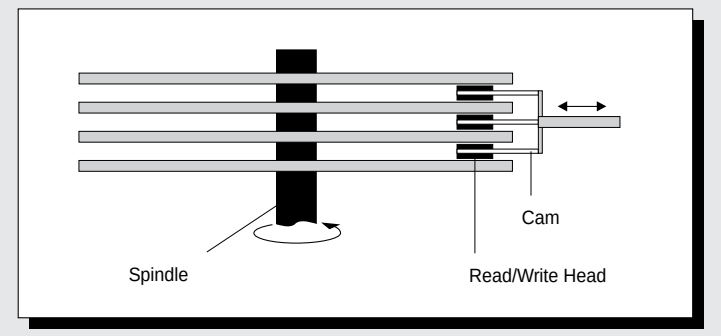
Hard drive structure

Since a hard drive rotates ten times faster than a floppy diskette, its data transfer rate is at least ten times higher than a floppy diskette's. The data transfer rate increases by a second power of ten, because modern 3.5-inch hard drives can store almost 100 sectors per track.

These characteristics don't change the fundamental structure of a hard drive. Think of a hard drive as a group of magnetic plates stacked on top of each other. Each magnetic plate is similar to a floppy diskette; it has two sides, is divided into tracks, and each track is subdivided into sectors. Above the surface of each side of the plate is a read/write head that accesses the data. The plates are aligned so track 0 on one of the plates is exactly above track 0 of another plate.

A read/write arm links all the read/write heads together. To access a particular track on one of the plates, the arm moves all the read/write heads to the specific track. Since this arrangement requires only a single positioning mechanism (the read/write arm), it simplifies the design and lowers the cost of the hard drive. However, with this arrangement, all the read/write heads must be moved to access data on a different track. So, to read data on track 1 of one plate, then data on track 50 of a different plate, and finally data on track 1 of the first plate again, the entire read/write arm must be moved twice. Positioning the arm like this requires a significant amount of time compared to the data transfer time.

Hard drive structure



To minimize the time needed to access data, you should prevent the data from being spread across multiple tracks. One way to optimize access time to a group of data is to write that data sequentially on a single track. If the data doesn't fit on a single track, then write in on the same track of a different plate. By doing this, the read/write arm doesn't need to be moved. Instead,

only the appropriate read/write head needs to be selected to read the desired data. Selecting (changing) heads is much faster than physically moving a mechanical read/write arm to change tracks.

The term cylinder is used to describe the multiple plates stacked on top of each other. A cylinder refers to all tracks that have the same track number but are located on different disk plates.

Disk Drives And Diskette Formats

Before describing the diskette BIOS routines, let's review the various diskette formats. Today's powerful PCs feature at least one of the following diskette formats:

1. 3.5-inch, 1.44 Meg high-density (HD)
2. 3.5-inch Super high density
3. 5.25-inch, 1.2 Meg high-density (HD)

Many sizes and formats have been used over the years (maybe you heard of the 8-inch floppy diskette?). The 3.5-inch, 1.44 Meg high-density (HD) format is currently the more popular of the three formats listed above. Its dominance is likely to continue for some time.

Diskette formats

Diskettes require formats that specify the exact qualities of a volume. Non-DOS diskettes cannot be read by DOS unless a special device driver is available. The following tables show the formats available for 3.5-inch and 5.25-inch diskettes:

3.5-inch diskettes formats					
Type	Density	Capacity	Tracks	Sectors	DOS Version
DS DD	135 tpi	720K	80	9	3.2
DS DD	135 tpi	1.44 Meg	80	18	3.3
DS DD	270 tpi	2.88 Meg	80	36	3.3

5.25-inch diskettes formats					
Type	Density	Capacity	Tracks	Sectors	DOS Version
SS SD	40 tpi	160K	40	8	1.0
SS SD	40 tpi	100K	40	9	2.0
DS SD	40 tpi	320K	40	8	1.1
DS SD	40 tpi	360K	40	9	2.0
DS HD	96 tpi	1.2 Meg	80	15	3.0

5.25-inch floppy disk drives and diskettes

The 5.25-inch floppy disk drive became the first PC standard floppy. Its still standard equipment for many PCs. Original 5.25-inch diskettes were single density diskette with four sectors per track and 40 sectors on each side. This provided a capacity of 80K per side or 160K for floppy disk drives with two read/write heads. Its transfer rate of 125K/sec is agonizingly slow compared to today's standards.

The single density diskettes were followed by double density diskettes. These diskettes doubled the number of sectors per track to eight while retaining the number of tracks per side (40). So, the capacity of the diskette is increased to 160K for a single-sided floppy disk drive and 320K for a double-sided floppy disk drive. The data transfer rate also doubled (to 250K/

sec). This 320K format was short-lived. Eight sectors doesn't quite fill up a track on double density diskette. So, there is still room for a ninth sector. Adding an extra sector to each track increases the capacity to 360K.

When the IBM/AT computer was introduced, a new format was also introduced. The new format, called high density diskette, gives the 5-1/4" floppy diskette a capacity of 1.2 Meg. The number of sectors per track was increased to 15 and the number of tracks per side was doubled to 80. As with double density diskettes, both sides of the diskette are used for this high density format. Theoretically it's possible to have 16 sectors on each track. But for practical considerations, developers settled on the 15 sectors per track arrangement to ensure a reliable floppy disk drive. High capacity floppy disk drives rotate at 360 RPM.

Instead of the 40 tracks used by the double density format, the high density format squeezes 80 tracks onto a diskette. So standard double density floppy disk drives cannot read or write this format. Only newer model MF (multifunction) drives are capable of reading and writing this format. A MF drive also adjusts to the standard double density format, which makes it possible to read and write standard 360K diskettes. In MF drives, the data transfer rate can be adjusted and the rotational speed can be reduced to the normal speed of 300 revolutions a minute. However, the higher number of sectors per track means that earlier PC and XT floppy disk drives cannot read or write to these high density diskettes. These drives cannot achieve the necessary data transfer rate of 500 kilobits per second because they cannot be designed or configured for the task.

Increasing the recording capacity from double density to high density isn't simply a matter of drive electronics. It's also a question of the "granulation" of the magnetic material on the diskette. The smaller the single magnetic particles, the more information can be recorded on a given surface. This is also the reason why double density diskettes can never be formatted error free in AT disk drives at 1.2 Meg; the granulation on those diskettes is simply too coarse.

While it may be possible to format a double density diskette at high density, these diskettes can usually be read only on the PC on which they were formatted. Other PCs will simply give you read errors, making it almost impossible to use the diskette. This is caused by the variances in the positioning of the read/write heads on different floppy disk drives.

The differences between read/write head positioning may only amount to fractions of a millimeter. A single track may actually be wider than the read/write head, allowing the head to be positioned within the track's range. Even allowing for variances in track size, this allows the head to correctly read the information. However, this causes problems when reformatting a standard double density diskette using a high density drive, because the high density drive's formatting capabilities may cause problems when a PC/XT double density drive attempts to read the newly formatted disks. The smaller read/write tolerances may cause track skipping and read errors.

Something similar occurs if you try to format a high density diskette in a double density floppy disk drive. Usually this is also doomed to failure or results in read errors on other drives. So, purchasing expensive HD (High Density) diskettes for your PC or XT disk drives isn't necessarily a good idea.

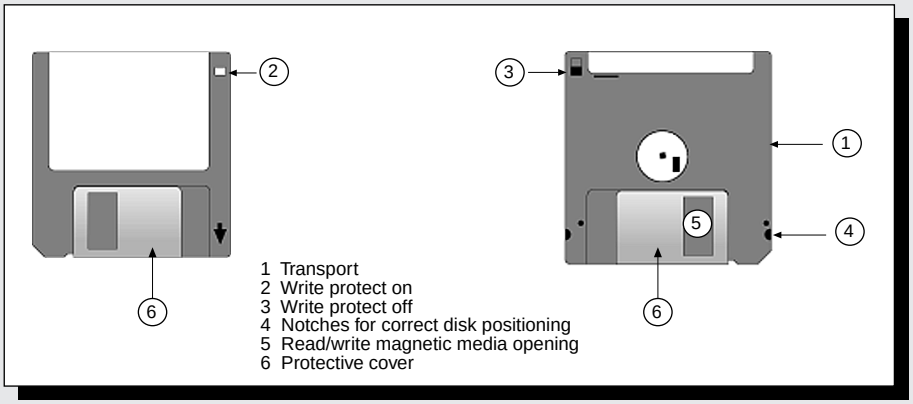
3.5-inch floppy disk drives and diskettes

Although 5.25-inch floppy disk drives are still widely used, they are being replaced by the smaller 3.25" floppy disk drives. These smaller drives first became popular on the laptop computers. Now, 3.5-inch floppy disk drives are also used on most desktop PCs. The 3.5-inch diskettes are preferred because of their convenient size and the sturdiness of their rigid plastic case. The magnetic surface of a 3.5-inch diskette is covered by a sliding metal door to protect the data from damage by dust and dirt particles.

The 3.5-inch floppy disk drives record data in either double density or high density. Double density has a capacity of 720K, with 9 sectors on a track and 80 tracks on each side of the diskette. A 3.5-inch diskette has a considerably higher track density than a 5.25-inch diskette, especially since the diameter of the magnetic media is much smaller.

The original 3.5-inch floppy disk drives have a double density format with a capacity of 720K. The newer 3.5-inch floppy disk drives have a 1.44 Meg capacity. A 1.44 Meg floppy disk drive is now the standard in 3.5-inch drives today. It also has 80 tracks per side, but has 18 sectors per track. This also doubles the data transfer rate, which makes it impossible to use these diskettes in the double density 3.5-inch floppy disk drives of earlier PCs and XTs.

Structure of a 3.5-inch diskette



A 3.5-inch high density floppy disk drive performs like a 5.25-inch MF floppy disk drive in that it can adjust to reading, writing and formatting double density diskettes.

Recently, a new 3.5-inch format was introduced. Diskettes using this format have a capacity of 2.88 Meg. This format is called extra high density (ED). ED diskettes have double the number of sectors (36) and can only be read by the new ED floppy disk drives. Like HD drives, ED floppy disk drives are also downwardly compatible. This means they are able to process both double density and high density diskette formats.

Floppy disk drives that can read and write the different formats must be able to determine the format in which the diskette is written. Then it can pass this information to the BIOS before accessing the data on the diskette. Finding the format of a 5.25-inch diskette isn't easy, because this information can be determined only by reading the data. So, the diskette must already be formatted.

However, the capacity of a 3.5-inch diskette can be determined by a small hole, which is located on the opposite side of the write-protect slider. Within the floppy disk drive itself is a light-sensitive sensor that can detect the presence or absence of the hole. Unlike high density diskettes, double density diskettes don't have this hole. Extra high density diskettes also have a hole, but the hole is located in a different position.

Disk drives and their controllers

A floppy disk drive consists of a motor that rotates the diskette at 300 revolutions per minute (360 RPM for HD 5.25-inch) and a mechanism for moving the read/write head. The drive also has an electronic component, called a data separator. The data separator converts a voltage into a binary data stream as the read/write head passes over the surface of the diskette.

The floppy drive is controlled by a separate diskette controller, which is either part of the computer's motherboard or on an I/O card in one of the computer's expansion slots. The main functions of the diskette controller are performed by an NEC PD765 or a similar chip from another manufacturer. Only NEC chips or NEC compatible chips are able to work with the ROM-BIOS. After all, it's the ROM-BIOS that uses this chip to control access to the floppy disk drive. Although it's possible to adapt the ROM-BIOS to work with another diskette controller, most manufacturers of ROM-BIOS systems use the established standard of the NEC PD765.

