

Assembly Language for Intel-Based Computers, 5th Edition

Kip R. Irvine

Chapter 14: Disk Fundamentals

Slide show prepared by the author

Revision date: June 4, 2006

(c) Pearson Education, 2006-2007. All rights reserved. You may modify and copy this slide show for your personal use, or for use in the classroom, as long as this copyright statement, the author's name, and the title are not changed.

Chapter Overview

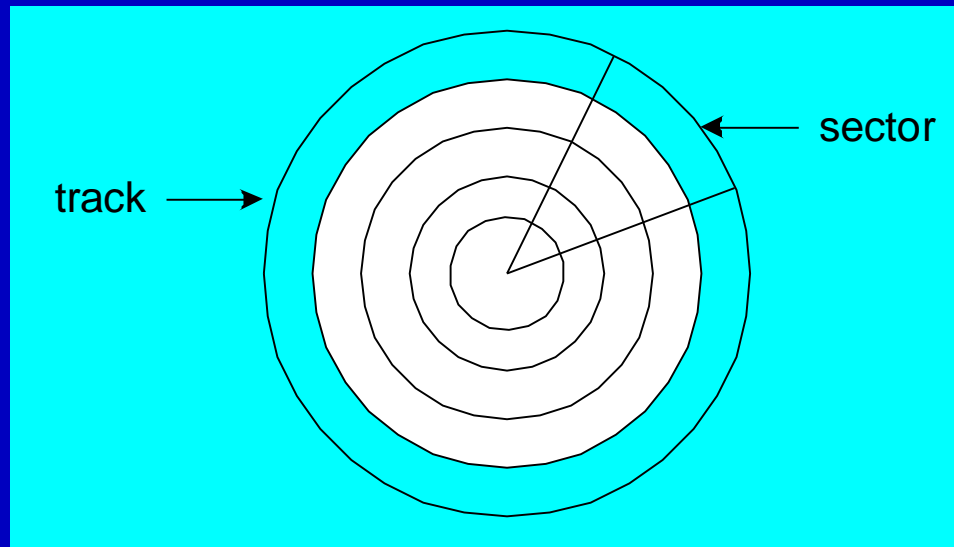
- **Disk Storage Systems**
- File Systems
- Disk Directory
- Reading and Writing Disk Sectors (7305h)
- System-Level File Functions

Disk Storage Systems

- Tracks, Cylinders, and Sectors
- Disk Partitions (Volumes)

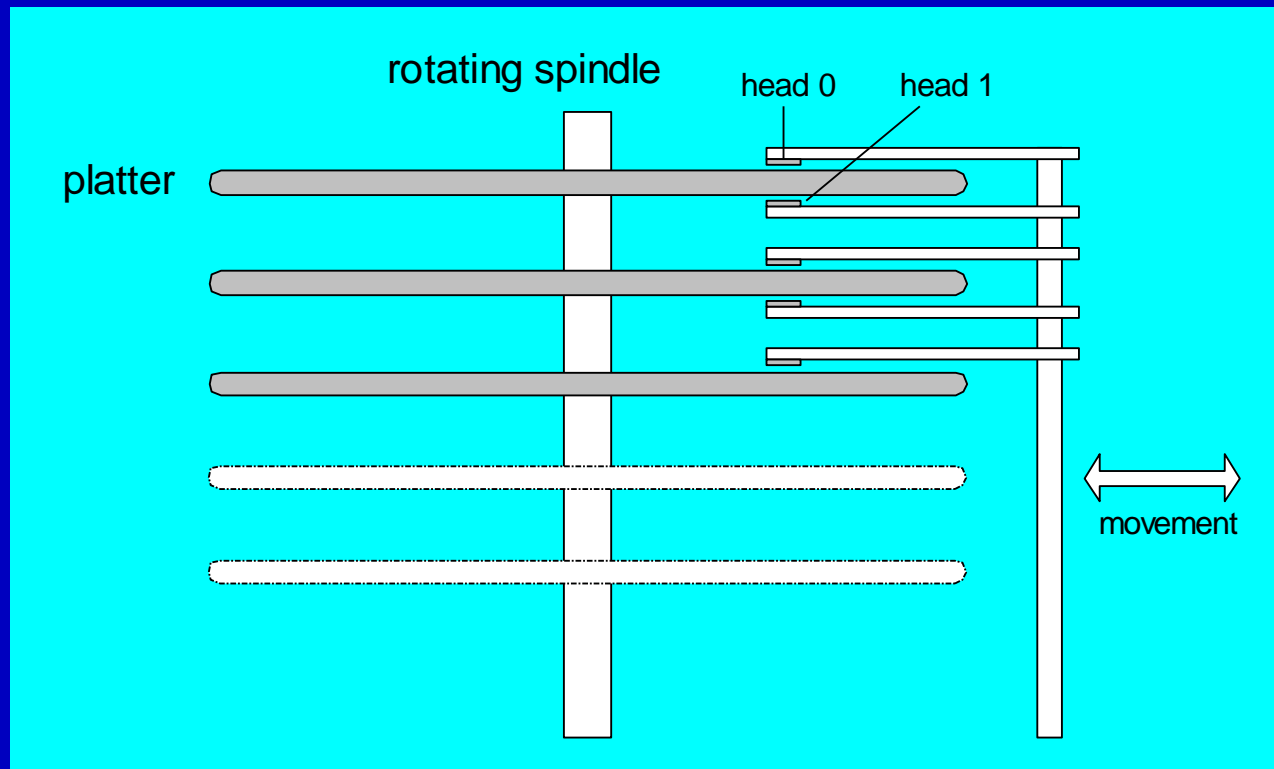
Tracks and Sectors

- **Physical disk geometry** - a way of describing the disk's structure to make it readable by the system BIOS
- **Track** - concentric circle containing data
- **Sector** - part of a track



