Introduction

This page is intended to provide an introduction to the original File Allocation Table (FAT) file system. This file system was used on all versions of MS-DOS and PC-DOS, and on early versions of Windows; it is still used on floppy disks formatted by Windows and some other systems. Modified versions are also still supported by Windows on hard disks, if required.

The FAT file system is heavily based on the *file map* model in terms of its on-disk layout; that model was around for many years before Microsoft inherited the initial FAT file system from the original writers of DOS (Seattle Computer Products). It is a reasonably simple, reasonably robust file system.

There are three basic variants of the FAT file system, which differ mainly in the construction of the actual file allocation table. Floppy disks and small hard disks usually use the *12-bit* version, which was superseded by the *16-bit* version as hard disks became bigger. This in turn was superseded by the *32-bit* version as disks became bigger still. We shall concentrate on the 16-bit version, since the 12-bit version can be tricky for beginners, and the 32-bit version is more complex than needed for this tutorial.

Overview

Any disk is made up of *surfaces* (one for each head), *tracks* and *sectors*. However, for simplicity, we can consider a disk as a simple storage area made up just of a number of sectors. Further, these sectors are considered to be numbered consecutively, the first being numbered 0, the second numbered 1, etc.; we will not worry about the physical location of any sector on the actual disk. Because we want to emphasise that the location of a sector is irrelevant to the actual disk structure, and because sectors have their own numbers within each track, we shall call these sectors *blocks* from now on; as previously stated, they form a linear, densely numbered list.

All blocks are the same size, 512 bytes, on practically all FAT file systems. Howewer, large disks can have too many blocks for comfort, so blocks are sometimes grouped together in pairs (or fours, or eights, etc...); each such grouping is called an *allocation unit*. The FAT file system actually works in allocation units, not blocks, but for simplicity we shall assume in the description below that each allocation unit contains exactly one block, which means that we can use the terms interchangeably.

A note on numerical values

Hexadecimal numbers are indicated using the convention commonly used in C; that is, a leading `0x`. The decimal number `17` would thus be written as `0x11` in hexadecimal notation here.

Values in the FAT file system are either stored in *bytes* (8 bit values, 0-255 unsigned) or in *words* (pairs of bytes, 16 bit values, 0-65535 unsigned). Note that the first byte of a pair is the least significant byte, and the second byte of a pair is the most significant byte. For example, if the byte at position 3 has a value of 0x15, and the byte at position 4 has a value of 0x74, they together make up a word with value 0x7415 (not 0x1574).

There are occasional 32-bit values (*doublewords*), and these use a similar approach (in this case 4 bytes, with least significant byte stored first).

Lastly, note that individual bits within a byte or word are numbered from the least significant end (right hand end), starting with bit 0.

The disk format

This section describes the *on-disk structure* of a FAT file system; that is, how the various areas of the disk are laid out, and what is stored in them.

Basic layout

All disks using the FAT file system are divided into several areas. The following table summarises the areas in the order that they appear on the disk, starting at block 0:

Area description	Area size
Boot block	1 block
File Allocation Table (may be multiple copies)	Depends on file system size
Disk root directory	Variable (selected when disk is formatted)
File data area	The rest of the disk

The boot block

The boot block occupies just the first block of the disk. It holds a special program (the *bootstrap program*) which is used for loading the operating system into memory. It would thus appear to be fairly irrelevant to this discussion.

However, in the FAT file system it also contains several important data areas which help to describe the rest of the file system. Thus, to understand how a particular disk is laid out, it is necessary first to understand at least part of the contents of the boot block. The relevant areas are shown in the following table, together with their byte offsets from the start of the boot block. We will see, later, which of these are actually important to us.

Offset from start	Length	Description
0x00	3 bytes	Part of the bootstrap program.
0x03	8 bytes	Optional manufacturer description.
0x0b	2 bytes	Number of bytes per block (almost always 512).
0x0d	1 byte	Number of blocks per allocation unit.
0x0e	2 bytes	Number of reserved blocks. This is the number of blocks on the disk that are not actually part of the file system; in most cases this is exactly 1, being the allowance for the boot block.
0x10	1 byte	Number of File Allocation Tables.
0x11	2 bytes	Number of root directory entries (including unused ones).
0x13	2 bytes	Total number of blocks in the entire disk. If the disk size is larger than 65535 blocks (and thus will not fit in these two bytes), this value is set to zero, and the true size is stored at offset 0x20.
0x15	1 byte	Media Descriptor. This is rarely used, but still exists. .
0x16	2 bytes	The number of blocks occupied by one copy of the File Allocation Table.
0x18	2 bytes	The number of blocks per track. This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x1a	2 bytes	The number of heads (disk surfaces). This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x1c	4 bytes	The number of *hidden blocks*. The use of this is largely historical, and it is nearly always set to 0; thus it can be ignored.
0x20	4 bytes	Total number of blocks in the entire disk (see also offset 0x13).