However, this standard causes problems with the new ED floppy disk drives. These drives have twice the track capacity (36 instead of 18 sectors) and the twice the data transfer rate. But both parameters are unknown to the BIOS. A ROM extension on the controller card can be used to avoid these limitations. The ROM extension wedges itself into the ROM-BIOS when DOS is first started. Then it manages all access to these floppy disk drives. Since the ROM extension is intended to be used only in real mode, it cannot be used if the computer is running protected mode.

Operating systems such as UNIX or OS/2 run in protected mode. These systems rely on the diskette controller to handle all the details of accessing the drives. So, they'll fail with new floppy disk drives unless special device drivers are written into

the operating system. Hopefully a BIOS standard for the new ED drives will be developed soon, making ROM extensions unnecessary. Since Windows will be taking over more of the BIOS tasks in the future, this problem will become more prevalent for PC users.

Accessing Floppy Disk Drives With The BIOS

There is a complete set of BIOS functions that access floppy disk drives. Interrupt 13H is used to call these functions. This interrupt is also an interface to the hard drive utilities of the BIOS. Wherever possible, similar floppy and hard drive functions share an identical function number. To differentiate between the drives, the drive specification is passed to the function in the DL register.

For floppy disk drives, either the value 0 (for drive A:) or 1 (for drive B:) is used. A few disk controllers support four floppy drives by providing a BIOS extension that also accepts the values 2 and 3 for the other two floppy disk drives. Hard drives are specified by the values 80H and 81H.

Here also, you must distinguish between the PC/XT-BIOS and the AT-BIOS. For example, there are a few BIOS functions that are specific to MF floppy disk drives. The following table lists the diskette functions of the BIOS interrupt 13H:

Diskette functions of the BIOS interrupt 13H

No.	Tasks	PC/XT	AT	No.	Tasks	PC/XT	AT
00H	Reset	Yes	Yes	08H	Request Format	Yes	Yes
01H	Read status	Yes	Yes	15H	Define drive type	No	Yes
02H	Read	Yes	Yes	16H	Detect diskette change	No	Yes
03H	Write	Yes	Yes	17H	Determine diskette format	No	Yes
04H	Verify	Yes	Yes	18H	Determine diskette format	No	Yes
05H	Format	Yes	Yes				

Notice functions 17H and 18H have the same task. This isn't an error. Function 18H was introduced for the 3.5-inch HD drives. The older function 17H isn't capable of supporting the drive, so it was replaced by a new function. We'll discuss this in more detail later.

Drive status

These BIOS functions also have another similarity. They return a status or error code. This status code is returned to the caller in the AH register. A nonzero value and a set carry flag indicate an error.

Status and error codes of the BIOS diskette functions

Code	Meaning	Code	Meaning
00H	No error	08H	DMA overflow
01H	Illegal function number	09H	Data transfer past the segment limit
02H	Address marking not found	10H	Read error
03H	Attempt to write to write-protected diskette	20H	Diskette controller error
04H	Addressed sector not found	40H	Track not found
06H	Diskette was changed	80H	Time out error, drives does not respond

You can also determine the diskette status through function 01H. Simply pass function number 01H in the AH register and the drive specification value in the DL register. After the function call, the drive status is returned to you in the AH register.

Resetting the floppy disk drive

After an error you must reset the floppy disk drive. To do this, use function 00H. Pass function number 00H in the AH register and the drive specification value in the DL register. After the function call, the current drive status is returned to you in the AH register. It doesn't matter whether you pass 0 or 1 as the drive specification; all the floppy disk drives will be reset. Remember the value in the DL register isn't ignored. Entering a value greater than 80H resets the hard drives.

Prompt for the drive type

A program needs to know the type and format of a floppy disk drive to use it. You can use the 08H and 15H functions to determine this information. Function 08H, which is found in the PC/XT-BIOS, is used to distinguish between the different floppy and diskette formats. Pass function number 08H in the AH register and the drive specification in the DL register. The following illustration shows the information that's returned.

Any error code is returned in the AH register with the carry flag set. By using this function, you can determine whether a given drive is installed to demonstrate, for example, that a second, third, or even fourth drive is present.

The value in the BL register is especially important. This value not only reveals the floppy disk drive type (3.5-inch or 5.25-inch), but also shows the diskette format (DD or HD). However, this value doesn't necessarily describe the format of the diskette that is located in the drive; it describes only the highest possible density. This information is taken from the CMOS-RAM, in which this information is stored when the computer is originally setup. The Drive Type value is standardized, but doesn't yet include the new ED 3.5-inch drives. However, in the future these new types will most likely appear under Drive Type 05H. Although the number of sectors, tracks, and heads can be derived from the Drive Type value, this information is also specified explicitly in the CH/CL, DH/DL registers.

Information returned by the 08H function	
Register	Information
BL	Drive Type 01H = 5.25-inch, 360K 02H = 5.25-inch, 1.2 Meg 03H = 3.5-inch, 720K 04H = 3.5-inch, 1.44 Meg
DH	Maximum number of sides (always 1)
CH	Maximum number of tracks
CL	Maximum number of sectors
ES:DI	Pointers to DDPT

The DDPT, which is referenced by the pointer in the register pair ES:DI, is the Disk Drive Parameter Table. This table contains a parameter the BIOS needs for programming the diskette controller. You'll find a description of this table later on in this chapter.

Function 15H has a different purpose. This function is only supported by ATs and their FM drives. Unlike PC/XT floppy disk drives, these drives are able to detect when a diskette has been changed. This feature is important for programs that depend on the presence of a specific diskette. This is true especially for DOS, which reads the FAT table before it accesses a diskette to determine which sectors of the diskette are occupied and which sectors are still unused.

If the diskette is changed without DOS knowing about it, DOS may continue to use the contents of the original diskette's allocation table (FAT) and may inadvertently overwrite and/or destroy data on the newly inserted diskette. However, if the same diskette is reinserted into the floppy, DOS won't have to reread the FAT again.

Drive codes of function 15H	
Code	Meaning
AH = 00H	Drive not present
AH = 01H	Disk drive, does not recognize diskette changes
AH = 02H	Disk drive, recognizes diskette changes
AH = 03H	Hard drive

Function 15H of the BIOS helps determine if a diskette has been replaced. Pass function number 15H in the AH register and the drive specification in the DL register. The table on the left shows what information is returned after the function call.

Reading diskette sectors

Reading diskette sectors is one of the most basic tasks the BIOS performs. Function 02H reads diskette sectors. Remember that this function can read several sectors through a single call if they're on same track and contiguous. Remember the data isn't transferred to a fixed memory location. Instead, the address of a buffer is passed in the register pair ES:BX. Register ES contains the segment address of the buffer and register BX contains the buffer's offset address.

Register when calling function 02H	
Register	Information
AL	Number of sectors to be read
DL	Drive specification value
DH	Side (0 or 1)
CL	Sector number (1 to N)
CH	Track number (0 to N-1)
ES:BX	Address of the buffer for the data to be read
AH = 03H	Hard drive

After the function call, the error status is returned in the AH register and the number of read sectors read is returned in the AL register. If the carry flag is set, it signals an error.

If you're using an MF drive, you can use a trick to determine the format of a diskette. By trying to read a diskette with a sector greater than 9, you can determine whether a diskette is DD or HD. Since the maximum number of sectors per track on a DD diskette is 9, function 15H will return a disk status error. The track number isn't important, but it should be less than 40. If a disk status error is returned, don't immediately abort the operation. You should repeat all read, write, and format operations at least three times before you give up and assume there is a "real" error. Often an operation fails the first time, but succeeds when you try it a second or third time. Perhaps the read/write head wasn't positioned properly the first time or the floppy drive wasn't synchronized to the electronics yet (see the "Recording Information On The Hard Drive" section in this chapter).

In cases of errors, you don't have to worry about the validity of the data because the drive uses parity checking to ensure the data in each sector is correct.

Writing diskette sectors

Function 03H is used to write to individual sectors. The parameters are passed according to the table on the right. Remember the buffer, to which the ES:BX register pair points, must contain the data to be written to the diskette.

Register when calling function 03H	
Register	Information
AL	Number of sectors to be written
DL	Drive specification value
DH	Side (0 or 1)
CL	Sector number (1 to N)
CH	Track number (0 to N-1)
AL	Number of sectors to be read
ES:BX	Pointer to the buffer containing the data

Verifying diskette sectors

Function 04H tests whether data have been correctly transferred to the diskette. The data in the memory aren't compared with the data on the diskette. Instead, a CRC value is used to determine whether the data was transferred correctly. CRC, which is an abbreviation for "Cyclical Redundancy Check", is a very reliable procedure for verifying accuracy. This procedure combines the values of each byte within the sector with a checksum through a complicated mathematical formula.

Since most disk drives are very reliable, most programmers consider this routine to be unnecessary and don't use it. DOS uses this function when writing data only if the DOS VERIFY ON command is active. The parameters for function 04H are the same as for function 02H and 03H except that a buffer address isn't required.

Formatting individual tracks on a diskette

Function 07H is used to format an entire diskette. But it's also possible to format individual tracks on a diskette. To do this, first use function 18H to tell the BIOS which format to use. The function number 18H is passed in the AH register, the drive specification value in the DL register, the number of tracks in the CH register, and the number of sectors per track in the CL register. After the function call, the carry flag signals the specified format is supported by the floppy disk drive. In this case, the register pair ES:DI is a pointer to DDPT, which is required for subsequent formatting functions.

We'll describe the DDPT in more detail later. For now, remember the pointer must be passed to interrupt vector 1EH, in which the BIOS keeps a pointer to the current DDPT.

Register when calling function 05H	
Register	Information
AL	Number of sectors in the track
DL	Number of the drive
CH	Number of the track
DH	Side (0 or 1)
ES:BX	Pointer to format table

After function 07H sets the desired format and the DDPT pointer is passed to interrupt vector 1EH, you can start the actual formatting process. To do this, use function 05H. This function formats a complete track. Although you can format individual sectors with 128, 256, 512, or even 1024 bytes per sector, only a 512 byte format can be used under DOS. This is because DOS supports only this size sector.

To use function 05H, pass the drive specification value in the DL register, the diskette side in the DH register, the number of sectors per track in the AL register, and the track number in the CH register (0 through 39 or 0 through 79).

You'll see the ES:BX register pair points to a "format table". This table represents the formatting attributes. The table on the right shows this as an array of 4-byte entries (one for each sector to be formatted). Although the track number and diskette side is passed to function 05H, it must be repeated in the table. The sectors are physically created in the same sequence as the table entries. So it's possible to format the first entry as sector number 1 and the second entry as sector number 7. The logical sector number is recorded in the header of each sector on the diskette, so the floppy disk drive can later identify the sector that is being searched.

ES:BX register pair "format table"	
Offset	Meaning
0	Track to be formatted
1	Diskette side (always 0 for one-sided diskettes): 0 = Front side 1 = Back side
2	Number of the sector
3	Number of bytes in this sector: 0 = 128 bytes 1 = 256 bytes 2 = 512 bytes 3 = 1024 bytes

Since the BIOS doesn't define the logical sector numbers, you can change the interleaving. Generally only hard drives use interleaving, as we'll see in Section 6.7. Since interleaving doesn't have any advantages for floppy diskettes, you should number the sectors consecutively when you create the format table.

The number of bytes per sector don't have to be identical either, since these numbers are defined explicitly for each sector in the table. You can change the number of bytes per sector to develop a form of copy protection, for example. We'll soon see how this is done. A program at the end of this chapter shows how to format diskettes using functions 18H and 05H.

Disk Drive Parameter Table

To program the diskette controller, the BIOS needs the physical formatting information described above and some additional information. We've already introduced you to the Disk Drive Parameter Table (DDPT). The ROM BIOS contains a table for every drive and supported diskette format. Also, you can define your own DDPT, since the BIOS always references the current DDPT through a FAR pointer, which is contained in the memory locations in which the interrupt vector 1EH is usually found. Since neither DOS nor the PC hardware use interrupt 1EH, you can change the contents of these memory locations.