Cylinders and Seeking

- **Cylinder** - all tracks readable from one head position
- **Seek** - move read/write heads between tracks

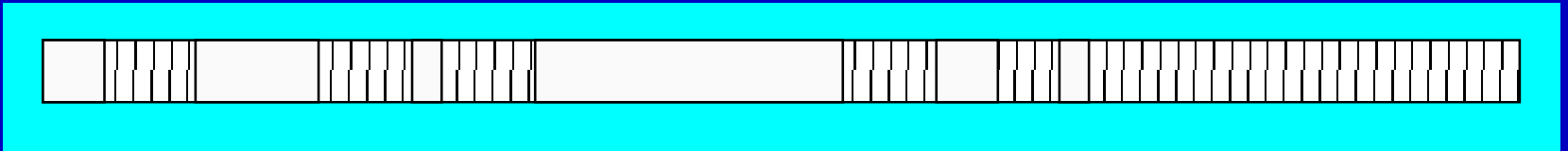


Disk Formatting

- Physical formatting
 - aka **low-level formatting**
 - Usually done at the factory.
 - Must be done before logical formatting
 - Defines the tracks, sectors, and cylinders
- Logical formatting
 - Permits disk to be accessed using sequentially numbered **logical sectors**
 - Installs a file system (ex: NTFS)
 - May install an operating system

Fragmentation

- A **fragmented** file is one whose sectors are no longer located in contiguous areas of the disk.
 - causes read/write heads to skip
 - slower file access
 - possible read/write errors



Translation

- Translation - conversion of physical disk geometry to a sequence of logical sector numbers
- Performed by a hard disk controller (firmware)
- **Logical sector numbers** are numbered sequentially, have no direct mapping to hardware

Disk Partitions

- Logical units that divide a physical hard disk
 - Also called **volumes**
- Primary partition
 - Up to four permitted
 - Each may boot a different OS
- Extended partition
 - Maximum of one permitted
 - May be divided into multiple **logical partitions**, each with a different drive letter
- Primary and Extended
 - Up to three primary and one extended

Logical Partitions

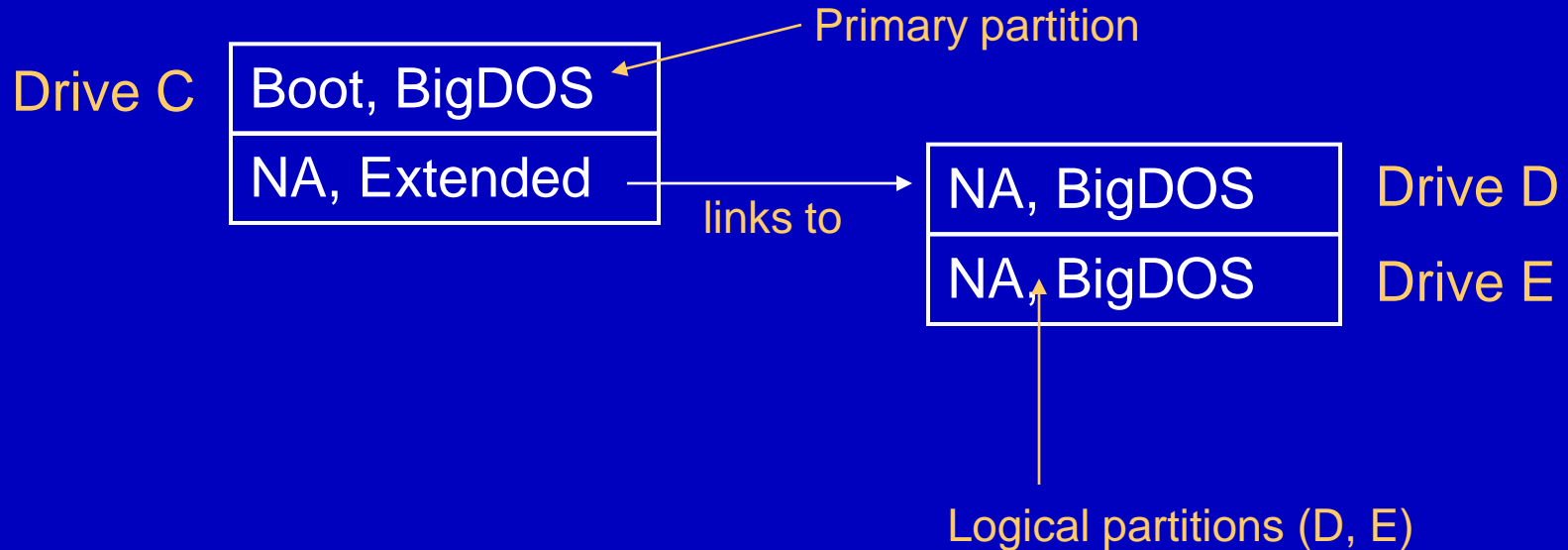
- Created from an extended partition
- No limit on the number
- Each has a separate drive letter
- Usually contain data
- Can be bootable (ex: Linux)

Disk Partition Table

- Located in the disk's Master Boot Record (MBR), following a block of executable code
- Four entries, one for each possible partition
- Each entry contains the following fields:
 - state (*non-active, bootable*)
 - type of partition (*BigDOS, Extended, . . .*)
 - beginning head, cylinder, & sector numbers
 - ending head, cylinder, & sector numbers
 - offset of partition from MBR
 - number of sectors in the partition

See also: www.datarescue.com/laboratory/partition.htm

Cascading Partition Tables



Boot = bootable (system)

