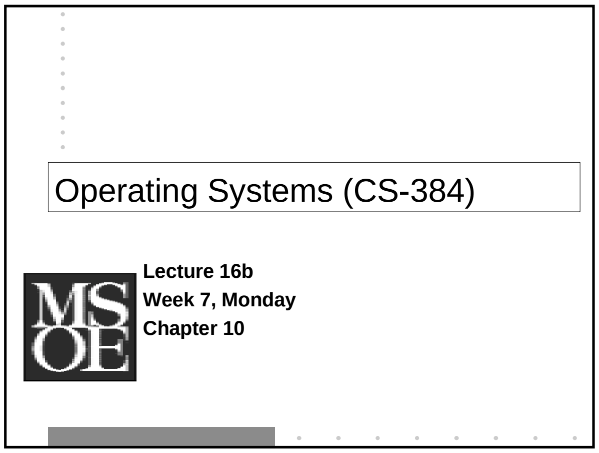
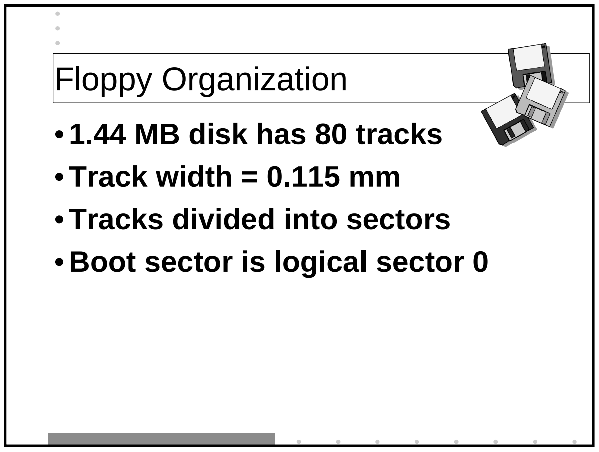


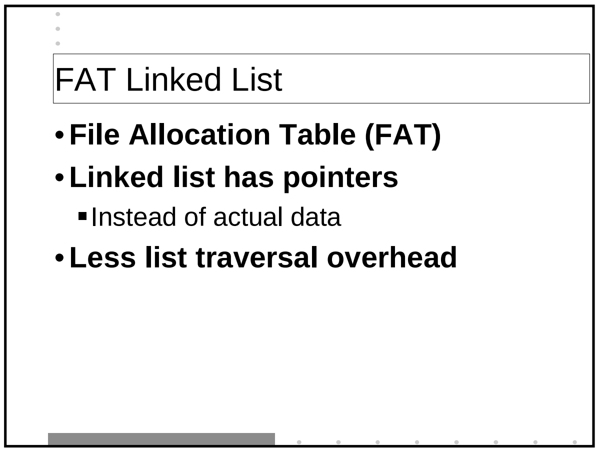
Operating Systems (CS-384)

Lecture 16b
Week 7, Monday
Chapter 10



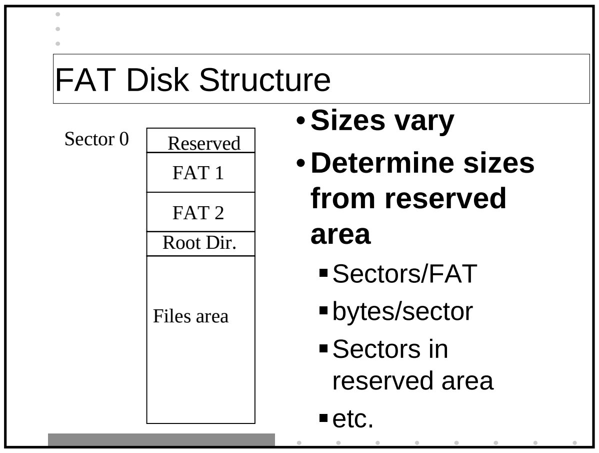
Floppy Organization

- **1.44 MB disk has 80 tracks**
- **Track width = 0.115 mm**
- **Tracks divided into sectors**
- **Boot sector is logical sector 0**