DOS creates its own DDPT. This DOS DDPT is designed to speed up access to the diskette. The table is 11 bytes in size, as shown by the table on the following page. Not all the parameters can be changed. However, the entries that are marked with an asterisk may be changed.

The first field of the DDPT table actually has two subfields: The step rate (bits 4-7) and the head unload time (bits 0-3). The step rate describes the time the controller has to move the read/write head from one track to another. This value is represented as milliseconds, with the value 0FH representing 1 ms, 0EH representing 2 ms, 0DH representing 3 ms, etc. The head unload time describes the time the read/write head has to lift up off of the surface of the diskette, for example when changing tracks. It's specified as a factor of 16 ms. The default value 0FH (240 ms) is extremely conservative and can usually be lowered.

Diskette functions of the BIOS interrupt 13H					
Offset	Meaning	Type	Offset	Meaning	Type
*00H	Step rate and head unload time	1 BYTE	06H	DTL (Data Length)	1 BYTE
*01H	Head load time	1 BYTE	07H	Length of GAP3 when formatting	1 BYTE
*02H	Post run-time of diskette motor	1 BYTE	*08H	Fill character for formatting	1 BYTE
03H	Sector size	1 BYTE	*09H	Head settle time	1 BYTE
04H	Sectors per track	1 BYTE	*0AH	Time to run up of diskette motor	1 BYTE
05H	Length of GAP3 when reading/writing	1 BYTE			

The second field is also two subfields: The head load time (bits 1-7) and the DMA flag (bit 0). The head load time is the time the read/write head has to settle to the surface of a track. This value is expressed as a factor of 2 ms. In accessing a diskette, it's usually necessary to wait much longer for the diskette motor to reach its required speed. So it's common to specify a very low value (1 or 2) for the head load time. The DMA flag is represented by bit 0. This flag must always be set to 0.

The third field is the post runtime of the diskette motor after a diskette operation. This is the period of time that elapses until the diskette motor is switched off when no other diskette operations are being performed. Because it takes a relatively long time to get the motor running, it shouldn't be switched off immediately after each diskette access. This value is related to a cycle of approximately 18 ticks per second (1 tick is approximately 55 ms). So a value of 18 represents a post runtime of about one second. The default value is 25H, which is approximately two seconds.

The fourth field specifies the number of bytes per sector that can be used in a read or write operation. This corresponds to the values for formatting a sector, so it usually contains the value 3 for 512 bytes per sector. To read or write to sectors with different sector sizes, you must first enter the appropriate value in this field.

The next field at offset address 04H is the maximum number of sectors per track, which depends on the selected diskette format. The next three fields refer to the coding and decoding of sector information, which is stored on the diskette along with the actual data. You should never tamper with these values. However, the field at offset address 08H can be changed. This field contains the ASCII code of the fill character to be used when the diskette is being formatted. During formatting, as the sectors are created, they are also given a fixed contents. The default fill character is a division sign (ASCII code 246).

The next field contains the head settle time. After the read/write head travels from one track to another, a short delay is needed to allow the vibrations from this movement to subside. Only then can the read/write head perform the subsequent data access properly. The value in this field represents a delay in milliseconds. The default value is 25 ms. The last field in the DDPT specifies the time it takes for the diskette motor to attain its operating speed. The value in this field is a factor of 1/8 seconds. While DOS defaults to a value of 1/4 second, the BIOS equivalent is 1/2 second.

Sample programs

Changing the various values in the DDPT won't produce performance miracles. However, you'll probably want to experiment with it. We've developed two small programs in Pascal and C to set the various parameters of the current DDPT. However, the programs won't change all the parameters because this would be too dangerous.

The programs DDPTP.PAS and DDPTC.C both work according to the same principle. You call both programs from the DOS command line without specifying any parameters. Each program displays the contents of the current DDPT (the DDPT of the last accessed disk drive). To address a certain disk drive, use the DIR command before the DDPTP or DDPTC program.

You can change the contents of a DDPT field by typing the command with the appropriate parameter. The parameter is a two-letter code (the code that appears on the screen when the fields are displayed), followed by a colon, and then followed by a two digit hexadecimal number that represents the new value.

For example, if you type the following command:

```
DDPTP MA:04 SR:08
```

the starting time of the diskette motor is set to one-half second and the step rate is lowered to 8 ms. This command will work only if the DDPT is in memory, not the one in the ROM-BIOS. Each program will indicate whether you're trying to change the contents of the ROM.

You'll find the following program(s) on the companion CD-ROM

DDPTP.PAS (Pascal listing)
DDPTC.C (C listing)

Do it yourself formatting

Programmers usually don't have to write data directly to or read data directly from a diskette using the BIOS. Generally, your application programs will be working with files. For this purpose, it's better to use the DOS functions. However, when you're formatting diskettes you must call various BIOS functions.

The DFP.PAS and DFC.C programs perform this task. These programs are replacements for the DOS FORMAT program. Similar to FORMAT, these programs not only format the diskette, but also create the various data structures that DOS expects. Among these are the boot sector, the root directory of the diskette, which is initially empty, and the FAT. For more information about these data structures and the general structure of mass storage systems under DOS, refer to Chapter 28.

DFP.PAS and DFC.C can process all known DOS formats (360/1200 on 5.25-inch diskettes and 720/1440 on 3.5-inch diskettes). Use the following model to call them:

```
         DFP Drive   Format   [ NV ]
             |          |          |
             |          |          |
   A: or B:  |          |          |
                        |          |
    360, 720, 1200, 1440|          |
                                   |
                  NV = No Verify   |
```

Since both programs are based on the same algorithm and work with the same data types and constants, we'll discuss them together. Within the main program, the first argument of the DOS command line is analyzed. It's assumed that this argument will be the drive identifier (letter). This value is converted into the drive number (0 or 1). Next, the format of the drive is determined by using the GetDriveType procedure. To do this, we use function 08H of the BIOS disk interrupt, which returns a type code between 0 and 4. Within the program, this code is represented by the respective constants NO_DRIVE, DD_525, HD_525, DD_35, and HD_35.

The older model PCs and XTs don't support function 08H. In this case, the carry flag is set after the function call to indicate an error. The program then defaults to a DD 5.25-inch drive that supports only a 360K format.

After GetDriveType() confirms the specified drive exists, the program then determines the logical and physical formatting parameters through the GetFormatParameter function. The format specified in the command line is passed to GetFormatParameter as a string along with the type code and two variables of type PhysDataType and LogDataType. You can see the organization of the two data structures in the Pascal version:

```
type DdptType = array[ 0..10 ] of byte;            { Structure for DDPT }
     DdptPtr = ^DdptType;                            { Pointer to DDPT }

     PhysDataType = record            { physical format parameters }
       Seiten,                    { desired side number of diskette }
       Spuren,                             { Number of tracks per side }
       sektoren : byte;                   { Number of sectors per track }
       DDPT     : DdptPtr;     { Pointer to Disk Drive Parameter Table }
     end;
```

```
 LogDataType = record                          { DOS format parameters }
   Media,                                              { Media-Byte }
   Cluster,                             { Number of sectors per cluster }
   FAT,                                 { Number of sectors for the FAT }
   RootSize : byte;                    { Entries in the root directory }
end;

SpurBufType = array[ 1..18, 1..512 ] of byte;    { Buffer for track }
```

PhysDataType contains the physical parameters needed for formatting. These are the number of sides, the tracks per side, and the number of sectors per track. Also, a pointer to the DDPT is stored here because both programs work with their own "private" DDPTs to speed up the formatting.

While the information in PhysDataType is needed for physical formatting, the information in LogDataType is needed for logical formatting, which applies to using different DOS data structures. That's why the DOS media ID, the number of sectors per cluster, the size of the FAT in sectors, and the number of entries in the root directory are recorded here.

The variables of type PhysDataType and LogDataType within GetFormatParameter are initialized with a series of typed constants (or STATIC variables in C), in which the necessary information for all supported formats is recorded. These constants, which are called DDPT_360, LOG_1200, or PHYS_720, can be identified quickly when looking through the listings. Before the procedure copies the parameters to the passed variables, it checks to determine whether the format specified in the command line is a valid one for the drive. The procedure displays the results of this test to the caller as its return value, which is FALSE if drive and format aren't valid.

After these checks, program execution continues by calling DiskPrepare. This procedure uses BIOS function 18H. Function 18H returns a pointer to the DDPT to the caller. The DDPT is part of the selected format. This pointer is ignored by the two programs, however, because a different DDPT is used by FormatGetParameter(). After DiskPrepare, the address of this DDPT is stored in the interrupt vector for interrupt 1EH. Remember the contents of this vector are first saved so the original DDPT address can be restored later.

Formatting then occurs using the PhysicalFormat function. The third parameter of the DOS command line indicates whether to verify the tracks. When formatting, PhysicalFormat uses the FormatTrack procedure to format one track at a time. FormatTrack calls the BIOS disk function 05H in a nested loop for tracks 0 to N; it formats side 0 first and then side 1 of each track. Obviously, it's also possible to format the entire first side and then the entire second side. However, this would take much longer because the read/write head would have to travel over the entire diskette twice. With this method, the drive is constantly switching between head 0 and head 1, but the formatting is performed much faster than moving the read/write arm from track to track.

After FormatTrack, PhysicalFormat calls the VerifyTrack procedure to determine the validity of the data (i.e., whether the contents of a sector and the corresponding CRC checksum match). This is performed only if FALSE is returned for the VERIFY parameter. VerifyTrack, like FormatTrack and WriteTrack, which we haven't discussed yet, is simply a procedure name for the corresponding BIOS function. If the BIOS reports an error, this call is repeated several times before a "real" error indication is finally returned to the caller. The maximum number of attempts is determined by the MaxVersuch constant, which is defined at the beginning of the listings. By setting this constant to one, both versions of the program will frequently report an unsuccessful format because errors occur more often than you might think.

Now let's return to the main program. If the diskette was perfectly formatted with PhysicalFormat, then the last processing step begins. This step involves logically formatting the drive using LogicalFormat. As we mentioned, in this step the different data structures that DOS needs for managing files are written to the diskette. We discuss this in detail in Chapter 28.

One interesting structure is the boot sector, which must be written to the diskette if it is to contain the DOS operating system. The contents of the boot sector are defined at the beginning of the program in the BootMaske variable. You'll find the details about the data and the small machine language program in Chapter 28. However, you should know the meaning of the

BootMes variable, which immediately follows the boot sector. This variable contains the string that appears on the screen when the computer is booted. It's written to the diskette with the boot sector.

You can alter the contents of this string. For example, add your own name or the name of your company so it appears on the screen when you boot the computer from the diskette. However, remember the end of the sector is indicated by a byte with the value 00H.

You'll find the following program(s) on the companion CD-ROM

DFP.PAS (Pascal listing)
DFC.C (C listing)

The end of LogicalFormat is essentially the end of program execution, because nothing happens in the main program except writing the status message.

Using BIOS To Access The Hard Drives

In this section we'll describe the BIOS functions for accessing hard drives. However, before we begin, we must warn you about experimenting with these functions. Unlike a floppy disk drive, in which you can insert an unused diskette for testing, a hard drive cannot be tested in this way. Using write and format functions carelessly can lead to irreparable data loss. Because of the structure DOS imposes on a hard drive, destroying one sector can cause all files and directories to disappear because DOS may no longer know where they are on the hard disk.

So, if you would like to "test" the BIOS functions, be sure to make a complete backup of your entire hard drive beforehand, or use another computer, if available. This is the only way you can avoid data loss, because even the most elaborate hard drive utility may not be able to help you.