NA = non active

BigDOS = over 32 MB

Dual-Boot Example

Volume	Layout	Type	File System	Status	Capacity	Free Space	% Free
	Partition	Basic		Healthy	5.13 GB	5.13 GB	100 %
	Partition	Basic		Healthy	2.01 GB	2.01 GB	100 %
BACKUP (E:)	Partition	Basic	FAT32	Healthy	7.80 GB	4.84 GB	62 %
DATA_1 (D:)	Partition	Basic	FAT32	Healthy	7.80 GB	2.66 GB	34 %
SYSTEM 98	Partition	Basic	FAT32	Healthy	1.95 GB	1.12 GB	57 %
WIN2000-A (C:)	Partition	Basic	NTFS	Healthy (System)	3.91 GB	1.43 GB	36 %
-	-	-	-	-	-	-	-

- **System 98** and **Win2000-A** are bootable partitions
 - One is called the **system partition** when active
- **DATA_1** and **BACKUP** are **logical partitions**
 - Their data can be shared by both operating systems

Master Boot Record (MBR)

- The MBR contains the following elements:
 - Disk partition table
 - A program that jumps to the boot sector of the system partition

What's Next

- Disk Storage Systems
- **File Systems**
- Disk Directory
- Reading and Writing Disk Sectors (7305h)
- System-Level File Functions

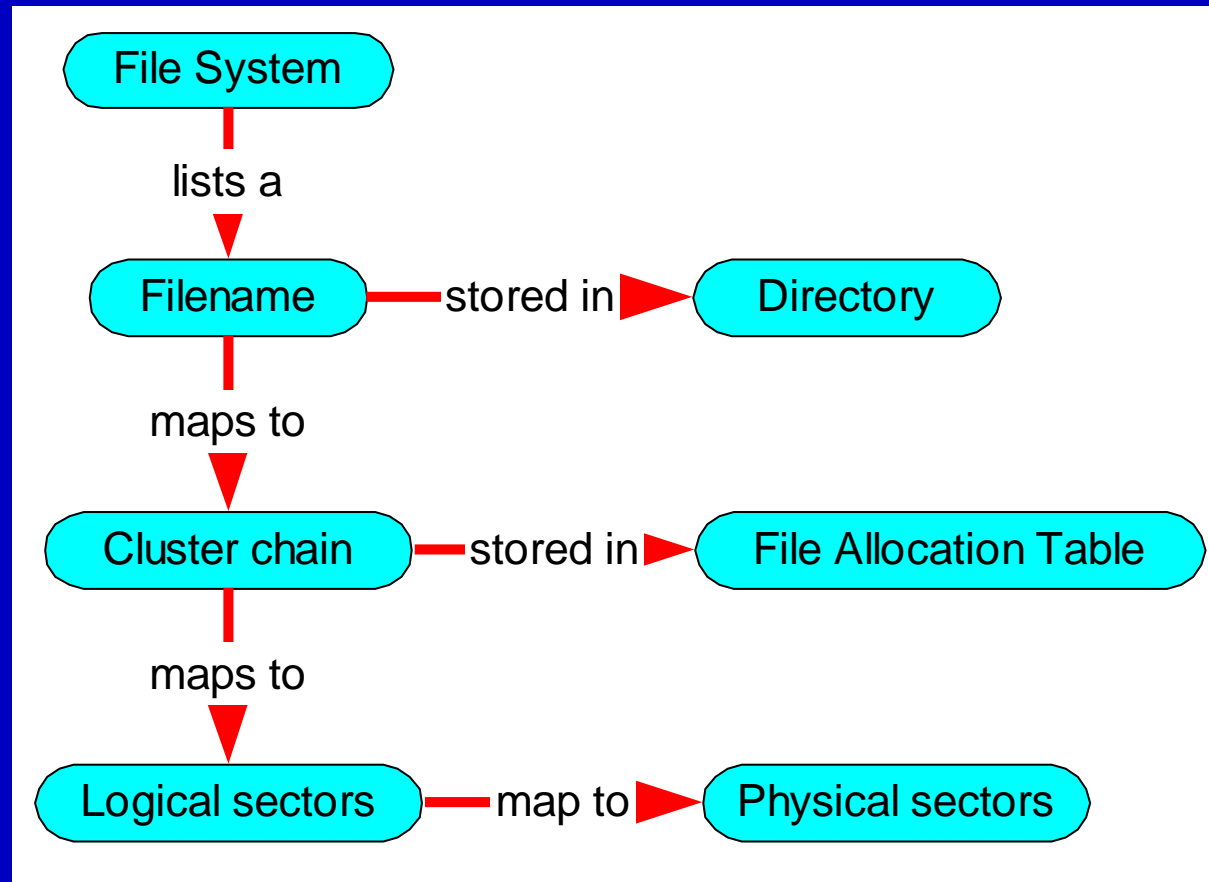
File Systems

- Directory, File, Cluster Relationships
- Clusters
- FAT12
- FAT16
- FAT32
- NTFS
- Primary Disk Areas

File System

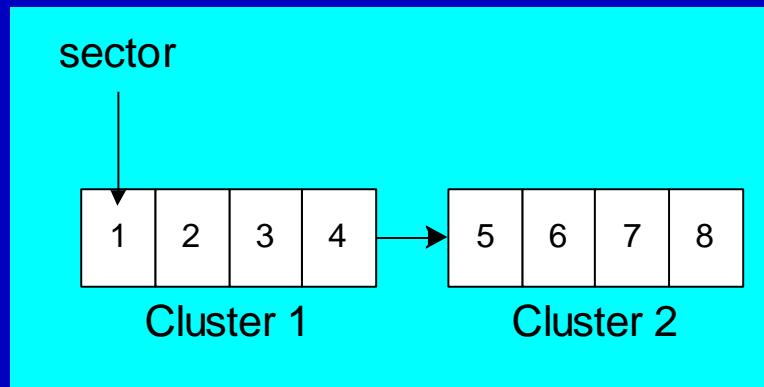
- This is what it does for you:
 - Keeps track of allocated and free space
 - Maintains directories and filenames
 - Tracks the sector location of each file and directory