0x24	2 bytes	Physical drive number. This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x26	1 byte	Extended Boot Record Signature This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x27	4 bytes	Volume Serial Number. Unique number used for identification of a particular disk.
0x2b	11 bytes	Volume Label. This is a string of characters for human-readable identification of the disk (padded with spaces if shorter); it is selected when the disk is formatted.
0x36	8 bytes	File system identifier (padded at the end with spaces if shorter).
0x3e	0x1c0 bytes	The remainder of the bootstrap program.
0x1fe	2 bytes	Boot block 'signature' (0x55 followed by 0xaa).

The Media Descriptor

Historically, the size and type of disk were difficult for the operating system to determine by hardware interrogation alone. A 'magic byte' was thus used to classify disks. This are still present, but rarely used, and its contents are known as the Media Descriptor. Generally, for hard disks, this is set to 0xf0.

The File Allocation Table (FAT)

The FAT occupies one or more blocks immediately following the boot block. Commonly, part of its last block will remain unused, since it is unlikely that the required number of entries will exactly fill a complete number of blocks. If there is a second FAT, this immediately follows the first (but starting in a new block). This is repeated for any further FATs.

Note that multiple FATs are used particularly on floppy disks, because of the higher likelihood of errors when reading the disk. If the FAT is unreadable, files cannot be accessed and another copy of the FAT must be used. On hard disks, there is often only one FAT.

In the case of the 16-bit FAT file system, each entry in the FAT is two bytes in length (i.e. 16 bits). The disk data area is divided into *clusters*, which are the same thing as allocation units, but numbered differently (instead of being numbered from the start of the disk, they are numbered from the start of the disk data area). So, the cluster number is the allocation unit number, minus a constant value which is the size of the areas in between the start of the disk and the start of the data area.

Well, almost. The clusters are numbered starting at 2, not 0! So the above calculation has to have 2 added to it to get the cluster number of a given allocation unit...and a cluster number is converted to an allocation unit number by subtracting 2...!

So, how does the FAT work? Simply, there is one entry in the FAT for every cluster (data area block) on the disk. Entry N relates to cluster N. Clusters 0 and 1 don't exist (because of the 'fiddle by 2' above), and those FAT entries are special. The first byte of the first entry is a copy of the media descriptor byte, and the second byte is set to 0xff. Both bytes in the second entry are set to 0xff.

What does a normal FAT entry for a cluster contain? It contains the *successor cluster number* - that is, the number of the cluster that follows this one in the file to which the current cluster belongs. The last cluster of a file has the value 0xffff in its FAT entry to indicate that there are no more clusters.

The Root Directory

The root directory contains an entry for each file whose name appears at the *root* (the top level) of the file system. Other directories can appear within the root directory; they are called *subdirectories*. The main

difference between the two is that space for the root directory is allocated statically, when the disk is formatted; there is thus a finite upper limit on the number of files that can appear in the root directory.

Subdirectories are just files with special data in them, so they can be as large or small as desired.

The format of all directories is the same. Each entry is 32 bytes (0x20) in size, so a single block can contain 16 of them. The following table shows a summary of a single directory entry; note that the offset is merely from the start of that particular entry, not from the start of the block.

Offset	Length	Description
0x00	8 bytes	Filename
0x08	3 bytes	Filename extension
0x0b	1 byte	File attributes
0x0c	10 bytes	Reserved
0x16	2 bytes	Time created or last updated
0x18	2 bytes	Date created or last updated
0x1a	2 bytes	Starting cluster number for file
0x1c	4 bytes	File size in bytes

The Filename

The eight bytes from offset 0x00 to 0x07 represent the filename. The first byte of the filename indicates its status. Usually, it contains a normal filename character (e.g. 'A'), but there are some special values:

0x00
: Filename never used.

0xe5
: The filename has been used, but the file has been deleted.

0x05
: The first character of the filename is actually 0xe5.

0x2e
: The entry is for a directory, not a normal file. If the second byte is also 0x2e, the cluster field contains the cluster number of this directory's parent directory. If the parent directory is the root directory (which is statically allocated and doesn't have a cluster number), cluster number 0x0000 is specified here.

Any other character
: This is the first character of a real filename.

If a filename is fewer than eight characters in length, it is padded with space characters.

The Filename Extension

The three bytes from offset 0x08 to 0x0a indicate the filename extension. There are no special characters. Note that the dot used to separate the filename and the filename extension is implied, and is not actually stored anywhere; it is just used when referring to the file. If the filename extension is fewer than three characters in length, it is padded with space characters.

The File Attributes

The single byte at offset 0x0b contains flags that provide information about the file and its permissions, etc. The flags are single bits, and have meanings as follows. Each bit is given as its numerical value, and these are combined to give the actual attribute value:

0x01
: Indicates that the file is read only.

0x02

Indicates a hidden file. Such files can be displayed if it is really required.

0x04
: Indicates a system file. These are hidden as well.

0x08
: Indicates a special entry containing the disk's volume label, instead of describing a file. This kind of entry appears only in the root directory.

0x10
: The entry describes a subdirectory.

0x20
: This is the archive flag. This can be set and cleared by the programmer or user, but is always set when the file is modified. It is used by backup programs.