The BIOS hard drive interrupt

As we mentioned, the hard drive shares interrupt 13H with the floppy disk drives. Although the functions for the hard drive and the floppy disk drives are identical, the BIOS controls the hard drive differently than the floppy disk drive. For this reason, the BIOS contains a module for controlling the hard drive and a separate one for controlling the floppy disk drives.

When interrupt 13H is called, the device number in the DL register determines whether a floppy or hard drive is being addressed. A value of 80H represents the first hard drive, while 81H represents the second hard drive. It's not possible to address more than two hard drives via the BIOS.

The functions of the hard drive BIOS have existed since the introduction of the XT. The original PC BIOS didn't have them. In 1981 no one thought of putting hard drives in microcomputers. When the AT and PS/2 model from IBM was introduced, some additional functions were added, as the following table shows:

Function	Task	Origin	Function	Task	Origin
00H	Reset	XT	0CH	Move read/write head	XT
01H	Read status	XT	0DH	Reset	XT
02H	Read	XT	0EH	Controller read test	only PS/2
03H	Write	XT	0FH	Controller write test	only PS/2
04H	Verify	XT	10H	Drive ready?	XT
05H	Format	XT	11H	Recalibrate drive	XT
08H	Check format	XT	12H	Controller RAM test	only PS/2
09H	Adapt to foreign drives	XT	13H	Drive test	only PS/2
0AH	Extended read	XT	14H	Controller diagnostic	XT
0BH	Extended write	XT	15H	Determine drive type	AT

Normal application programs don't usually access the hard drive through the BIOS. We'll describe only the most important functions in this section. You can find more information in the Appendix (located on the companion CD-ROM in which all the functions are described.

Status code

The hard drive functions use the carry flag to indicate an error. If the carry flag is set, then an error has occurred and the error status code is returned in the AH register. The following table lists the meanings of these codes:

Error codes when calling BIOS Disk Interrupt 13h for accessing the hard drive			
Code	Meaning	Code	Meaning
00h	No error	10h	Read error
01h	Function number or drive not permitted	11h	Read error corrected by ECC
02h	Address not found	20h	Controller defect
04h	Addressed sector not found	40h	Search operation failed
05h	Error on controller reset	80h	Time out, unit not responding
07h	Error during controller initialization	AAh	Unit not ready
09h	DMA transmission error.Segment border exceeded.	CCh	Write error
0Ah	Defective sector		

When one of these error occurs (except for error 1), you should first reset the drive and retry the function. Usually, the operation will then be successful. If error 11H is returned after a read function, the data isn't necessarily invalid. Actually, this status code indicates that a read error was detected, but was able to be corrected using an ECC (Error Correction Code) algorithm. This procedure is similar to the CRC procedure used by floppy disk drives. The individual bytes of a sector are calculated through a complicated mathematical formula. The resulting sum is written to the sector on the hard disk as four additional bytes. If a read error is detected, it can usually be corrected by using the ECC.

Using the hard drive functions

The hard drive functions also use registers for passing parameters. The function number is passed in the AH register. When the hard drive number must be identified, its value is passed in the DL register. The value 80H always represents the first hard drive, and 81H represents the second hard drive. The number of the read/write head and the side (0 or 1) are passed in the DH register.

The CH register specifies the cylinder number. Since you can represent only 256 cylinders with this 8-bit register and the hard drive of an XT has more than 306 cylinders, this register alone cannot specify the cylinder number. For this reason, bits 6 and 7 of the CL register are "appended" to the value in the CH register to determine the cylinder number. They form bits 8 and 9 of the cylinder number, so a maximum of 1024 cylinders (numbered 0 to 1023) can be addressed. Bits 0 to 5 of the CL register specify the sector number (1 to 17 per cylinder). If more than one sector is being accessed at the same time, the AL register specifies the number of sectors. In read and write operations you must also specify the address of a buffer, from which the data is written or to which the data are transferred. In this case, the ES register indicates the segment address and the BX register indicates the offset address of the buffer.

Resetting the hard drive controller

One function that doesn't require all the parameters is function 00H, which, like function 0DH, resets the controller. For example, after an error occurs, this function is routinely performed before the next data access. The only parameter needed is the hard drive number that is passed in the DL register.

Determining the status of the hard drive

Using function 01H, you can determine the status of the hard drive. Again, the drive number whose status is being checked is passed in the DL register.

Reading hard drive sectors

Function 02H reads one or more sectors of the hard drive. On each call to this function, you can read a maximum of 128 sectors. Perhaps you're wondering why the maximum is 128 instead of 256 sectors. The hard disk controller uses DMA capability to transfer data between the computer's memory and the hard drive. However, the DMA components can transfer a maximum of 64K of data at one time. This is equivalent to 128 sectors (64K = 128 sectors * 512 bytes/sector). Another restriction is the DMA components can transfer data only within a single memory segment. So, the read/write buffer is usually aligned to the start of a memory segment. Remember the ES:BX register pair point to the buffer. In this case, the ES register will point to the start of this segment and the BX register will have a zero offset.

When you use function 02H to read more than one sector per call, the sectors are read in the following order: First, the sectors in the specified cylinder and side are read in ascending order (by sector number). When the end of the cylinder is reached, the first sector on the same cylinder, but on the next head, is read. Sectors on the next cylinder aren't read until after the last head in the same cylinder is reached and there are sectors remaining to be read.

Writing hard drive sectors

Function 03H is used to write one or more sectors to the hard drive. This function is similar to function 02H except the data is written from the buffer to the hard drive. For a description of this function, refer to the one above.

Verifying hard drive sectors

Function 04H verifies the sectors of a cylinder. However, the data on the hard drive is compared with the ECC value instead of the data in memory (which is why it isn't necessary to specify a buffer address in ES:BX). The number of sectors to be verified is specified in the AL register.

Formatting the hard drive cylinders

A hard drive must be formatted before it can be used. Function 05H performs this task. This function is similar to the function for formatting a floppy diskette. The address of a buffer is passed in the ES:BX register pair. This buffer must be 512 bytes in size, although only the first 34 bytes are used. The buffer consists of two one-byte entries for each of the 17 sectors to be formatted. The first byte indicates whether the sector is good or bad. Before calling this function, we assume that each sector is good. So we store a zero value here. The second byte is the logical sector number.

Bytes 1 and 2 of the table are used when the first physical sector of the cylinder is formatted. Bytes 3 and 4 are used when the second physical sector is formatted, etc. So, while the physical sequence is fixed, the logical sequence of sectors is defined by the two bytes of a sector specification in this table.

The most obvious way to format the hard drive is to assign a sector's physical sector number to each logical sector. However, a technique called sector interleaving is actually used to speed up hard disk performance. We'll discuss sector interleaving in more detail in the "Hard Drive Advancements" section later in this chapter.

The first byte of each table entry may contain the value 00H, which indicates the sector is good, or 80H, which indicates the sector is bad. During formatting, this byte is transferred to the sector marker that indicates that DOS shouldn't use this sector to store data.

Determining the hard drive parameters

Unlike floppy diskettes, hard drives don't have uniform characteristics. For some programs, it's important to know the hard drive's parameters. To do this, use function 08H to pass the hard drive number in the DL register.

After calling the function, the DL register contains the number of hard drives connected to the controller. The value returned may be 0, 1, or 2. The DH register contains the number of read/write heads. Since this value is relative to 0, a value of 7 means there are 8 heads. The number of cylinders is returned both in the CL register (bits 0-7) and the two upper bits of the CH register (bits 8 and 9). Again, this value is relative to 0. Finally, the number of sectors per track is returned in the lower 6 bits of the CH register. This specifies the number of sectors per track, but is relative to 1, not 0.

Initializing a "foreign" hard drive

The BIOS in each computer already contains the specifications for various hard drives. This makes it easy to select the hard drive specifications during SETUP. However, suppose the specifications for a particular hard drive aren't in the BIOS. There's another way to make the drive's specifications known to the BIOS. First a table containing the specifications is constructed. Then the address of the table is stored at interrupt 41H or interrupt 46H, depending on whether hard drive 0 or hard drive 1 is being initialized. The format of the table is predefined by BIOS and describes the characteristics of the hard drive.

Finally, function 09H, which initializes the controller with the new hard drive specifications, is called. The drive number (80H or 81H) is passed in the DL register. Usually the device driver provided by the hard drive manufacturer manages this function.

Extended hard drive sector read/write

Functions 0AH and 0BH are similar to the read and write functions 02H and 03H. However, one difference is that, in addition to the 512 bytes of data per sector that's transferred, the four ECC bytes at the end of each sector are also transferred. Since each sector is 516 bytes instead of 512 bytes, the maximum number of sectors that can be read or written at a time is 127 sectors, while functions 02H and 03H can handle 128 sectors.

Function 10H tests whether the hard drive, whose number is passed in the DL register, is ready to execute commands. If the carry flag is set to indicate the drive isn't ready, then the AH register will contain the error code.

Recalibrating the hard drive

Function 0BH is used to recalibrate the hard drive. After the function call, this function returns the error status along with the drive number in the DL register.

Self test of the hard drive controller

Function 14H is used to perform a self test. If the controller passes the test, the carry flag will be reset.

The final hard drive interrupt function is 15H, which is available only on ATs, not XTs. This interrupt returns the drive type. The drive number (80H or 81H) is passed in the DL register. If the drive isn't available, a value of 0 is returned in the AH register. A value of 1 or 2 indicates a floppy disk drive. A value of 3 indicates a hard drive. In this case, registers CX and DX contain the number of sectors on this hard drive. The two registers form a 32 bit number, with the CX register containing the high-order byte and the DX register containing the low-order byte.

Hard Drives And Their Controllers

The advancements in hard drive technology are related to four types of controllers: ST506, ESDI, SCSI, and IDE. The format, in which the data are saved on the hard drive, not only depends on the controller, but also on the data transfer rate between the computer and the hard drive.

The following table shows the maximum data transfer rates that are possible with the different controllers. However, these are theoretical maximum values that not only are seldom attained, but also are affected by other factors. Imagine a set of data on its way from the hard drive to be displayed on the screen (e.g., when you load a document into a word processor). In addition to the controller, the program must interface with many levels: the BIOS, the DOS, the application program, and perhaps one ore more TSRs.

Maximum data transfer rates of the various PC hard drive controllers

Controller	Maximum Data Transfer Rate	Controller	Maximum Data Transfer Rate
ST506	1 Meg per second	IDE	4 Meg per second
ESDI	2.5 Meg per second	SCSI	5 Meg per second

Another factor that affects the data transfer rate is the speed of the bus. This limits the speed of hard drive controllers because at least the ISA bus still operates at 8 MHz, although the speed of the CPU has already reached the 100 MHz limit. So any published values quickly diminish to about a fifth by the time the data actually appears. Next, we'll introduce you to hard drive controllers, examine their structure and describe their advantages and disadvantages. The role of the BIOS is also discussed.

ST506 controller

The first hard drives that were widely accepted in the PC world were developed for the ST506 controller and compatible controllers. As its name indicates, this controller was manufactured from Seagate.

Even today, ST506 controllers are still the most widely used controllers, even though new hard drives usually are equipped with IDE controllers. Generally, hard drives designed for hookup to an ST506 controller are identified by the label "MFM/RLL". These letters refer to the two recording methods by which the controllers save data on the hard drive. Usually you can use DIP switches to set the format the controller uses. The RLL format is preferred because it provides higher hard drive capacity. (Refer to the "Recording Information On The Hard Drive" section in this chapter for more information.)

Because of its wide distribution, the ST506 controller has set different standards in hardware control systems, partly because the BIOS conforms to this controller type. The effects of this are evident today. For example, IDE and ESDI controllers are (and must be) compatible with ST506 controllers in various ways. We'll discuss this in more detail later.