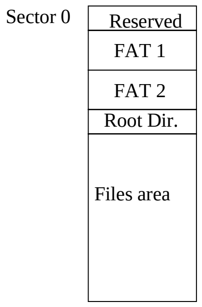


FAT Linked List

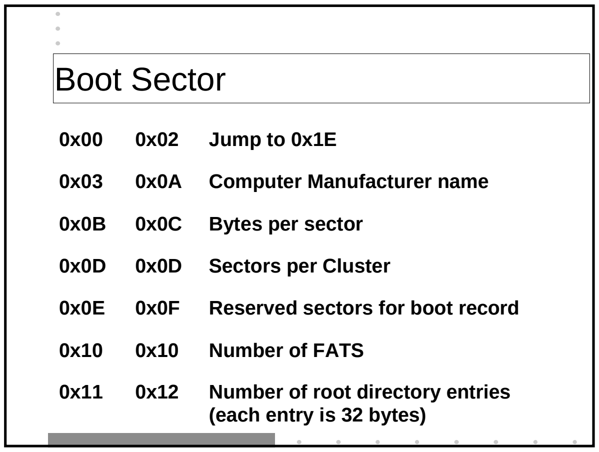
- **File Allocation Table (FAT)**
- **Linked list has pointers**
  - Instead of actual data
- **Less list traversal overhead**



FAT Disk Structure

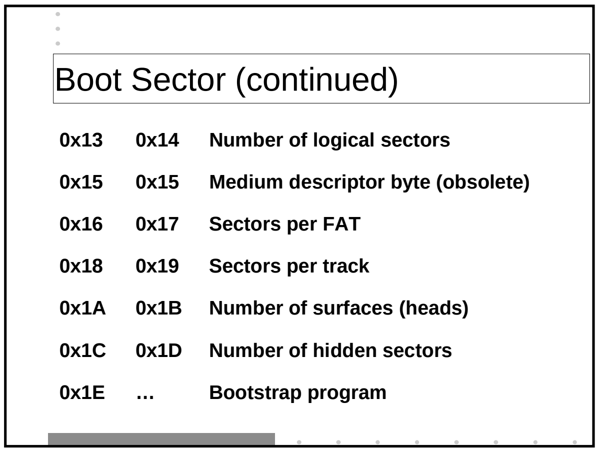


- **Sizes vary**
- **Determine sizes from reserved area**
  - Sectors/FAT
  - bytes/sector
  - Sectors in reserved area
  - etc.



Boot Sector

0x00	**0x02**	**Jump to 0x1E**
0x03	**0x0A**	**Computer Manufacturer name**
0x0B	**0x0C**	**Bytes per sector**
0x0D	**0x0D**	**Sectors per Cluster**
0x0E	**0x0F**	**Reserved sectors for boot record**
0x10	**0x10**	**Number of FATS**
0x11	**0x12**	**Number of root directory entries (each entry is 32 bytes)**



Boot Sector (continued)

0x13	**0x14**	**Number of logical sectors**
0x15	**0x15**	**Medium descriptor byte (obsolete)**
0x16	**0x17**	**Sectors per FAT**
0x18	**0x19**	**Sectors per track**
0x1A	**0x1B**	**Number of surfaces (heads)**
0x1C	**0x1D**	**Number of hidden sectors**
0x1E	**…**	**Bootstrap program**

Floppy organization

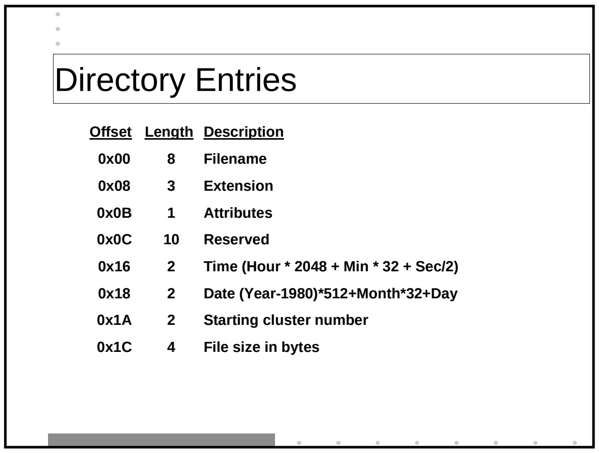
Logical Sector	Content
0	**Boot sector**
1	**First sector in FAT 1**
10	**First sector in FAT 2 (if present)**
19	**First sector of root directory**
XX	**Last sector in root directory**
XX+1	**Start of data area**