0x40
: Not used; must be set to 0.

0x80
: Not used; must be set to 0.

The File Time

The two bytes at offsets 0x16 and 0x17 are treated as a 16 bit value; remember that the least significant byte is at offset 0x16. They contain the time when the file was created or last updated. The time is mapped in the bits as follows; the first line indicates the byte's offset, the second line indicates (in decimal) individual bit numbers in the 16 bit value, and the third line indicates what is stored in each bit.

```
<------- 0x17 --------> <------- 0x16 -------->
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
h  h  h  h  h  m  m  m  m  m  m  x  x  x  x  x
```

where:

`hhhhh`
: indicates the binary number of hours (0-23)

`mmmmmm`
: indicates the binary number of minutes (0-59)

`xxxxx`
: indicates the binary number of two-second periods (0-29), representing seconds 0 to 58.

The File Date

The two bytes at offsets 0x18 and 0x19 are treated as a 16 bit value; remember that the least significant byte is at offset 0x18. They contain the date when the file was created or last updated. The date is mapped in the bits as follows; the first line indicates the byte's offset, the second line indicates (in decimal) individual bit numbers in the 16 bit value, and the third line indicates what is stored in each bit.

```
<------- 0x19 --------> <------- 0x18 -------->
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
y  y  y  y  y  y  y  m  m  m  m  d  d  d  d  d
```

where:

`yyyyyyy`
: indicates the binary year offset from 1980 (0-119), representing the years 1980 to 2099

`mmmm`
: indicates the binary month number (1-12)

`ddddd`
: indicates the binary day number (1-31)

The Starting Cluster Number

The two bytes at offsets 0x1a and 0x1b are treated as a 16 bit value; remember that the least significant byte is at offset 0x1a. The first cluster for data space on the disk is always numbered as 0x0002. This strange

arrangement is because the first two entries in the FAT are reserved for other purposes.

The File Size

The four bytes at offsets 0x1c to 0x1f are treated as a 32 bit value; remember that the least significant byte is at offset 0x1c. They hold the actual file size, in bytes.

Worked examples

The best way to understand how to use the above information is to work though some simple examples.

Interpreting the contents of a block

We assume that there is a tool available to display the contents of a block in both hexadecimal and as ASCII characters. Most such tools will display unusual ASCII characters (e.g. carriage return) as a dot. For example, here is a display of a typical boot block:

```
Block 0 (0x0000)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  eb 3c 90 49 42 4d 2d 37 2e 30 20 00 02 01 01 00 .<.IBM-7.0 .....
010  01 40 00 a1 13 f8 14 00 0a 00 01 00 00 00 00 00 .@..............
020  00 00 00 00 00 00 29 2a 65 bc 00 43 4f 38 38 33 ......)*e..CO883
030  2d 41 32 20 20 20 46 41 54 31 36 20 20 20 fa 31 -A2   FAT16   .1
040  c0 8e d0 bc 00 7c fb 8e d8 e8 00 00 5e 83 c6 19 .....|......^...
050  bb 07 00 fc ac 84 c0 74 06 b4 0e cd 10 eb f5 30 .......t.......0
060  e4 cd 16 cd 19 0d 0a 4e 6f 6e 2d 73 79 73 74 65 .......Non-syste
070  6d 20 64 69 73 6b 0d 0a 50 72 65 73 73 20 61 6e m disk..Press an
080  79 20 6b 65 79 20 74 6f 20 72 65 62 6f 6f 74 0d y key to reboot.
090  0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa ..............U.
```

As an illustration, one field in the boot block has been highlighted in red (the highlight appears twice, once for the hexadecimal representation and once for the ASCII representation). The numbers down the left hand side are the offsets (from the start of the block) of the first byte on that row, and the first row of digits along the top are the offset of each byte within the row. We can thus easily see that the highlighted area starts at offset 0x36.

The area in question is (look back at the boot block layout) the file system type, in this case FAT16. To save us looking up each byte in a table of ASCII characters, we can simply consult the equivalent representation on the

right hand side. 0x46 represents F, 0x41 represents A, and so on.

Example 1 - find the root directory

To find the root directory, we need to examine the file system data in the boot block. So, let's look again at the boot block of our example disk:

```
Block 0 (0x0000)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  eb 3c 90 49 42 4d 2d 37 2e 30 20 00 02 01 01 00 .<.IBM-7.0 .....
010  01 40 00 a1 13 f8 14 00 0a 00 01 00 00 00 00 00 .@..............
020  00 00 00 00 00 00 29 2a 65 bc 00 43 4f 38 38 33 ......)*e..CO883
030  2d 41 32 20 20 20 46 41 54 31 36 20 20 20 fa 31 -A2   FAT16   .1
040  c0 8e d0 bc 00 7c fb 8e d8 e8 00 00 5e 83 c6 19 .....|......^...
050  bb 07 00 fc ac 84 c0 74 06 b4 0e cd 10 eb f5 30 .......t.......0
060  e4 cd 16 cd 19 0d 0a 4e 6f 6e 2d 73 79 73 74 65 .......Non-syste
070  6d 20 64 69 73 6b 0d 0a 50 72 65 73 73 20 61 6e m disk..Press an
080  79 20 6b 65 79 20 74 6f 20 72 65 62 6f 6f 74 0d y key to reboot.
090  0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa ..............U.
```