The hardware

In the ST506 standard, the hard drive and controller are two separate components. The controller is on a separate card and occupies one of the PC's expansion slots.

A controller can usually manage two hard drives. Two types of cables connect the controller to each hard drive. Each hard drive connects to the controller with its own 20-pin data cable. If both hard drives are installed, they share the 34-pin control cable. The control cable sends electrical signals to the hard drive to select the desired read/write heads, search for the desired cylinder, etc. Data to be read from or written to the hard drive is transferred over the data cable in serial and analog mode.

The controller can convert the digital information on a cylinder into bit strings. The information on magnetic media exists as values of 0 or 1. The controller can reverse the digital values as needed; this process is called flux reversal.

The data transfer rate can be as high as 5 megabits/second using MFM recording and 7.5 megabits/second using RLL recording. Since the control information still must be "filtered out" from the stream of data, the effective transfer rate of useful data is considerably lower. Even so, MFM controllers can process .5 megabytes per second and RLL controller can even process 0.75 megabytes per second. However, usually these theoretical values ignore factors such as head select time, cylinder seek time, etc., and assume that you'll read contiguous sectors. We'll discuss this in more detail in the section on interleaving.

The higher transfer rate of RLL controllers is a result of a more efficient recording scheme. You'll see that more sectors can be written to each track using RLL. MFM drives can write 17 sectors on a track, but RLL drives can fit 26 sectors on each track. In both formats, the rotational speed of the drive is 3600 RPM.

When the XT first appeared, the only controller available was the ST506. As a result, the new hard drive functions of the ROM-BIOS were developed specifically for this controller. These hard drive functions have imposed rigid limitations on PC manufacturers. For example, the number of drives is limited to two, the maximum number of cylinders to 1204, the maximum number of sectors per track to 63, and the maximum number of heads to 16. Also, the sector size is fixed at 512 bytes. When combined with all the other factors, the maximum capacity of a drive is 504 Meg.

To overcome these limitations, some hard drive controllers "trick" the system into believing there are two hard drives that are on the same drive. This makes it possible to have a hard drive with a capacity of up to 1 Gigabyte. However, an ST506 controller is too slow for a drive with such enormous capacity. So, it isn't practical to connect such large hard drives to systems with ST506-type controllers. Because of this, the ST506-type controller will gradually disappear in the coming years.

ESDI controllers

The ESDI controller was developed after the ST506 controller. ESDI, which is an acronym for "Enhanced Small Devices Interface", is found in many IBM PS/2 models. This controller represents an advancement of the ST506 model. Generally, ESDI controllers are compatible with the ST506 and can be used in computers whose BIOS is programmed to support only ST506 controllers.

Unlike the ST506, ESDI doesn't transfer every flux reversal serially to the controller via the data line. Instead, part of the decoding logic, called the data separator, is already on the hard drive. The data separator prepares the data read from the hard drive and transfers only the useful data, in digital form, to the controller.

Because the controller and the data separator work in parallel, the transfer rate can be as high as 10 megabits/second. This represents only a doubling of the transfer rate of an MFM ST506 drive. However, this is accompanied by another performance enhancement. ESDI controllers usually have a sector buffer that makes an interleave factor of 1:1 possible. What does this mean? An ST506 drive with an interleave factor of six requires six full revolutions to transfer the contents of an entire track. With an interleave factor of three, this same drive still requires three full revolutions to transfer the contents of an entire track. However, an ESDI can perform the same task in a single revolution. This results in increasing the access speed by a factor of three to six.

Also, some ESDI systems can reach a transfer rate of 15, 20, and even 24 megabits per second. However, such controllers are rare and usually quite expensive. So most ESDI controllers work with 10 megabits.

The data separator isn't the only intelligent component on an ESDI system. An ESDI hard drive also stores information about its physical format and the addresses of defective sectors and can send this information to the controller. Then the controller can perform its own SETUP, which is a task that a user has to do with ST506 drives.

The BIOS and ESDI controllers

The hard drive information is then stored in the CMOS RAM of an AT. The BIOS must know the parameters of the hard drive and pass this information to the DOS device driver.

Because the ESDI controller can request this information from the hard drive, the information in the BIOS doesn't have to match the actual characteristics of the drive. Often this discrepancy cannot be avoided, because each BIOS knows only a limited number of hard drive types and their specifications.

You'll encounter problems on an ST506-type hard drive if the installed drive doesn't appear in the BIOS list. In this case, you must select a drive, from the list, whose specifications most closely match those of the installed drive. This often means "wasting" some of the drive capacity. You must select a BIOS entry for a drive with fewer cylinders, tracks, or heads. Otherwise the system might try to access sectors that don't even exist on the installed drive, which results in an error.

Another problem is when the BIOS doesn't have a hard drive type that works with the same number of sectors per track. Then you must select a BIOS entry for a drive with the next smallest sector size, which means wasting valuable sectors in every track on the drive and increasing the number of unused sectors to an undesirable level.

With ST506 drives, this isn't a problem because they only work with 17 or 26 sectors, depending on the recording method. However, ESDI drives have 34 or 36 sectors per track. So, if the BIOS contains entries for drives with only 26 sectors, then you might end up wasting one-third of your expensive hard drive capacity.

Fortunately, most ESDI controllers can avoid this problem. The BIOS entry for a drive with a slightly smaller capacity than the ESDI hard drive is selected during SETUP. Since ESDI hard drives can send their specifications to the ESDI controller, the controller knows the physical characteristics of the drive. Using a special feature called sector translation, the ESDI controller converts the BIOS logical drive specifications into the hard drive physical specifications. The conversion takes slightly longer, but ensures the ESDI hard drive can be used with almost any BIOS entry without wasting a lot of its capacity.

The ability to perform this conversion depends entirely on the ESDI controller. Some controllers support only certain BIOS entries, while others are more flexible and can accept any format. This is also an advantage if an ESDI drive encounters the

limits imposed by the BIOS hard drive functions, for example, an ESDI drive with more than 1024 cylinders. Instead of wasting the cylinders above 1024, the controller simulates a greater number of sectors per track. Although the ESDI drive may have only 34 tracks per sector, it can pretend to have up to 63 tracks per sector. The ESDI controller then translates a logical cylinder/sector/head specification to the hard drive's physical specification.

SCSI controllers

The SCSI controller (pronounced "scuzzy") standard, isn't really a hard drive interface. Instead, it's a way to connect up to eight entirely different devices to a PC. Besides hard drives, you can connect tape backup streamers, CD-ROM drives, or scanners to a SCSI interface.

Unlike other hard drive controller standards, the SCSI (Small Computer System Interface) isn't found only in PCs, but also on many 68000 systems (Macintosh and Atari ST) and large workstations. One reason for its appeal is that you can easily couple and uncouple devices from the SCSI interface because the devices communicate with the controller through a bus that's separate from the PC bus.

Both the SCSI line specifications and the SCSI commands to control the devices are standardized. SCSI devices can be easily exchanged between different systems; only the SCSI controller must be matched to the host computer system.

The SCSI bus, which links different SCSI devices together, is usually a cable with an 80-pin plug. The bus allows 8-bit parallel data transfer. A new version, called SCSI2 allows 16-bit parallel data transfer.

Manufacturers of SCSI controllers like to tempt their customers by quoting data transfer rates of 4 or 5 Meg per second. Actually, these rates aren't possible. While the SCSI controller can handle these high data transfer rates, the hard drive connected to it cannot. So, you can expect a data transfer rate of between 1.5 and 2 Meg per second, unless you purchase an expensive EISA controller, which can manage 2.5 Meg per second.

SCSI drives continue the trend started by ESDI drives by integrating much of the control circuitry directly on the hard drive. Actually, this must be done with this system, because the controller must remain device-independent and not concern itself with the specific characteristics and features of a hard drive.

Shorter paths aren't the only advantage provided by the close proximity of the hard drive and its control circuitry. This also makes it easier to trick the controller into using a certain disk format that doesn't really exist. Like an ESDI controller, the SCSI controller asks the attached devices for its specification when the system starts and passes this information to the requester.

SCSI systems depend on their own built in BIOS. From a software standpoint, SCSI systems don't support the ST506 standard of the ROM-BIOS. The original functions of the BIOS Disk Interrupt are replaced by the SCSI BIOS. Unfortunately, the SCSI BIOS isn't useful in protected mode, in which "operating environments" such as Novell or OS/2 access the disk directly and often support only the ST506 standard. This requires a specific device driver, which may not be available for the particular operating environment. This is one of the biggest disadvantages of the SCSI interface.

However, if you have the required driver, the hard drive specifications are automatic. To install a new SCSI drive, simply connect it to the bus. The SCSI controller handles the rest by using the correct "driver ware".

IDE

The new star of hard drive controllers is the IDE controller (Intelligent Drive Electronics) interface. The IDE interface is found in almost every new PC. Its development started in 1984, when PC manufacturer Compaq asked Western Digital to develop an ST506-compatible controller that would fit on the hard drive to save space.

Using an IDE drive provides a hard drive and a controller in one. A single 40-pin cable combines the functions of a data cable and a control cable and connects the IDE drive directly to the system bus.

This is also where IDE drives get their nickname; sometimes they're called "AT Bus Drives". But IDE drives aren't limited to ATs with their 16 bit data buses. You can also use IDE drives on an 8-bit XT bus.

Many PCs have a connector for an IDE cable directly on the motherboard. With other PC's you must use a small expansion slot board, which then connects to the IDE cable.

Combining the drive and controller gives the IDE some of the same advantages of the SCSI controllers. Among these are the ability to emulate any drive format and track caching, in which the drive reads an entire track and keeps the data in an internal cache buffer until it's needed. IDE drives can operate with an interleave factor of 1:1, which provides faster access. IDE combines the advantages of the other three standards; it's flexible like SCSI, fast like ESDI, and is compatible to the ST506 standard so it can be connected to most PC systems easily.

Also, IDE drives have a very low power consumption, so they're ideal for laptops and notebooks.

Many IDE drives have special commands for working with laptops and notebooks. For example, these commands can put the notebook computer "to sleep", which minimizes power consumption. These commands are generally used in connection with special drivers or a BIOS that's adapted to work with IDE drives. From the BIOS' point of view, IDE drives behave like normal ST506 controllers, making it easy to integrate them into existing systems.

New standards are being defined for IDE drives. In the near future we'll most likely see new BIOS systems supporting IDE drives directly. Then PC makers can fully use the extended features offered by these drives, which go unused in many of today's systems.

From controller to memory

Regardless of the speed of the hard drive and the controller, the way in which the data is transferred by the controller to the memory determines the effective speed of a controller-hard drive combination. Four different methods can be used to do this:

- Programmed I/O (PIO)
- DMA
- Memory Mapped I/O
- Busmaster DMA

Programmed I/O

With programmed I/O, the different controller I/O ports manage both the drive commands and the transfer of the data between the controller and the main memory. If you use programmed I/O, you'll use the IN and OUT assembly language instructions. This means that every byte or word must be channeled through the CPU.

Here, the data transfer rate is limited by the speed of the PC bus and the performance of the CPU. While the ISA bus allows a maximum transfer rate of 5.33 Meg per second (16-bit rate), this rate is unattainable with any of today's CPUs. With fast 386es, 486es or Pentiums, the data transfer rate is limited to about 3 or 4 Meg and can be attained only with very fast and expensive hard drives.

Memory Mapped I/O

The CPU can process data from a disk controller even faster if it stores them in a fixed memory region. The segment located above the video RAM is generally used for this purpose. Data in a program's memory area can be transferred faster using MOV instructions. This is faster than accessing the I/O ports with IN and OUT.

Even by using memory mapped I/O, today's speedy CPUs can request data faster than the controller is capable of transferring. The controller can never reach the theoretical maximum value of 8 Meg per second. It can't even reach 5 to 6 Meg per second.