Low-level Reads

```
#include <windows.h>
…
string name = “\\\\.\\a:";
…
static HANDLE hDriver;
hDriver = CreateFile(name.c_str(),
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,        // default security
    OPEN_EXISTING,
    NULL, NULL);
if (hDriver == INVALID_HANDLE_VALUE)
{ // use GetLastError()
…
```

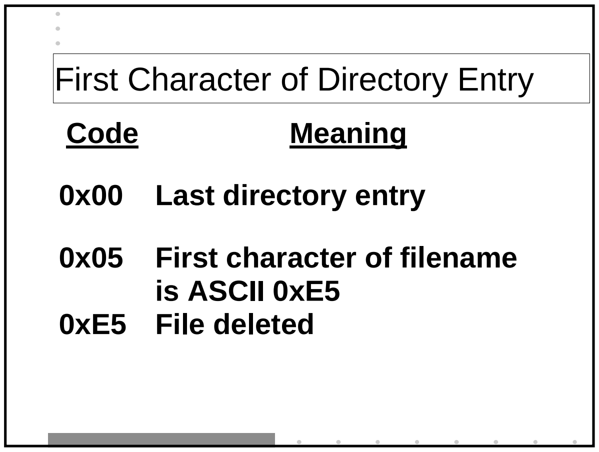
Low-level Reads (continued)

```
BYTE data[512];  //enough for a sector
DWORD bytesread;
…
if (ReadFile(hDriver,&data,512,
    &bytesread,NULL) == FALSE)
{
// error handler
}
else
{
…
```



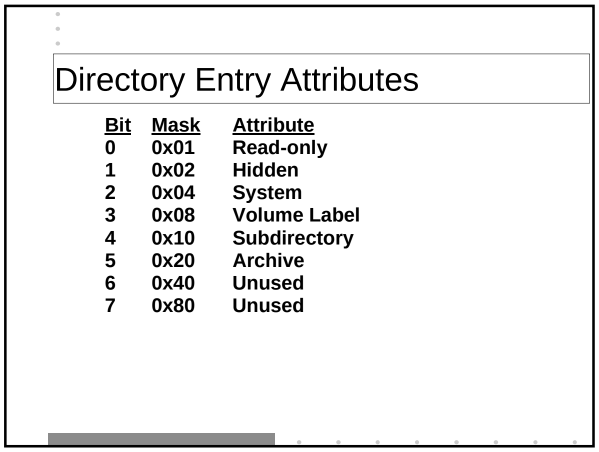
Directory Entries

Offset	Length	Description
0x00	8	Filename
0x08	3	Extension
0x0B	1	Attributes
0x0C	10	Reserved
0x16	2	Time (Hour * 2048 + Min * 32 + Sec/2)
0x18	2	Date (Year-1980)*512+Month*32+Day
0x1A	2	Starting cluster number
0x1C	4	File size in bytes



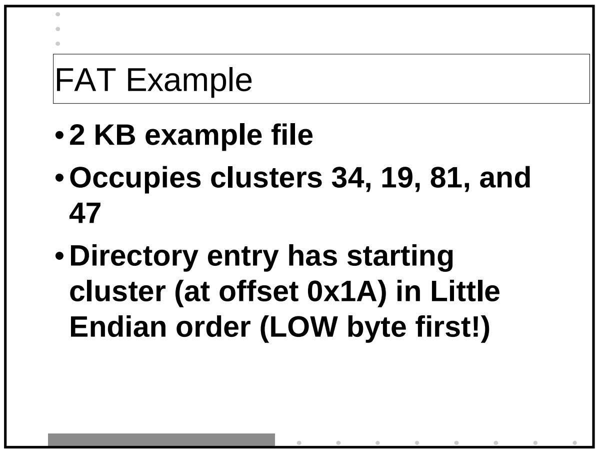
First Character of Directory Entry

Code	Meaning
0x00	Last directory entry
0x05	First character of filename is ASCII 0xE5
0xE5	File deleted