Directory, File, Cluster, Sector Relationships



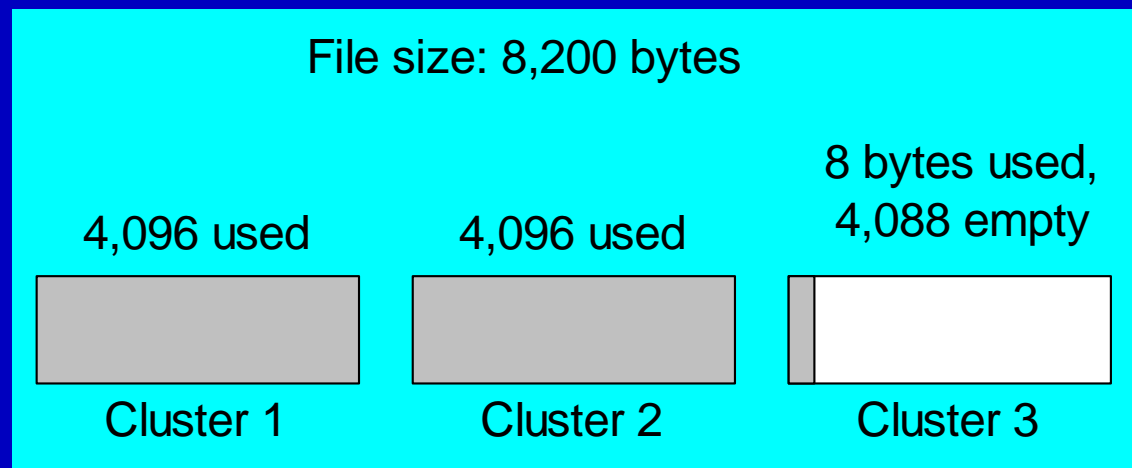
Cluster (1 of 2)

- Smallest unit of space used by a file
- Consists of one or more adjacent sectors
- Size depends on both the type of file system in use and the disk partition size
- A **file** is a linked sequence of clusters. Example:



Cluster (2 of 2)

- A file always uses at least one cluster
- A high percentage of space may be wasted
- Example: 8,200-byte file requires three 4K clusters:



FAT12

- Designed for diskettes
- Each FAT entry is 12 bits
- Very little fault tolerance
 - two copies of the FAT (cluster table)
- Optimal storage for small files
 - 512-byte clusters

FAT16

- MS-DOS format for hard disks
- 16-bit FAT entries
- Large cluster size when disk > 1 GB
 - inefficient for small files
- Max 2 GB size under MS-DOS
- Little or no recovery from read/write errors

FAT32

- Supports long filenames
- Supported by all version of MS-Windows from Windows 95 onward
 - (except Windows NT)
- 32-bit FAT entries
- 32 GB maximum volume size
- Improved recovery from read/write errors

NTFS

- Supported by Windows NT, 2000, and XP
- Handles large volumes
 - can span multiple hard drives
- Efficient cluster size (4K) on large volumes
- Unicode filenames
- Permissions on files & folders
- Share folders across network
- Built-in compression and encryption
- Track changes in a *change journal*
- Disk quotas for individuals or groups
- Robust error recovery
- Disk mirroring

Primary Disk Areas

- A disk or volume is divided into predefined areas and assigned specific logical sectors.
- Example: 1.44 MB diskette
 - Boot record (sector 0)
 - File allocation table (sectors 1 – 18)
 - Root directory (sectors 19 – 32)
 - Data area (sectors 33 – 2,879)

Your turn . . .

1. A 1.44 MB diskette has 512 bytes per cluster. Suppose a certain file begins in cluster number 5. Which logical disk sector contains the beginning of the file? (*Hint: see page 503*).
2. Suppose a certain hard drive has 4 KB per cluster, and we know that the data area begins in sector 100. If a particular file begins in cluster 10, which logical sectors are used by the cluster?

(answers on next panel . . .)

Answers

1. The data area begins in Sector 33 (see page 503). Each cluster = 1 sector, so the file begins in sector $33 + 5 =$ sector 38.
2. The hard drive has 8 sectors per cluster. The starting cluster number of the file is $100 + (8 * 10) = 180$. Therefore, sectors 180 – 187 are used by the file's first cluster.

Boot Record (1 of 2)

- Fields in a typical MS-DOS boot record:
 - Jump to boot code (JMP instruction)
 - Manufacturer name, version number
 - Bytes per sector
 - Sectors per cluster
 - Number of reserved sectors (preceding FAT #1)
 - Number of copies of FAT
 - Maximum number of root directory entries
 - Number of disk sectors for drives under 32 MB
 - Media descriptor byte
 - Size of FAT, in sectors
 - Sectors per track

Boot Record (2 of 2)

(continued)

- Number of drive heads
- Number of hidden sectors
- Number of disk sectors for drives over 32 MB
- Drive number (modified by MS-DOS)
- Reserved
- Extended boot signature (always 29h)
- Volume ID number (binary)
- Volume label
- File-system type (ASCII)
- Start of boot program and data

What's Next

- Disk Storage Systems
- File Systems
- **Disk Directory**
- Reading and Writing Disk Sectors (7305h)
- System-Level File Functions