DMA

DMA (Direct Memory Access) transfer is more widely known than the first two methods. Using DMA, a device (hard drive, floppy disk drive, CD-ROM, etc.) can transfer data directly to the computer's memory. The CPU is bypassed. To use DMA, a program only needs to tell the DMA controller how many bytes should be transferred from one location to another. This makes DMA seem like the best method for transferring data.

However, the DMA controller in the PC is inflexible and slow. In fact, it's so slow that Programmed I/O is faster on 386 and higher systems. The DMA controller operates at 4 MHz on the AT or later systems even though it worked at 4.77 MHz on earlier PCs. So, DMA transfer is faster than Programmed I/O only on PCs. Using DMA, the data transfer rate is limited to about 2 Meg per second.

Therefore, the DMA method is no longer used on most modern hard drives, even though many hard drives do support this method in addition to Programmed I/O.

Busmaster DMA

Busmaster DMA is another form of direct memory access, but isn't related to the DMA circuitry on the motherboard of the computer. Using this method, the hard drive controller disconnects the CPU from the bus and transfers data to memory on its own using its own Busmaster DMA controller. Transfer rates of up to 8 Meg per second are possible. Unfortunately, this feature increases the price of the controller. Busmaster DMA is generally used only with very powerful SCSI controllers.

Protected mode

The methods of DMA transfer are limited to real mode programs. Without special intervention, DMA cannot be used in protected or virtual mode. How virtual memory management is performed by the CPUs memory management unit (MMU) is responsible for this limitation. The MMU maps a program's virtual memory addresses into real physical memory addresses.

A protected or virtual mode program doesn't realize this, because it's never aware of the physical addresses; it works only with virtual addresses. When this program wants to perform a DMA transfer, it passes a virtual address to the DMA chip. However, because the MMU isn't involved in the DMA transfer, it cannot map this virtual address into a physical address. As a result, the data is transferred to a different memory area. The system will soon crash because important memory areas are overwritten.

This is a characteristic of protected mode operating systems, such as Windows and OS/2. However, this also occurs in Virtual 86 mode using DOS if the EMM386.EXE device driver is used to emulate expanded memory. EMM386.EXE depends on the virtual memory management of the processor.

The solution is to "watch" the DMA controller. Do this in protected mode to control all the I/O ports. In Windows, for example, a virtual control monitor in the background is installed to watch the programming of the DMA controller via the BIOS or another program. This monitor converts the actual physical addresses before they are written to the register of the DMA controller.

Recording Information On The Hard Drive

To understand how information is written on a hard drive, you must first forget the concept of binary coding. Zeros and ones aren't stored on the magnetic surface of a hard drive. It's impossible to represent these two states as "magnetized" and "not magnetized".

Why isn't this possible? If you try to represent data as sequences of magnetized and non-magnetized particles, the read head of the hard drive wouldn't be able to keep the individual magnetic particles separate. So, it wouldn't be able to distinguish between three or five zeros.

One way to avoid this problem is by knowing the length of a magnetic particle and the elapsed time for each magnetic signal. In other words, you need a kind of clock that indicates, with each tick, that it's time for a new bit.

However, the constant period of time for a cycle cannot be clearly defined because of various factors. For example, the rotational speed of the hard drive may vary slightly. But a bigger factor is that a single magnetic particle can never be magnetized; only a group of magnetic particles, whose number isn't always constant, can be magnetized.

It is possible, however, to record flux reversals, which are short passages between non-magnetized particles and magnetized particles. The reversals in flux create an electrical pulse in the hard drive's read head. This pulse is then passed to the electronic circuitry, where it is used to decode the stored information as zeros and ones.

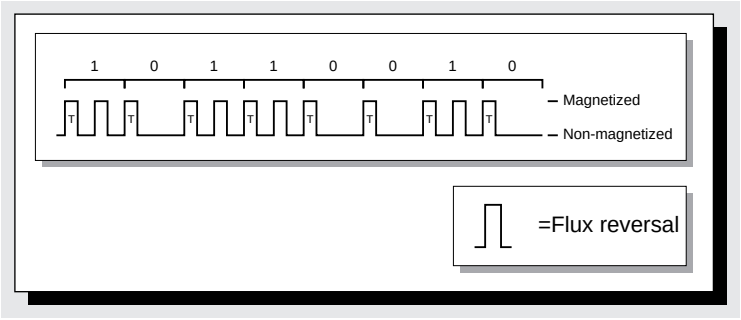
This coding and decoding of binary information has always been important to hardware developers The number of flux reversals that can be recorded per square inch on a hard drive is limited. This limit depends on the composition of the magnetic material, the gap of the read/write head, its sensitivity, etc. Anyone who can find ways to record more zeros and ones on the hard drive with an equal number of flux reversals will lead the competition in higher and higher disk capacities.

FM method

The simplest way to encode zeros and ones to a magnetic surface is to record a flux reversal for each one-bit and omit a flux reversal for each zero-bit. However, you'll encounter a problem when you want to record a long series of zeros. In this instance, you must omit many flux reversals to represent the string of zeros. This would confuse the controller that depends on flux reversals to keep in sync with the data on the hard drive.

To avoid this problem, a clock signal is written onto the drive along with the data. Using the FM method (Frequency Modulation) recording technique, a one-bit might be recorded as two consecutive flux reversals and a zero-bit recorded as a flux reversal followed by no flux reversal. In both cases, the initial flux reversal represents the timing signal. The data bit is then modulated "between" the timing signal (second flux reversal for one-bit and no flux reversal for zero-bit).

The FM method of recording



Although this method is simple and inexpensive, it has one major disadvantage. Each data bit requires two flux reversals, which reduces the potential disk capacity by half.

MFM method

To reduce the number of flux reversals of the FM method and thereby increase the density at which information can be recorded, another encoding technique is used. This technique is called MFM method (Modified FM). Basically, the data is encoded as follows:

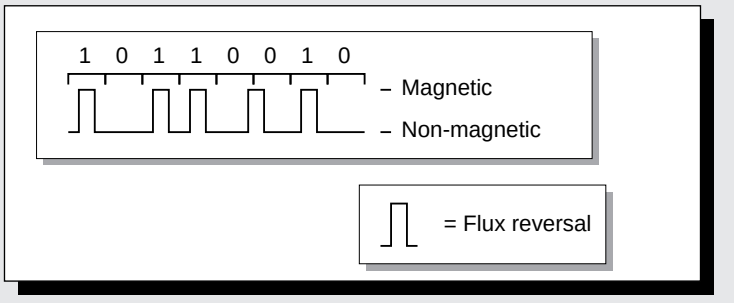
Code table for the MFM

Data bit value	Encoded as:
1	Flux reversal
0 following another 0-bit	Flux reversal followed by no flux reversal
0 following another 1-bit	No flux reversal followed by no flux reversal

With this method, the timing signal is also used to store data. Both zero-bits and one-bits are recorded using only a single flux reversal. Longer sequences of zeros and ones appear as a continuous sequence of flux reversals. This encoding method requires improved control circuitry so the hard drive can synchronize with the normal sequence of timing flux reversals for longer sequences of zeros. The only problem is when a one is followed by a zero, which requires a flux reversal to the normal timing position. The time between the flux reversal of the one and that of the zero amounts to only half the normal interval between two flux reversals. However, this won't work because the shortest interval between two flux reversals cannot be shorter; otherwise the electronic circuitry would no longer be able to keep up. Therefore, a flux reversal isn't stored for a zero that follows a one.

The next flux reversal comes after one and a half times the time for a flux change (bit combination 100b) or even after twice the time (bit combination 101b). The following illustration demonstrates this.

The MFM method of recording



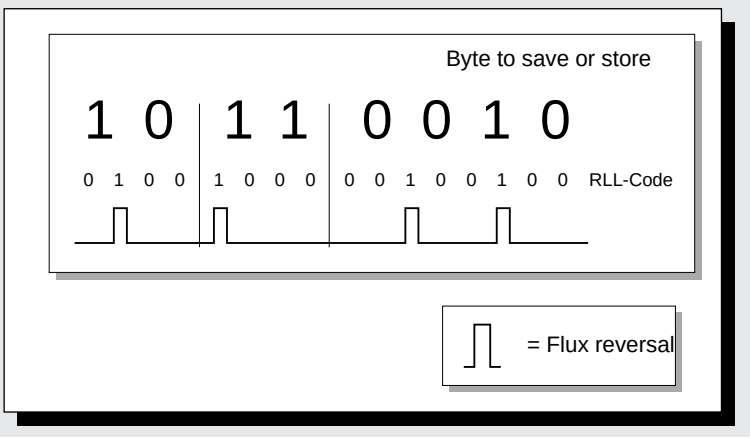
RLL method

Another encoding method, called RLL method (Run Length Limited), packs up to 50 percent more information on the disk than MFM.

In RLL, ones are stored as flux reversals; zeros are stored as the absence of flux reversals. A timing signal isn't recorded. The hard drive circuitry itself supplies the timing reference. However, even with a very constant rotation of the hard drive and improvements in the read head and its electronic circuitry, this is possible only if there aren't too many zeros between two ones. With each zero, the time to the next flux reversal increases, and along with it, the possibility the hard drive controller will lose its beat.

However, the ones cannot follow each other too rapidly. Otherwise, the controller may not be able to keep pace.

RLL encoding method of recording



Instead of encoding a single bit, the RLL method looks at and encodes a group of data bits. This group or bit string is from 2 to 4 bits in length. The new encoded string is twice as long as the original, but ensures the sequences of zeros aren't too long and the distances between ones don't become too short (and thereby overwork the circuitry). The 2,7 RLL encoding scheme is today's standard and is used in most modern hard drives. In the encoded string, a minimum of 2 and a maximum of 7 zeros will appear between two ones. This method increases the capacity of a drive by about 50 percent compared to MFM.

Another scheme is 3,9 RLL, which is called "advanced RLL", lets you fit even more information on the drive. With this method, the encoded scheme has a minimum of 3 and a maximum of 9 zeros between two ones.

The table on the right shows the encoding scheme for RLL 2.7. At first, you may think that a byte such as 00000001b cannot be encoded. However, don't forget that, on this level, you're working with sectors instead of single bytes. So, it's even possible to encode a byte, such as 00000001b, by including the bits of the following byte in the coding.

Code table for the RLL 2.7			
Bit pattern	Encoded as	Bit pattern	Encoded as
000	000100	11	1000
10	0100	011	001000
010	100100	001	00001000
0010	00100100		

The only problem in coding occurs with the last byte of a sector because this method needs the following byte. This problem is solved by simply using a byte from the hard drive controller. The excess bits are simply truncated so the last byte of a sector is correctly decoded.

Hard Drive Advancements

One of the remarkable achievements in PC computers has been the "smaller, faster, cheaper" advancements in hard drive technology. For instance:

- Access times in the top performance models have decreased from over thirty milliseconds to about ten milliseconds.
- Increases in maximum storage capacities. Hard drives with a capacity of more than 1 Gigabyte are now common.
- 1 Meg of hard drive capacity costs less than ten dollars.

The hard drive's performance and capabilities has increased because of the optimization in recording and playback techniques. In this section we'll discuss some of these techniques.

Interleaving in the first XT and AT hard drives from IBM			
AT		XT	
Physical sectors	Logical sectors	Physical sectors	Logical sectors
1	1	1	1
2	7	2	4
3	13	3	7
4	2	4	10
5	8	5	13
6	14	6	16
7	3	7	2
8	9	8	5
9	15	9	8
10	4	10	11
11	10	11	14
12	16	12	17
13	5	13	3
14	11	14	6
15	17	15	9
16	6	16	12
17	12	17	15

Interleaving and the interleave factor

Today hard drive controllers are so fast they can read an entire track even when only the data in a single sector is requested. So, when the data in the next sector is needed, the hard drive controller doesn't have to read the next sector. Instead, it can take the data from the controller's internal buffer. This significantly speeds up data access.