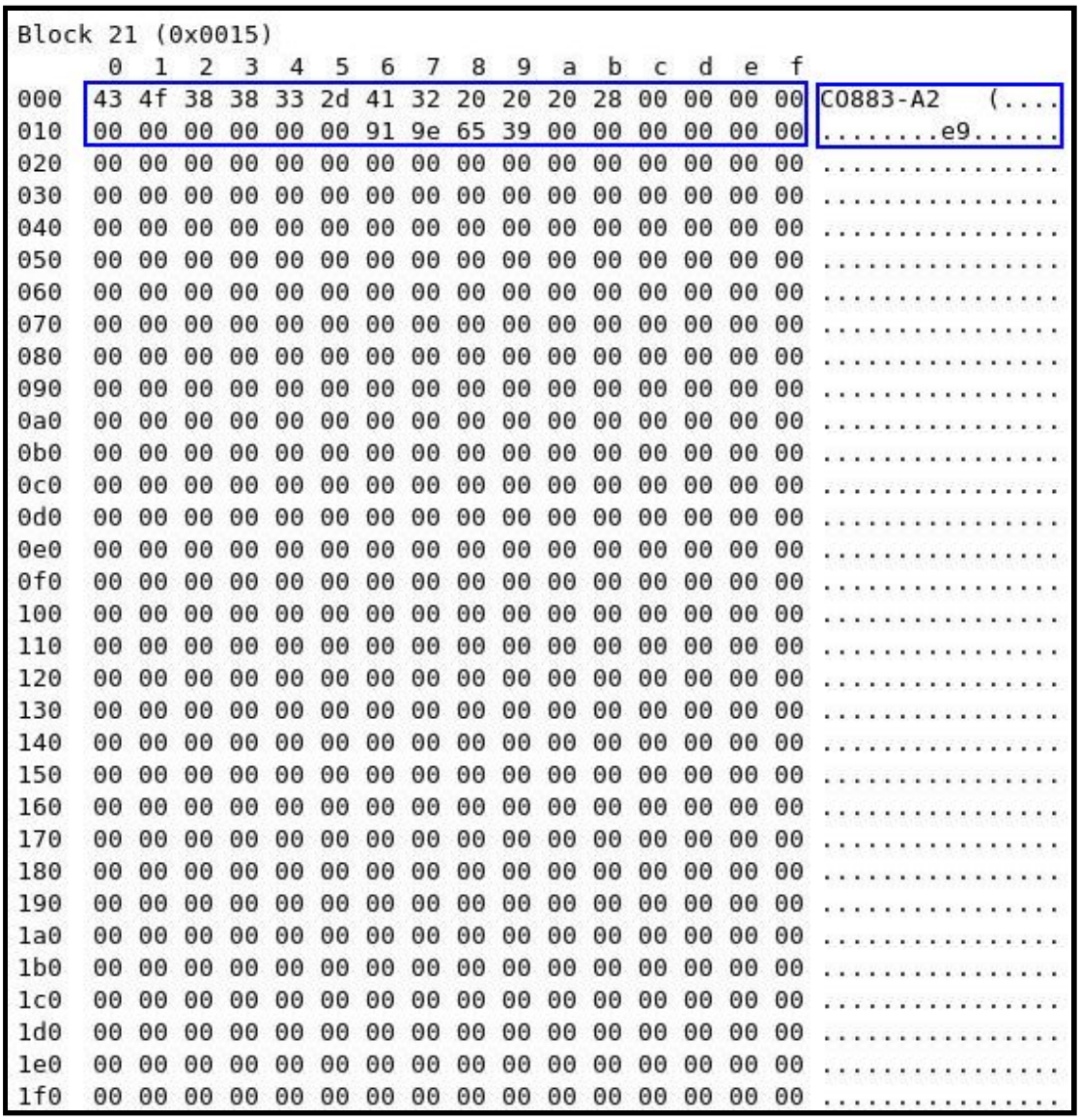
We know that the root directory appears immediately after the last copy of the FAT. So what we need to find out is the size of the FAT, and how many copies there are. We also need to know the size of anything else that appears before the FAT(s); there is just the single block of the boot block. So, the number of blocks that appear before the root directory is given by:

```
(size of FAT)*(number of FATs) + 1
```

All we need to do, then, is discover these values. First, we know that the number of FATs is stored at offset 0x10 (highlighted in green above); this tells us that there is just one FAT. Next, we need to know the size of a FAT; this is at offsets 0x16 and 0x17, where we find 0x14 and 0x00 respectively (highlighted in red above). Remember that these two bytes together make up a 16 bit value, with the least significant byte stored first; in other words, the value is 0x0014 (in decimal, 20). So, the total number of blocks that precede the root directory is given by:

```
0x0014*1 + 1  =>  0x0015  (decimal 21)
```

We should thus find the root directory in block 0x15, so let's look at it...