Keeping Track of Files

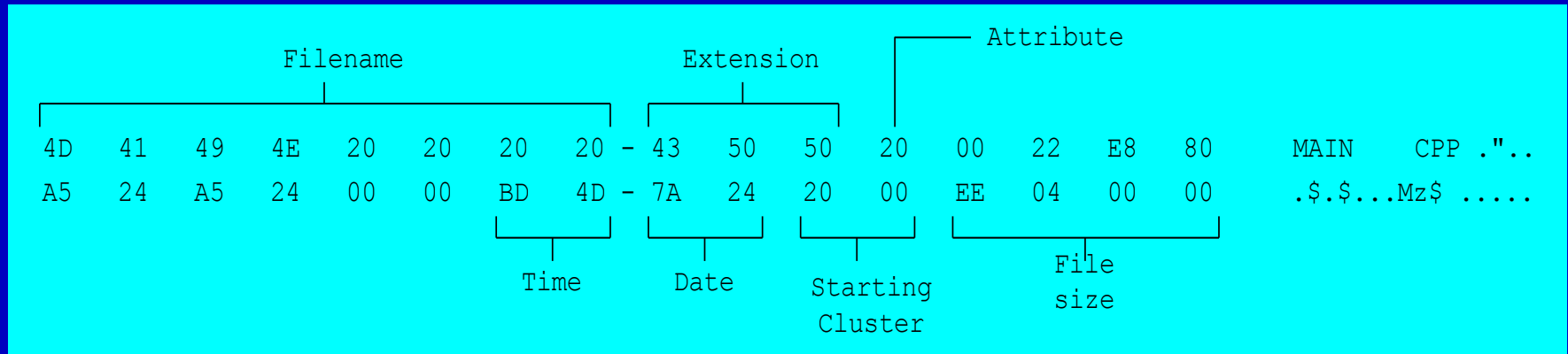
- MS-DOS Directory Structure
- Long Filenames in MS-Windows
- File Allocation Table

MS-DOS Directory Structure (1 of 2)

Hexadecimal Offset	Field Name	Format
00-07	Filename	ASCII
08-0A	Extension	ASCII
0B	Attribute	8-bit binary
0C-15	Reserved by MS-DOS	
16-17	Time stamp	16-bit binary
18-19	Date stamp	16-bit binary
1A-1B	Starting cluster number	16-bit binary
1C-1F	File size	32-bit binary

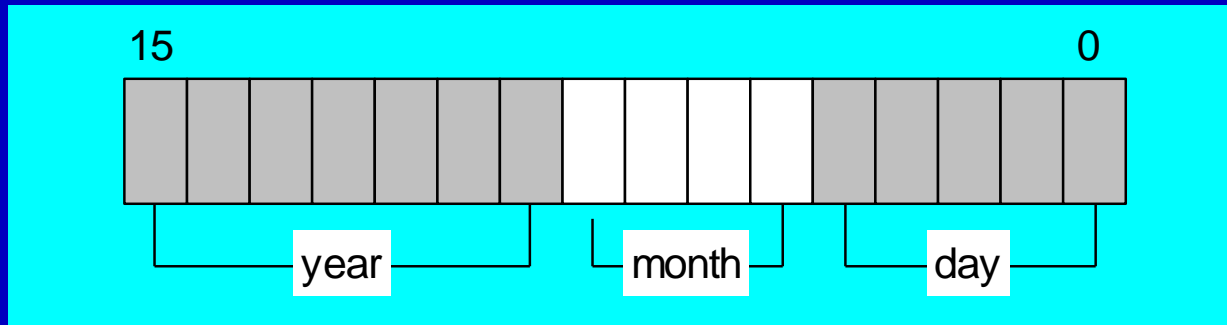
MS-DOS Directory Structure (2 of 2)

Time field equals 4DBDh (9:45:58), and the Date field equals 247Ah (March 26, 1998). Attribute is normal:

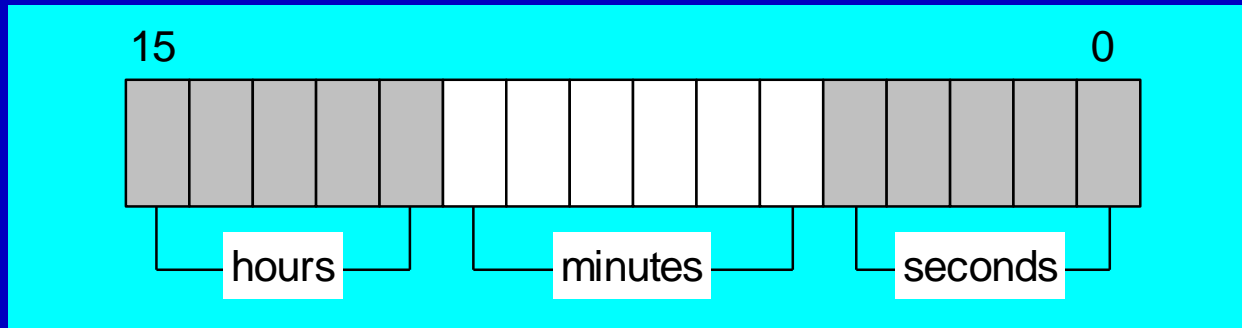


Date and Time Fields

- Date stamp field:

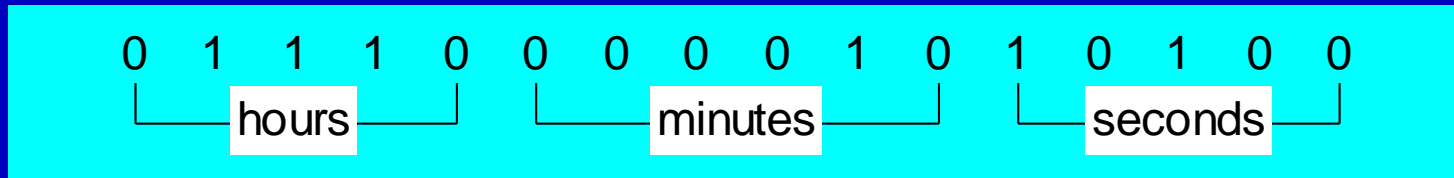


- Time stamp field:

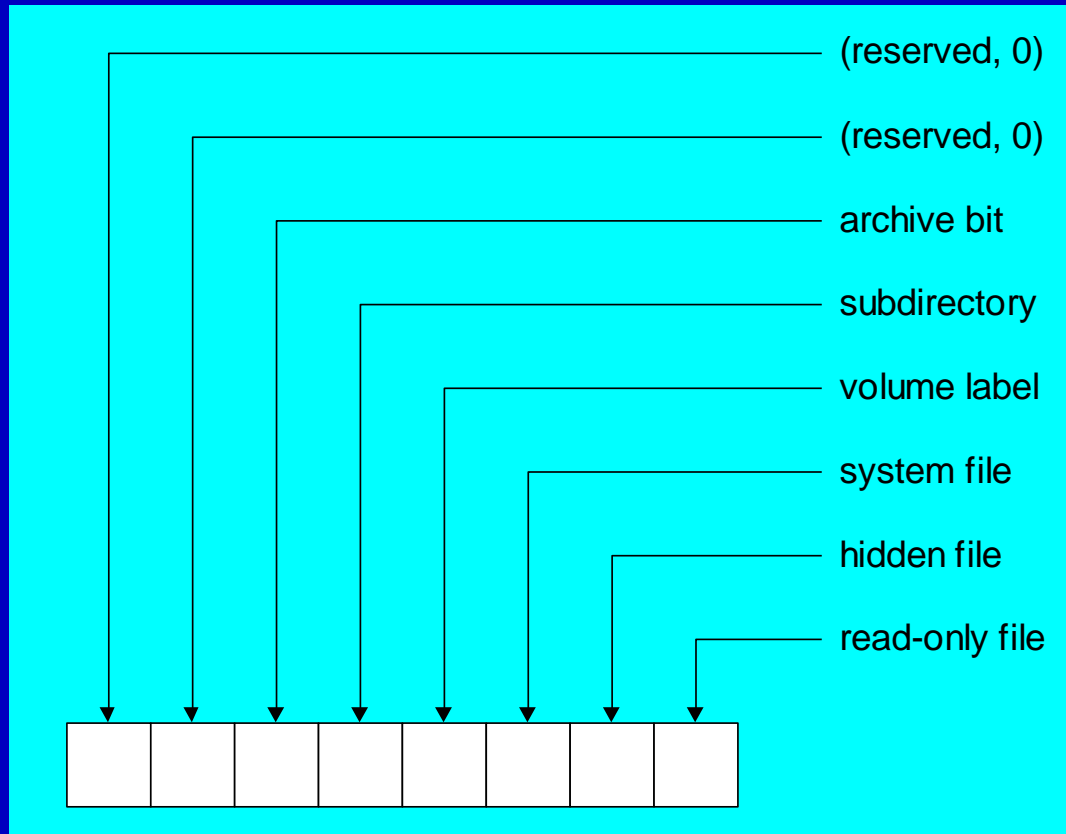


Your turn . . .

- What time value is represented here?