Most ST506 controllers have only an internal sector buffer (with a capacity to store the contents of exactly one sector), so only the newer controller types (SCSI, ESDI, and IDE) are track buffer compatible. Also, the sector buffer cannot be reused until the last byte of the previous sector is passed to the CPU. So, additional time is required until the next sector passes under the read head. The data in the subsequent sector cannot be read until the sector passes under the read head again; this requires almost an entire revolution of the hard drive. This process continues with each sector, noticeably reducing the speed of disk access. To minimize this delay, interleaving is used to spread the logical sectors across the track.

By interleaving the logical sectors, the controller has enough time to process and transfer the data in one sector before the next logical sector passes under the read head. This avoids the rotational delay of having to wait for a complete revolution before the sector can be read.

Interleaving is measured by the interleave factor. This value is the number of sectors by which the logical sector number was shifted, compared to the physical sector number. The original XT hard drive uses an interleave factor of 1:6, while the AT

uses an interleave factor of 1:3. However, the interleave factor of the AT hard drive can be reduced to 1:2, which increases the access speed. Today, interleave factors of 1:1 are common, which means there is no interleaving at all.

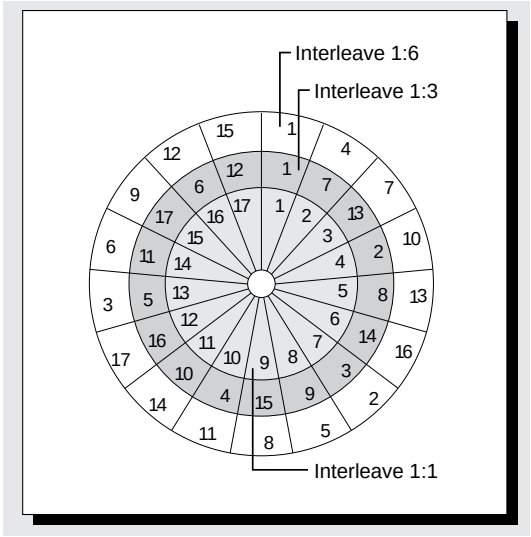
However, an interleave of 1:6 requires that five physical sectors be skipped before the next logical sector can be read. An interleave of 1:3 requests that two physical sectors be skipped. The table on the previous page shows the logical arrangement of sectors on a track with 17 sectors.

Despite the advantages of this method, the best interleave is actually no interleave at all. Without an interleave, an entire track be read within a hard drive revolution. However, two, three, or even more revolutions would be needed with interleaving, depending on the interleave factor.

Setting the interleave

Set the interleave in the "low-level-format" of the hard drive, which creates the address labels and sector numbers on the surface on an unused hard drive.

Different interleave factors and their influence on the formatting of a track



Because the logical sector number of a single sector is determined by the low-level-formatting program, it's possible to shift the sectors. Many hard disk utilities will prompt you for the desired interleave factor. If you select a "bad" value, your hard drive may slow down when loading programs or accessing files.

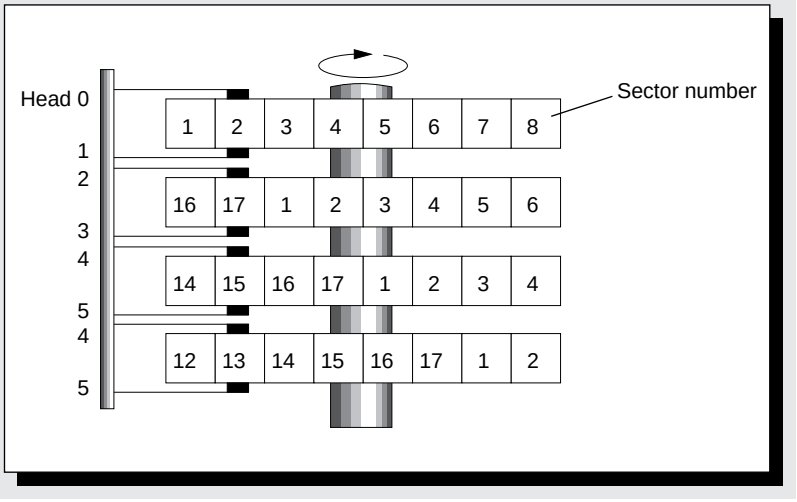
Even if you do select a "bad" interleave factor, you can still eliminate the problem later. You can use a program, such as the Norton Utilities, to do this. Such programs determine the optimum interleave factor and then execute the appropriate low-level-format for you. This usually takes just a few minutes.

You may think that this type of utility will lead to data loss because the program is "tampering" with the hard drive. However, as long as the program is error-free, you shouldn't loose any data. Since these programs work below DOS level, DOS is completely unaffected by the changes. Remember, DOS doesn't know the origin of a sector which it reads or writes. So, as long as the data are read before a track is reformatted and written back, DOS believes that nothing has changed.

Track and cylinder skewing

Carefully selecting an interleaving factor significantly increases hard disk performance. After the data from one track is read, data, which is in same cylinder but the following head, are usually accessed. As the controller switches the read/write head, the hard drive continues to rotate. After reading the last sector on the track, the first sector on the next head has already passed the read/write head. So you must wait almost an entire revolution.

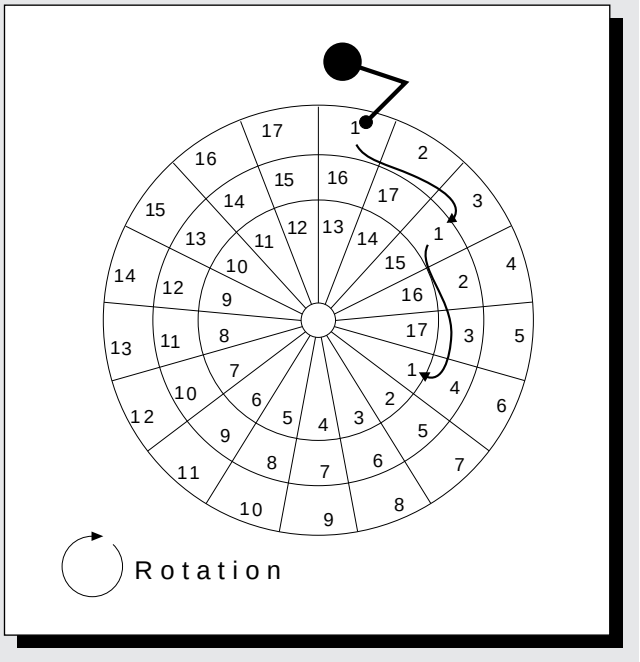
Cylinder skewing



To prevent this from happening, use cylinder skewing. All of the sectors on a track are shifted so, after switching to the next head, the first sector can be read without any rotational delay. You can set the cylinder skewing during the low-level-formatting of the drive.

In addition to cylinder skewing, there is also track skewing. This is similar to cylinder skewing, except that it considers the time needed by the drive to move the entire read/write arm to the next track.

Track skewing with an interleave of 1:1



Multiple zone recording

One way to increase the capacity of a hard drive is to format more sectors on the outer tracks. Since they have a larger circumference, the outer tracks have more space than the inner tracks. With ST506 controllers, the number of sectors per track was fixed and limited by the capacity of the innermost (shortest) track.

With modern SCSI and IDE disks, which only simulate the number of heads, tracks, and sectors, it's possible to format individual tracks with a different number of sectors. These controllers and drives translate a logical head, cylinder, and sector number to a physical head, cylinder, and sector number. So, it's possible to vary the number of sectors on outer tracks.

This places a greater demand on the controller circuitry and the read/write head. In the same (rotational) time, more sectors must be read or written to the outer tracks than on the inner track. So, while the length of the flux reversal decreases, the number of data bits to be read or written increases.

This can increase the capacity of a hard drive by between 20 and 50 percent compared to the usual, fixed sector per track arrangement.

Error correction

One important indicator of hard drive performance is its reliability. Impurities in the magnetic material are unavoidable and cause some sections of the hard drive to be unusable.

In the past, a list of defective sectors were printed on the hard drive case as they were detected by the manufacturer. During the low-level format, the user entered these sector numbers so the operating system would recognize these defective areas. Then the operating system would mark them as defective in the FAT (File Allocation Table) and avoid using them for storing data.

Newer hard drives, especially IDE models, no longer have a defective sector list. Either the sectors are already recorded on a separate data track by the manufacturer or they are recognized during the low-level format. These defective sectors are either skipped or replaced by other sectors from "alternate" areas. The tables that identify the defective sectors and their alternates are passed to the hard drive controller during initialization. When the hard drive controller tries to access a sector identified as defective, it can switch to the alternate sector. Although this slows access to such a sector, since there are very few defective sectors, this is hardly noticeable.

Many drives are also able to recognize defective sectors during read and write operations. This is the purpose of the ECC (Error Correction Code) that is automatically generated and saved with each sector. If an error is encountered during an operation, in many cases the data is reconstructed, an alternate sector is assigned, and the data is rewritten to this alternate sector. This feature is found in most of the IDE drives.

Other hard drive components

A hard drive contains more than just the data bit information. When a sector is formatted, an entire set of information is also written so the hard drive correctly recognizes and reads data. Although the format of this information is different depending on the controller, the information itself is the same (e.g., the cylinder number, the sector number, and the error correction code).

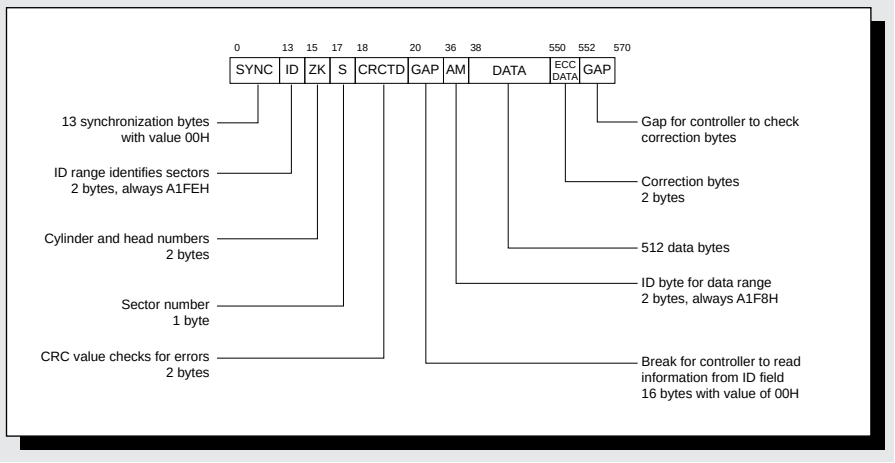
Each sector begins with a series of thirteen synchronization bytes with the value 00H (SYNC field). This bit pattern results in a constant series of flux reversals that help the controller synchronize itself to the sector.

Following this is the ID field, which identifies the sector. The cylinder, head, and sector numbers are recorded here (CH/S fields). This information starts with a special ID byte (ID field) and ends with a two byte error code that's used to check the validity of the information (CRCID field).

Next is a gap giving the controller a break before the start of the data field. The controller needs this break to read the information from the ID field to determine whether this is the desired sector. The gap also lets the controller resynchronize itself. The actual data field follows. The data field also has an ID byte (AM field) at the start, which is for identification. The 512 data bytes (DATA field) are next. The data field ends with two correction bytes (ECC-DATA field) the controller uses to determine whether the data is valid.

The sector ends with another gap to give the controller time to check the correction bytes. Here the gap contains bytes with the value 4EH. So the sector's total length is 570 bytes.