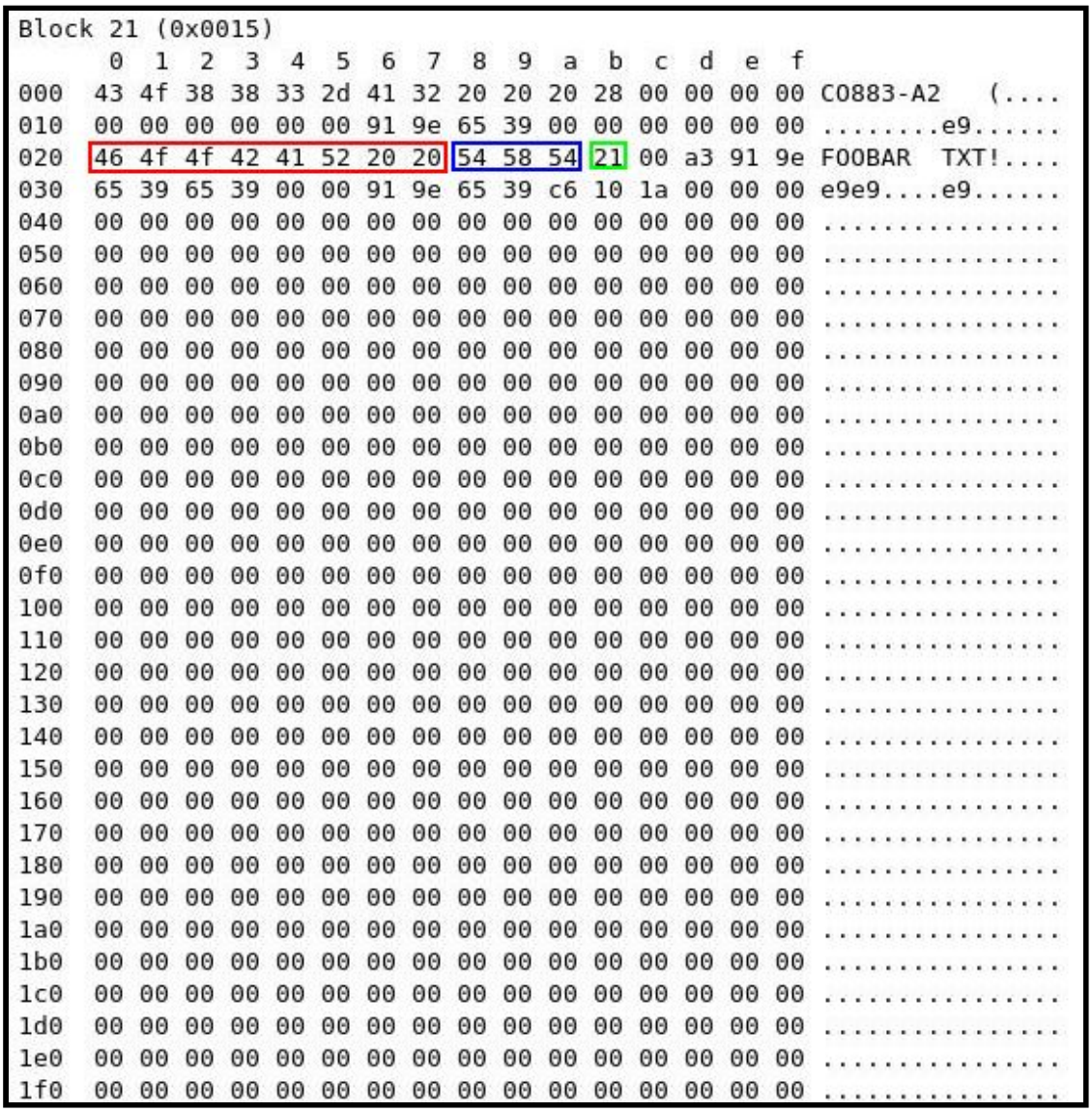
```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000   43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00  CO883-A2   (....
010   00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00  ........e9......
020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

It seems to have something occupying the first 0x20 bytes, and it's...a directory entry! We won't go into detail here, but detailed examination of those bytes would show that it's the special entry for the disk label. There don't appear to be any more entries in this directory.

Example 2 - find the attributes of a file

In this example, the file FOOBAR.TXT has been created on the same disk, and it appears in the root directory. We wish to find out which attribute flags are set on the file.

First, we need to find the root directory; we have already done this in example 1. Let's take a look at it after FOOBAR.TXT has been created:



```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00 CO883-A2   (....
010  00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00 ........e9......
020  46 4f 4f 42 41 52 20 20 54 58 54 21 00 a3 91 9e FOOBAR  TXT!....
030  65 39 65 39 00 00 91 9e 65 39 c6 10 1a 00 00 00 e9e9....e9......
040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

We can see fairly easily that the second directory entry (the one at offset 0x20) is that for FOOBAR.TXT. Remember that the dot between the filename and the filename extension is not actually stored, but is implied. We see the filename (highlighted in red) and the filename extension (highlighted in blue). We know that the attribute byte appears at offset 0x0b, and it is highlighted in green here.