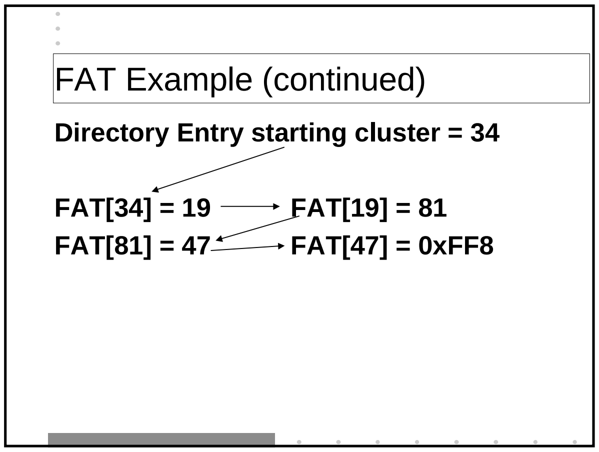
Directory Entry Attributes

Bit	Mask	Attribute
0	0x01	Read-only
1	0x02	Hidden
2	0x04	System
3	0x08	Volume Label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Unused
7	0x80	Unused



FAT Example

- **2 KB example file**
- **Occupies clusters 34, 19, 81, and 47**
- **Directory entry has starting cluster (at offset 0x1A) in Little Endian order (LOW byte first!)**

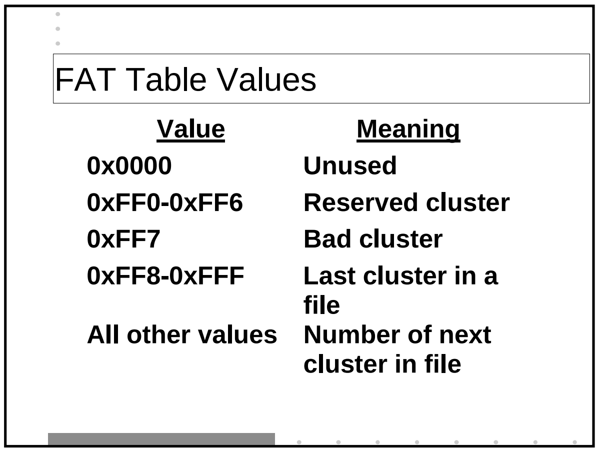


FAT Example (continued)