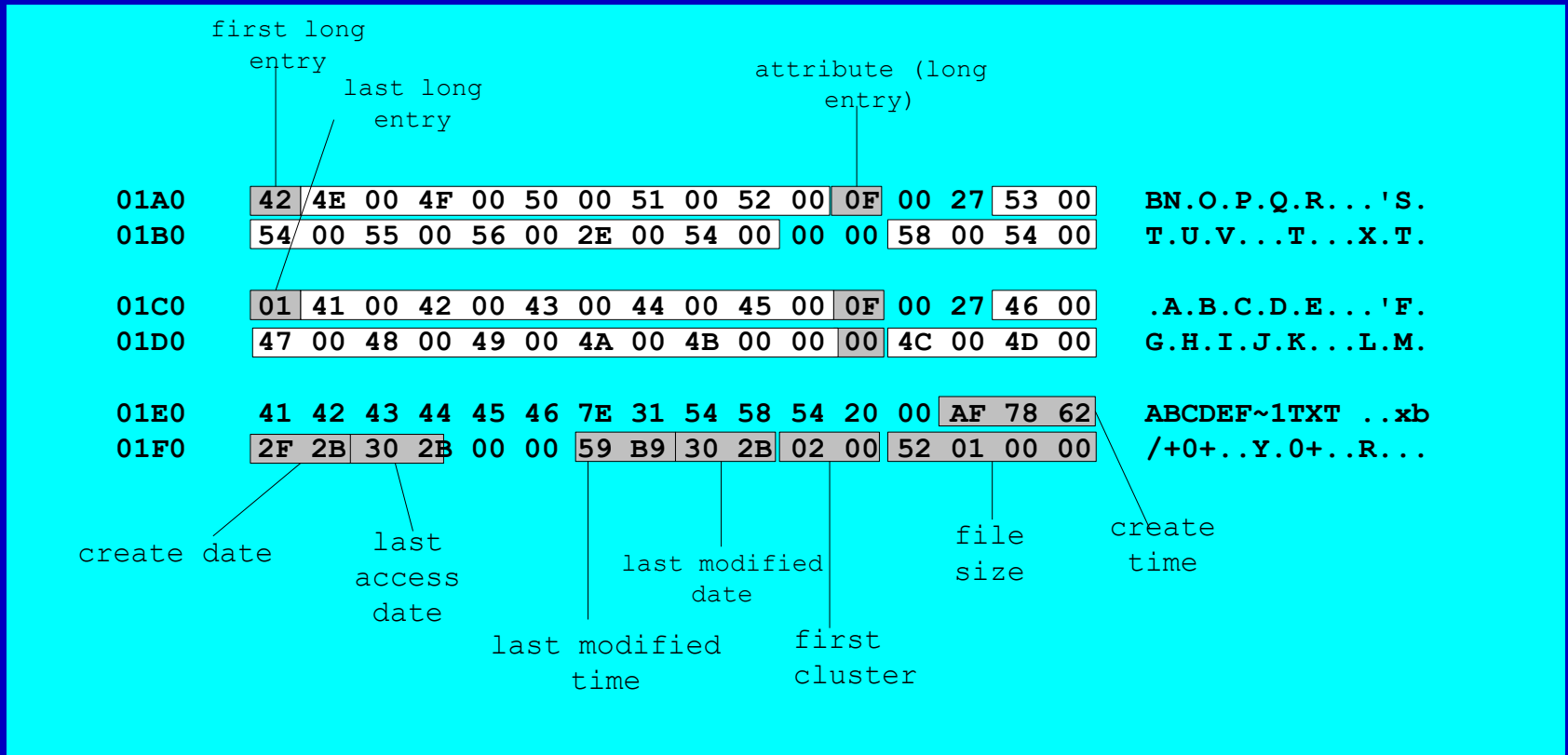
File Attribute Values



What type of file has attribute **00100111** . . . ?

Long Filenames in MS-Windows

Filename: ABCDEFGHIJKLMNOPQRSTUVWXYZ.TXT



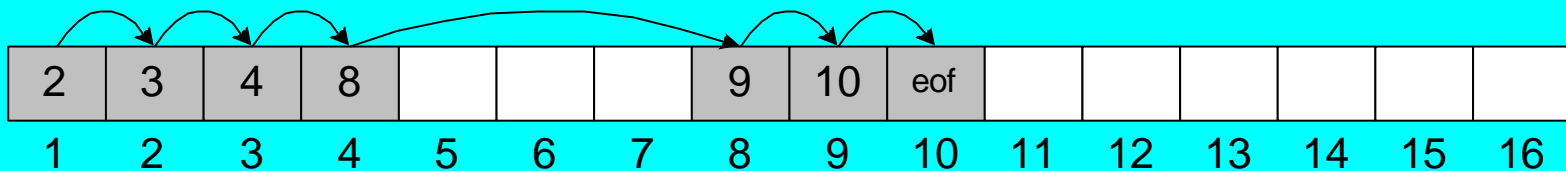
File Allocation Table (1 of 2)

- A map of all clusters on the disk, showing their ownership by specific files
- Each entry corresponds to a cluster number
- Each cluster contains one or more sectors
- Each file is represented in the FAT as a linked list, called a **cluster chain**.
- Three types of FAT's, named after the length of each FAT entry:
 - FAT-12
 - FAT-16
 - FAT-32

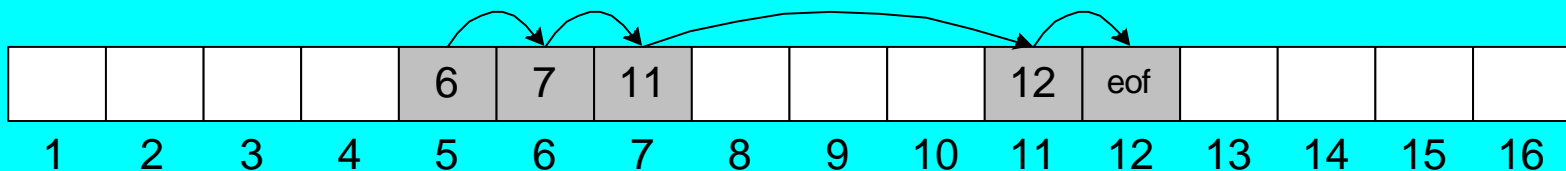
File Allocation Table (2 of 2)

- Each entry contains an n -bit integer that identifies the next entry. ($n=12,16$, or 32)
- Two cluster chains are shown in the following diagram, one for **File1**, and another for **File2**:

File1: starting cluster number = 1, size = 7 clusters



File2: starting cluster number = 5, size = 5 clusters



What's Next

- Disk Storage Systems
- File Systems
- Disk Directory
- **Reading and Writing Disk Sectors (7305h)**
- System-Level File Functions

Reading and Writing Disk Sectors (7305h)

- INT 21h, Function 7305h (absolute disk read and write)
- Reads and writes logical disk sectors
- Runs only in 16-bit Real-address mode
- Does not work under Windows NT, 2000, or XP
 - Tight security!

Reading and Writing Disk Sectors (7305h)

- AX = 7305h CX = FFFFh DL = drive number (01h=A:, etc.)
- SI = read/write mode flags, see below
- DS:BX -> disk I/O packet
- **Return:**
CF clear if successful CF set on error AX = error code
- Bitfields for Extended Absolute Disk Read/Write mode flags:
 - Bit(s) Description (Table 01791)
 - 0 direction (0=read, 1=write)
 - 12-1 reserved (0)
 - 14-13 write type (should be 00 on reads).

DISKIO Structure

- Used by Function 7305h:

```
DISKIO STRUCT
```

```
    startSector  DWORD 0           ; starting sector number
```

```
    numSectors   WORD 1           ; number of sectors
```

```
    bufferOfs    WORD buffer      ; buffer offset
```

```
    bufferSeg    WORD @DATA       ; buffer segment
```

```
DISKIO ENDS
```

Example

Example: Read one or more sectors from drive C:

```
.data
buffer BYTE 512 DUP(?)
diskStruct DISKIO <>
.code
    mov ax,7305h           ; absolute Read/Write
    mov cx,0FFFFh        ; always this value
    mov dl,3              ; drive C
    mov bx,OFFSET diskStruct
    mov si,0               ; read sector
    int 21h
```

Sector Display Program

- Pseudocode:
 - Ask for starting sector number and drive number
 - do while (keystroke <> ESC)
 - Display heading
 - Read one sector**
 - If MS-DOS error then exit
 - Display one sector
 - Wait for keystroke
 - Increment sector number**
 - end do

[View the source code](#)

Question 0 – Read sector 9.