Sector format created by an ST506-type controller



Extra tracks

A hard drive also contains other "reserved" areas. For example, there are servo tracks, which are used by the hard drive's electronic circuitry to synchronize itself to the data clock. This is especially important when used with RLL controllers and drives.

Some drives also contain "reserved" or "alternate" tracks. These tracks are unknown even to the BIOS. The area on a reserved track is used as a replacement for defective sectors.

Finally, there is the park area. When the computer is switched off, the read/write heads "land" or settle on the surface of the park area. If the read/write head lands on a track containing data, that data might be destroyed or damaged. The park area contains no data and is a safe area that cannot be scratched or otherwise damaged by the read/write heads.

Access times

A hard drive's performance is measured by its access speed. Hardware manufacturers often claim their hard drives have access speeds of 10 or 30 milliseconds. This value represents the average seek time between two file accesses.

Most manufacturers also use track-to-track seek time and maximum seek time to express hard drive speed. The track-to-track seek time is the time needed to move the read/write head from one cylinder to the next. The maximum seek time is the time needed to move the read/write arm from the first cylinder to the last cylinder of the drive.

Although these factors have an important effect on how a hard drive accesses a file at the DOS level, there are other important factors. For example, a hard drive with extremely fast specification is useless unless the controller can keep pace with it. Also, a hard drive could be so fragmented the read/write arm is constantly jumping back and forth between different cylinders to access data. So, performance must also be measured according to the time that a read or write request spends at the various levels of the application program, at the DOS level with its device drivers, the BIOS, and any special hard drive drivers up to and including the programming of the hard drive controller.

To verify the performance data of a hard drive manufacturer, you can use a performance measuring program, such as SystemInfo from Norton Utilities or CORETEST from Core International.

Generally these programs measure a hard drive's data transfer rate by reading the largest possible data block from the hard drive. However, the size of this block is restricted to one complete cylinder, which means the read/write arm doesn't move during operation. We already know that ESDI, SCSI, and IDE controllers often "hide" the true specifications from the BIOS and instead translate the drive's cylinder, head, and sector values into physical ones. So, the results of these tests may not

be accurate. A controller translation that results in a track change is also accompanied by a track-to-track delay, which distorts the result of the measurements.

When measuring the track-to-track seek time, you'll encounter a similar problem. To do this, read two sectors in adjacent tracks. However, the controller translation of these two sectors may not result in a track change, which produces an unrealistic zero track-to-track seek time.

Cache controllers and cache programs

The results of these measurements are especially suspicious when cache programs are used. At the software level, disk cache programs, such as SMARTDRV or PCKwik, can have a major effect on disk drive performance. These cache programs hook into the BIOS hard drive interrupt and intercept the read and write calls of the application programs and the device drivers of DOS.

When an application program wants to read data from a hard drive, the cache program intercepts the read request , passes the read request to the hard drive controller in the usual way, saves the data that was read in its cache buffer, and then passes the data back to the application program.

Depending on the size of the cache buffer, numerous sectors are read into and saved in the buffer. When the application wants to read more data, the cache program again intercepts the request and examines its buffers to see if the data is still in the cache. If it is, the data is immediately passed back to the application without another hard drive operation. As you can imagine, this speeds up access tremendously and can greatly affect the disk drive performance measurements.

In order to maintain accurate measurements, you can write a program to check the BIOS disk interrupt 13H. If it's still pointing to the ROM BIOS, then a cache program isn't active. If it's not, then you can request the user remove or disable the cache program until the measurements are completed.

Another type of disk cache is one that is part of the hard drive controller. This is a hardware disk cache and doesn't use any BIOS interrupts. Instead, the caching is performed at the hardware level and is invisible to normal performance measurement software.

Hard Drive Partitions

If you've ever installed a hard drive or added an operating system, such as XENIX or OS/2, then you've used the DOS FDISK command. This command is used to partition the hard drive. A hard drive must be partitioned when you want to logically divide the hard drive into separate volumes or when you want to install more than one operating system on the same hard drive.

Formatting process

To prepare a hard disk to be used by an operating system, you must perform three tasks. First you must perform a low-level format. When you do this, you are organizing the drive into cylinders, tracks, and sectors by writing the appropriate address markers onto the drive surface. The address markers are later used by the controller to identify the specific sectors.

In the early days of the PC, the DOS DEBUG command was used to perform a low-level format. Today, low-level formatting is no longer complicated because most hard drive manufacturers provide programs that perform this task.

Partitioning the hard drive

Next, you must partition the hard drive. There are two reasons why this is necessary. First, it enables you to install multiple operating systems, such as DOS and XENIX, on a single drive. Partitioning a hard drive into separate areas lets each of the operating systems manage its disk space without the conflicts caused by different file structures.

The other reason for partitioning a hard drive is to be able to use the additional capacity of larger drives. The original XT had a hard drive with a 10 Meg capacity. Then drives with 40 and 80 Meg capacities appeared. Early versions of DOS were able to manage hard drive capacities of only 32 Meg. But this limitation was addressed by DOS 3.3. Now the 32 Meg maximum capacity applied only to a partition. DOS 3.3 allows one primary partition with a 32 Meg maximum and an extended partition

that can be divided into as many as 23 logical devices (drives C: through Z:). Each of these logical devices can hold up to 32 Meg, which makes the entire hard drive capacity 768 Meg.

DOS 4.0 goes even farther, supporting drives with a maximum capacity up to 2 Gigabytes. Nevertheless, many users continue to partition the hard drive into logical drives, because they would rather work with multiple drives than one large drive and several hundred or even several thousand files.

While the primary partition must be located within the first 32 Meg of the hard drive, the extended partition can be located anywhere. FDISK refers to these partitions as "PRI DOS" and "EXT DOS".

Partition sector

The partition sector is the structure that all versions of DOS use to define a hard drive's partitions. When you run the FDISK program for the first time, it creates the partition sector in the hard drive's first sector (cylinder 0, head 0, sector 1).

The BIOS first loads the partition sector, instead of the DOS boot sector, after the system is started or reset. The partition sector is loaded into memory at address 0000:7C00, if there are no diskettes in drive A:.

If the BIOS finds the values 55H, AAH, in the last two bytes of the partition sector's 512 total bytes, it considers the sector to be executable and starts executing the program at the first byte of the sector. Otherwise, the BIOS displays an error message and either starts ROM-BASIC or goes into a continuous loop, depending on the version of BIOS and the manufacturer.

Structure of the partition sector of a hard drive

Address	Contents	Type
+000h	Partition code	Code
+1BEh	1.Entry in the partition table	16 BYTE
+1CEh	2.Entry in the partition table	16 BYTE
+1DEh	3.Entry in the partition table	16 BYTE
+1EEh	4.Entry in the partition table	16 BYTE
+1FEh	IDcode(AA55h),which identifies the partition sector as such	2 BYTE

Length: 1EH (30) bytes

This program recognizes and starts the active partition's operating system. To do this, it must load the operating system's boot sector and pass control to the program within that boot sector. Since the later's program code must also be loaded at memory address 0000:7C00, the code from the partition sector is moved to the memory address 0000:0600 first, to make room for the boot sector.

Partition table

The program in the partition sector must be able to find the boot sector for the active partition. For this, it uses the partition table. This table is located at offset 1BEH of the partition sector.

Each entry in the partition table is 16 bytes. The table is located at the end of the partition sector, leaving enough room for 4 entries. So the number of partitions is limited to 4. To accommodate more than four partitions, some hard drive manufacturers use a special configuration program that relocates and enlarges the partition table and adapts the partition sector code to use this relocated table.

Sometimes the partition sector code is changed to allow you to boot any of the installed operating systems on the hard drive. This makes it easy to choose which of the operating systems should run when the computer is first started.

The partition table has the following layout:

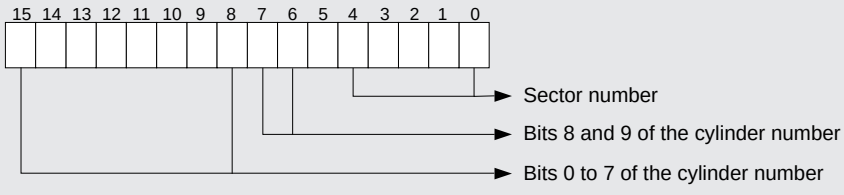
Structure of an entry in the partition table		
Address	Contents	Type
+00h	Partition status 00h = inactive 80h = Boot-Partition	1 BYTE
+01h	Read/write head, with which the partition begins	1 BYTE
+02h	Sector and cylinder, with which the partition begins	1 WORD
+04h	Partition type * 00h = Entry not allocated 01h = DOS with 12-Bit-FAT (primary Part.) 02h = XENIX 03h = XENIX 04h = DOS with 16-Bit-FAT (primary Part.) 05h = extended DOS-Partition (DOS 3.3) 06h = DOS-4.0 partition with more than 32 Meg DBh = Concurrent DOS	1 BYTE
+05h	Read/write head, with which the partition ends	1 BYTE
+06h	Sector and cylinder, with which the partition ends	1 WORD
+08h	Removal of first sector of the partition (Boot-sector) of partition sector in sectors	1 DWORD
+0Ch	Number of sectors in this partition	1 DWORD
Length: 10h (16 Bytes)		
* Other codes possible combined with other operating systems or special driver software.		

Starting the boot partition

The first field of each partition table entry indicates whether a partition is active. A value of 00H indicates that partition isn't active; a value 80H indicates that partition is active and should be booted. If the partition sector program detects that more than one partition is active or that none of the partitions are active, it aborts the booting process, displays an error message, and waits in a continuous loop. You can exit this loop only by resetting.

When the partition sector program recognizes the active partition, is uses the next two fields to determine the location of this partition on the hard drive. The sector and cylinder numbers are expressed exactly as BIOS interrupt 13H (Diskette/Hard drive), including bits 6 and 7 of the sector number, which represent bits 8 and 9 of the cylinder number. At this point, BIOS interrupt 13H and its functions are the only way to access the hard drive. The DOS functions aren't available because DOS hasn't been booted yet.

Coding of the cylinder and sector number within the fields in the partition sector



Other information in the partition table

The partition table also contains additional information. For example, each entry has a field that describes the operating system for that partition. The above figure shows the types that are supported.

In addition to the partition's starting sector, another field contains the partition's ending sector, expressed as cylinder, head, and sector numbers.

There are two additional fields for each table entry. The first is the total number of sectors within the partition. The last is the distance of a partition's boot sector from the partition sector, counted in sectors.

Remember the first partition on a drive usually begins in the first sector of track 0, head 1. In other words, almost an entire track is "wasted" because the partition sector occupies only one sector on track 0, head 0.

Structure of the extended partition under DOS

DOS 3.3 allows you to define one primary and one extended partition on a hard drive. FDISK builds the partition sector and partition table to identify the partitions, but doesn't write the program code into the partition sector.

The partition table has two entries. The first entry is for the first logical device of the extended partition and a partition type (value 1 or 4 that indicates a DOS partition with 12-bit or 16-bit FAT). The second entry is for the next logical device within the extended partition, is one exists.

To support other logical devices, this structure is repeated for each additional device. This results in a chained list that continues until the "partition type" field, in the second table entry of the partition table of the partition sector of an extended partition, contains has a zero value.

Sample programs to examine the partition structure

You can use the FIXPARTP.PAS, FIXPARTC.C and FIXPARTB.BAS programs to examine the partition structures of a hard drive. FIXPARTP.PAS is written in Pascal and FIXPARTC.C in C. The programs display the contents of the partition sectors and the extended partitions (if there are any) from the hard drive. By default, they access the first hard drive, number 0, but you can also specify a different number (1, 2, 3, etc.) when running the programs to examine a different hard drive.

You'll find the following program(s) on the companion CD-ROM

FIXPARTP.PAS (Pascal listing)
FIXPARTC.C (C listing)
FIXPARTB.BAS (BASIC listing)