Directory Entry starting cluster = 34

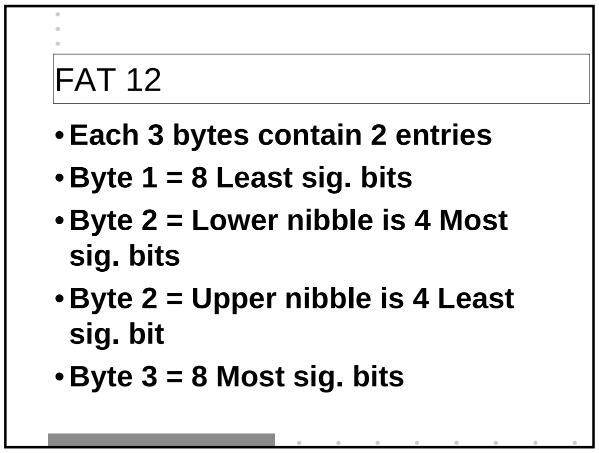
FAT[34] = 19 → **FAT[19] = 81**

FAT[81] = 47 → **FAT[47] = 0xFF8**



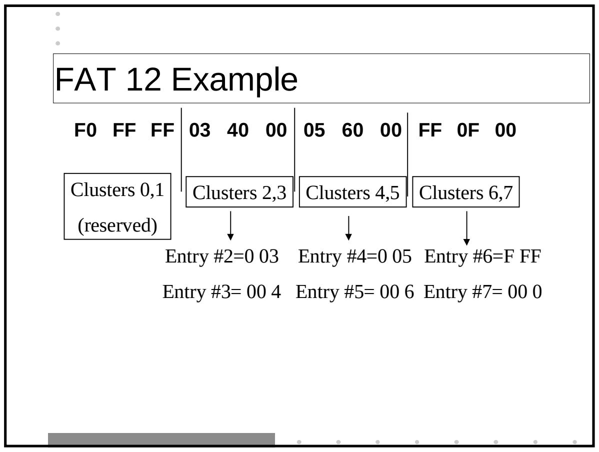
FAT Table Values

Value	**Meaning**
0x0000	**Unused**
0xFF0-0xFF6	**Reserved cluster**
0xFF7	**Bad cluster**
0xFF8-0xFFF	**Last cluster in a file**
All other values	**Number of next cluster in file**

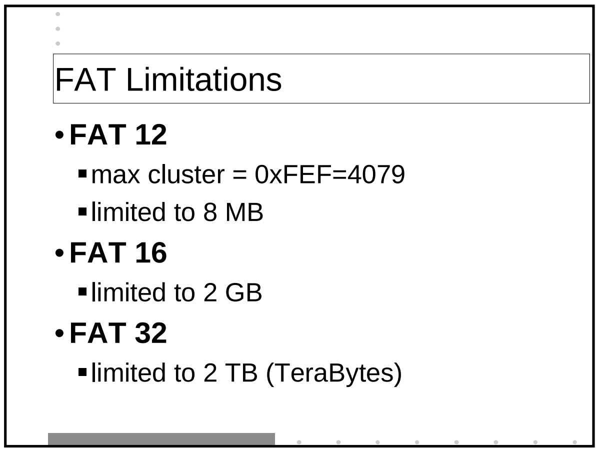


FAT 12

- **Each 3 bytes contain 2 entries**
- **Byte 1 = 8 Least sig. bits**
- **Byte 2 = Lower nibble is 4 Most sig. bits**
- **Byte 2 = Upper nibble is 4 Least sig. bit**
- **Byte 3 = 8 Most sig. bits**



FAT 12 Example



FAT Limitations

- **FAT 12**
  - max cluster = 0xFEF=4079
  - limited to 8 MB
- **FAT 16**
  - limited to 2 GB
- **FAT 32**
  - limited to 2 TB (TeraBytes)

VFAT (long file names)

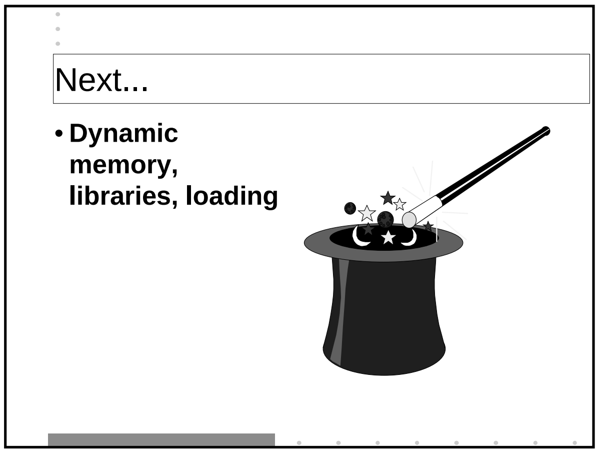
- **DOS ignores entry with attribute of 0x0F (read-only, hidden, system, and volume name)**
- **DOS sees short file name of first 6 characters, ~ (tilde), then number (starting with 1)**
- **VFAT long filename entries precede "normal" entry**

VFAT

- **Can fit 13 characters/directory entry**
- **May use up to 10 entries (for 128 byte filename)**
- **Characters are unicode characters (2 bytes/character)**

Long file name entry

Offset	**Meaning**
0x00	Bits 0:4 sequence number
	Bit 5 = 0
	Bit 6 = 1 if final component
	Bit 7 = 0
0x01-0x0A	5 filename characters
0x0B	File attribute (0x0F)
0x0C	Type indicator (always 0)
0x0D	Checksum
0x0E-0x19	Next 6 characters
0x1A-0x1B	Starting cluster (always 0)
0x1C-0x1F	Next 2 characters



Next...

- **Dynamic memory, libraries, loading**