- AX = 7305h CX = FFFFh DL = drive number (01h=A:, etc.)
- SI = read/write mode flags, see below
- DS:BX -> disk I/O packet
- **Return:** CF clear if successful , AX = error code
- Bitfields for Extended Absolute Disk Read/Write mode flags:

Bit(s) Description (Table 01791)

0 direction (0=read, 1=write)

12-1 reserved (0)

14-13 write type (should be 00 on reads).

DISKIO STRUCT

startSector DWORD 0 ; starting sector number

numSectors WORD 1 ; number of sectors

bufferOfs WORD buffer ; buffer offset

bufferSeg WORD @DATA ; buffer segment

DISKIO ENDS

What's Next

- Disk Storage Systems
- File Systems
- Disk Directory
- Reading and Writing Disk Sectors (7305h)
- **System-Level File Functions**

System-Level File Functions

- Common Disk-Related Functions
- Get Disk Free Space
- Create Subdirectory
- Remove Subdirecrory
- Set Current Directory
- Get Current Directory

Common Disk-Related Functions

Function Number	Function Name
0Eh	Set default drive
19h	Get default drive
7303h	Get disk free space
39h	Create subdirectory
3Ah	Remove subdirectory
3Bh	Set current directory
41h	Delete file
43h	Get/set file attribute
47h	Get current directory path
4Eh	Find first matching file
4Fh	Find next matching file
56h	Rename file
57h	Get/set file date and time
59h	Get extended error information

ExtGetDskFreSpcStruc Structure (1 of 2)

Function 7303h (Disk Free Space) return structure:

- **StructSize**: A return value that represents the size of the ExtGetDskFreSpcStruc structure, in bytes.
- **Level**: Always 0.
- **SectorsPerCluster**: The number of sectors inside each cluster.
- **BytesPerSector**: The number of bytes in each sector.
- **AvailableClusters**: The number of available clusters.
- **TotalClusters**: The total number of clusters in the volume.

ExtGetDskFreSpcStruc (2 of 2)

- **AvailablePhysSectors**: The number of physical sectors available in the volume, without adjustment for compression.
- **TotalPhysSectors**: The total number of physical sectors in the volume, without adjustment for compression.
- **AvailableAllocationUnits**: The number of available allocation units in the volume, without adjustment for compression.
- **TotalAllocationUnits**: The total number of allocation units in the volume, without adjustment for compression.
- **Rsvd**: Reserved member.

Function 7303h – Get Disk Free Space

- AX = 7303h
- ES:DI points to a ExtGetDskFreSpcStruc
- CX = size of the ExtGetDskFreSpcStruc variable
- DS:DX points to a null-terminated string containing the drive name

View the [DiskSpc.asm](#) program

Create Subdirectory

```
.data
pathname BYTE "\ASM",0

.code
    mov ah,39h                ; create subdirectory
    mov dx,OFFSET pathname
    int 21h
    jc  DisplayError
    .
    .
DisplayError:
```

Create Subdirectory

MKDIR - CREATE SUBDIRECTORY

AH = 39h

DS:DX -> ASCIZ pathname

Return:

CF clear if successful

AX destroyed

CF set on error

Notes: All directories in the given path except the last must exist.

Question 1: Write a program that creates a hidden, read-only directory named **__secret**.

Create File

CREAT - CREATE OR TRUNCATE FILE

AH = 3Ch

CX = file attributes (see below)

DS:DX -> ASCIZ filename

Return:

CF clear if successful, set on error

AX = file handle

AX = error code (03h,04h,05h)

Notes: If a file with the given name exists, it is truncated to zero length

Bitfields for file attributes:

0 read-only

1 hidden

2 system

3 volume label (ignored)

4 reserved, must be zero (directory)

5 archive bit

Question 2: Write a program that creates a hidden, read-only file named **C335.asm**.

Remove Subdirectory

```
.data
pathname  BYTE  'C:\ASM',0

.code
    mov  ah,3Ah                ; remove subdirectory
    mov  dx,OFFSET pathname
    int  21h
    jc   DisplayError
    .
    .
DisplayError:
```

Remove Subdirectory

RMDIR - REMOVE SUBDIRECTORY

AH = 3Ah

DS:DX -> ASCIZ pathname of directory to be removed

Return:

CF clear if successful AX destroyed

CF set on error

Notes: Directory must be empty (contain only '.' and '..' entries).

Question 3: Try to remove the directory by calling Function 3Ah.

Set Current Directory

```
.data
pathname  BYTE "C:\ASM\PROGS",0

.code
    mov  ah,3Bh                ; set current directory
    mov  dx,OFFSET pathname
    int  21h
    jc   DisplayError
    .
    .
DisplayError:
```

Get Current Directory

```
.data
pathname  BYTE 64 dup(0)    ; path stored here by MS-DOS

.code
    mov ah,47h              ; get current directory path
    mov dl,0                ; on default drive
    mov si,OFFSET pathname
    int 21h
    jc  DisplayError
    .
    .
DisplayError:
```

Your turn . . .

- Write a program that creates a hidden, read-only directory named `__secret`.
- Create a hidden file inside the new directory named `$$temp`.
- Try to remove the directory by calling Function 3Ah.
- Display the error code returned by MS-DOS.

Summary

- Disk controller: acts as a broker between the hardware and the operating system
- Disk characteristics
 - composed of tracks, cylinders, sectors
 - average seek time, data transfer rate
- Formatting & logical characteristics
 - master boot record, contains disk partition table
 - clusters – logical storage units
 - file allocation table – used by some systems
 - directory – root directory, subdirectories

The End