The value of the attribute byte is 0x21. We can express this in binary as:

```
0 0 1 0   0 0 0 1
```

Taking each of the bits separately, and making a hexadecimal number out of them, we get:

```
0 0 1 0   0 0 0 0  =>  0x20
0 0 0 0   0 0 0 1  =>  0x01
```

Our table of attribute values shows that 0x20 means that the 'archive flag' is set, and 0x01 indicates that the file is read-only.

Example 3 - find the date of a file

Here, we want the date attached to a particular file (only one date is kept, which is the date of creation or last modification). The file in question is FOOBAR.TXT again.

Let's look once more at the root directory; we have already done this in example 2, and indeed we already know that FOOBAR.TXT has a directory entry at offset 0x20:

```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00 CO883-A2   (....
010  00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00 ........e9......
020  46 4f 4f 42 41 52 20 20 54 58 54 21 00 a3 91 9e FOOBAR  TXT!....
030  65 39 65 39 00 00 91 9e 65 39 c6 10 1a 00 00 00 e9e9....e9......
040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

This time we are interested in the file date, and we know from our root directory layout that this is at offset 0x18 within each directory entry. Thus, the date for FOOBAR.TXT is at offset 0x20+0x18, or 0x38 (highlighted in red above). Once again, this is a 16 bit value with the least significant byte stored first. The bytes are 0x65 and 0x39 respectively, so reversing these and putting them together gives a value of 0x3965.

Now all we have to do is analyse the components of this value. An easy way is first to convert it to binary, and this is even easier if we take it one hexadecimal digit at a time:

```
   3         9         6         5
   |         |         |         |
   V         V         V         V

0 0 1 1  1 0 0 1  0 1 1 0  0 1 0 1
```

Let's push all the digits together:

```
0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1
```

Now we can split them again on boundaries corresponding to the individual components of the date, as defined in the file date format. Then we convert each part back to decimal:

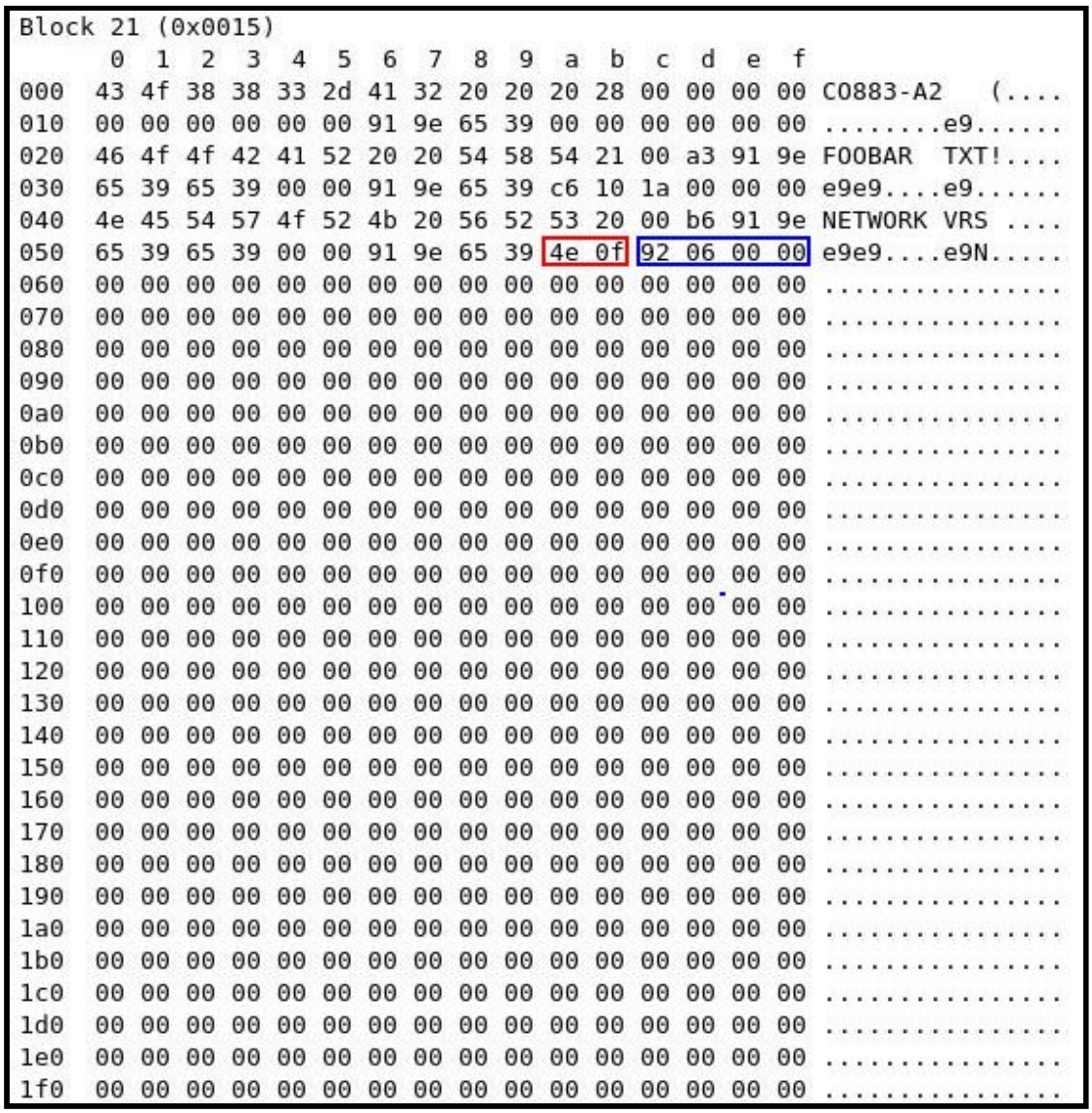
```
0 0 1 1 1 0 0   1 0 1 1   0 0 1 0 1

      |             |          |
      V             V          V

     28            11          5
  (year)       (month)      (day)
```

Remember that the year is based at 1980, so if we add 1980 to 28, we get 2008. The entire date is thus the 5th of November 2008.

Example 4 - find the data blocks for a file

Here, we wish to find out the numbers of the blocks containing data for a particular file which has now been added to the disk. The name of the file is NETWORK.VRS.

Once again, we find the root directory. Here are its latest contents, after NETWORK.VRS has been created:



```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000   43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00 CO883-A2   (....
010   00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00 ........e9......
020   46 4f 4f 42 41 52 20 20 54 58 54 21 00 a3 91 9e FOOBAR  TXT!....
030   65 39 65 39 00 00 91 9e 65 39 c6 10 1a 00 00 00 e9e9....e9......
040   4e 45 54 57 4f 52 4b 20 56 52 53 20 00 b6 91 9e NETWORK VRS ....
050   65 39 65 39 00 00 91 9e 65 39 4e 0f 92 06 00 00 e9e9....e9N.....
060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Note that the third directory entry (starting at offset 0x40) is that for NETWORK.VRS. We know that the starting cluster number for the file data occupies bytes at offsets 0x1a and 0x1b in a particular directory entry; thus the bytes we want are at offsets 0x5a and 0x5b (we just added 0x40, the offset of the start of the entry). These (highlighted in red) contain 0x4e and 0x0f respectively, and, remembering that the first byte is the least significant one, the number we want is 0x0f4e. Incidentally, the next four bytes (highlighted in blue) are the file size, again with the least significant byte first. These are 0x92, 0x06, 0x00, 0x00 respectively, making a value of 0x00000692. This (in decimal) is 1682. So, this file is **1682** bytes long.

Let's review what we know so far...

- The starting cluster of the file is cluster 0x0f4e.
- The root directory starts at block 0x15.
- The first allocation unit starts at the first block after the root directory.

What else do we need to know? We know where the root directory starts, but not where it ends. So we need the size of the root directory, in blocks. Let's look once again at the boot block:

```
Block 0 (0x0000)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  eb 3c 90 49 42 4d 2d 37 2e 30 20 00 02 01 01 00 .<.IBM-7.0 .....
010  01 40 00 a1 13 f8 14 00 0a 00 01 00 00 00 00 00 .@..............
020  00 00 00 00 00 00 29 2a 65 bc 00 43 4f 38 38 33 ......)*e..CO883
030  2d 41 32 20 20 20 46 41 54 31 36 20 20 20 fa 31 -A2   FAT16   .1
040  c0 8e d0 bc 00 7c fb 8e d8 e8 00 00 5e 83 c6 19 .....|......^...
050  bb 07 00 fc ac 84 c0 74 06 b4 0e cd 10 eb f5 30 .......t.......0
060  e4 cd 16 cd 19 0d 0a 4e 6f 6e 2d 73 79 73 74 65 .......Non-syste
070  6d 20 64 69 73 6b 0d 0a 50 72 65 73 73 20 61 6e m disk..Press an
080  79 20 6b 65 79 20 74 6f 20 72 65 62 6f 6f 74 0d y key to reboot.
090  0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa ..............U.
```

What we need to find this time is the maximum number of entries in the root directory; this is fixed when the disk is formatted. We know from the boot block layout that this appears in the two bytes starting at offset 0x11 in the boot block (these are highlighted in red above). These bytes contain 0x40 and 0x00 respectively, so (arranging as usual) this gives us a value of 0x0040 (64 in decimal). So there are 64 root directory entries. We know that one directory entry occupies 32 bytes, so the total space occupied by the root directory is 64*32 bytes, or 2048 bytes. Each block is 512 bytes, so the number of blocks occupied by the root directory is 2048 divided by 512...that is, 4.

So, the root directory starts at block 0x15. Thus the first allocation unit starts at 0x15+4, or 0x19. So, to convert an allocation unit number to a block number, we need to add the constant value 0x19. And to convert a cluster number (which is what appears in the root directory) to a block number, we need to add 0x17, to allow for that strange offset of 2.

We now know that the first data block of the file is at cluster number 0xf4e (see above). Adding the constant we have discovered, we find that this is block number 0xf4e+0x17, or 0xf65. Let's look at block 0xf65:

```
Block 3941 (0x0f65)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  20 20 20 54 77 61 73 20 74 68 65 20 6e 69 67 68    Twas the nigh
010  74 20 62 65 66 6f 72 65 20 73 74 61 72 74 2d 75 t before start-u
020  70 20 61 6e 64 20 61 6c 6c 20 74 68 72 6f 75 67 p and all throug
030  68 20 74 68 65 20 6e 65 74 2c 0a 20 20 20 20 20 h the net,.
040  6e 6f 74 20 61 20 70 61 63 6b 65 74 20 77 61 73 not a packet was
050  20 6d 6f 76 69 6e 67 3b 20 6e 6f 20 62 69 74 20  moving; no bit
060  6e 6f 72 20 6f 63 74 65 74 2e 0a 20 20 20 54 68 nor octet..   Th
070  65 20 65 6e 67 69 6e 65 65 72 73 20 72 61 74 74 e engineers ratt
080  6c 65 64 20 74 68 65 69 72 20 63 61 72 64 73 20 led their cards
090  69 6e 20 64 65 73 70 61 69 72 2c 0a 20 20 20 20 in despair,.
0a0  20 68 6f 70 69 6e 67 20 61 20 62 61 64 20 63 68  hoping a bad ch
0b0  69 70 20 77 6f 75 6c 64 20 62 6c 6f 77 20 77 69 ip would blow wi
0c0  74 68 20 61 20 66 6c 61 72 65 2e 0a 20 20 20 54 th a flare..   T
0d0  68 65 20 73 61 6c 65 73 6d 65 6e 20 77 65 72 65 he salesmen were
0e0  20 6e 65 73 74 6c 65 64 20 61 6c 6c 20 73 6e 75  nestled all snu
0f0  67 20 69 6e 20 74 68 65 69 72 20 62 65 64 73 2c g in their beds,
100  0a 20 20 20 20 20 77 68 69 6c 65 20 76 69 73 69 .     while visi
110  6f 6e 73 20 6f 66 20 64 61 74 61 20 6e 65 74 73 ons of data nets
120  20 64 61 6e 63 65 64 20 69 6e 20 74 68 65 69 72  danced in their
130  20 68 65 61 64 73 2e 0a 20 20 20 41 6e 64 20 49  heads..   And I
140  20 77 69 74 68 20 6d 79 20 64 61 74 61 73 63 6f  with my datasco
150  70 65 20 74 72 61 63 69 6e 67 73 20 61 6e 64 20 pe tracings and
160  64 75 6d 70 73 0a 20 20 20 20 20 70 72 65 70 61 dumps.     prepa
170  72 65 64 20 66 6f 72 20 73 6f 6d 65 20 70 72 65 red for some pre
180  74 74 79 20 62 61 64 20 62 72 75 69 73 65 73 20 tty bad bruises
190  61 6e 64 20 6c 75 6d 70 73 2e 0a 20 20 20 57 68 and lumps..   Wh
1a0  65 6e 20 6f 75 74 20 69 6e 20 74 68 65 20 68 61 en out in the ha
1b0  6c 6c 20 74 68 65 72 65 20 61 72 6f 73 65 20 73 ll there arose s
1c0  75 63 68 20 61 20 63 6c 61 74 74 65 72 2c 0a 20 uch a clatter,.
1d0  20 20 20 20 49 20 73 70 72 61 6e 67 20 66 72 6f     I sprang fro
1e0  6d 20 6d 79 20 64 65 73 6b 20 74 6f 20 73 65 65 m my desk to see
1f0  20 77 68 61 74 20 77 61 73 20 74 68 65 20 6d 61  what was the ma
```

Well, that certainly looks like the start of a poem! Each line of the text is separated by a special character called *newline*, which has the code 0x0a (decimal 10). The first few of these are highlighted in red.

We have nearly finished. There is obviously more of this file, and for us to find the rest of it, we need to consult the FAT. Recall that the starting *cluster* number of the file (the block we just looked at) is 0xf4e. Each entry in the FAT is two bytes in size, so we'll find the entry for that cluster at offset 0xf4e*2 in the FAT, which is offset 0x1e9c (it's easier to add the value twice than attempt multiplication). We know that one disk block (and thus one block of the FAT) is 0x200 bytes in size, so we just need to divide 0x1e9c by 0x200. This sounds hard, but it isn't. You can find tools for this, or do it yourself. Let's look at these two numbers in binary:

```
0x0200       => 0 0 0 0  0 0 1 0  0 0 0 0  0 0 0 0
0x1e9c       => 0 0 0 1  1 1 1 0  1 0 0 1  1 1 0 0
```

The first number is a power of two, so to divide by it we simply shift the second number right - in this case by nine places:

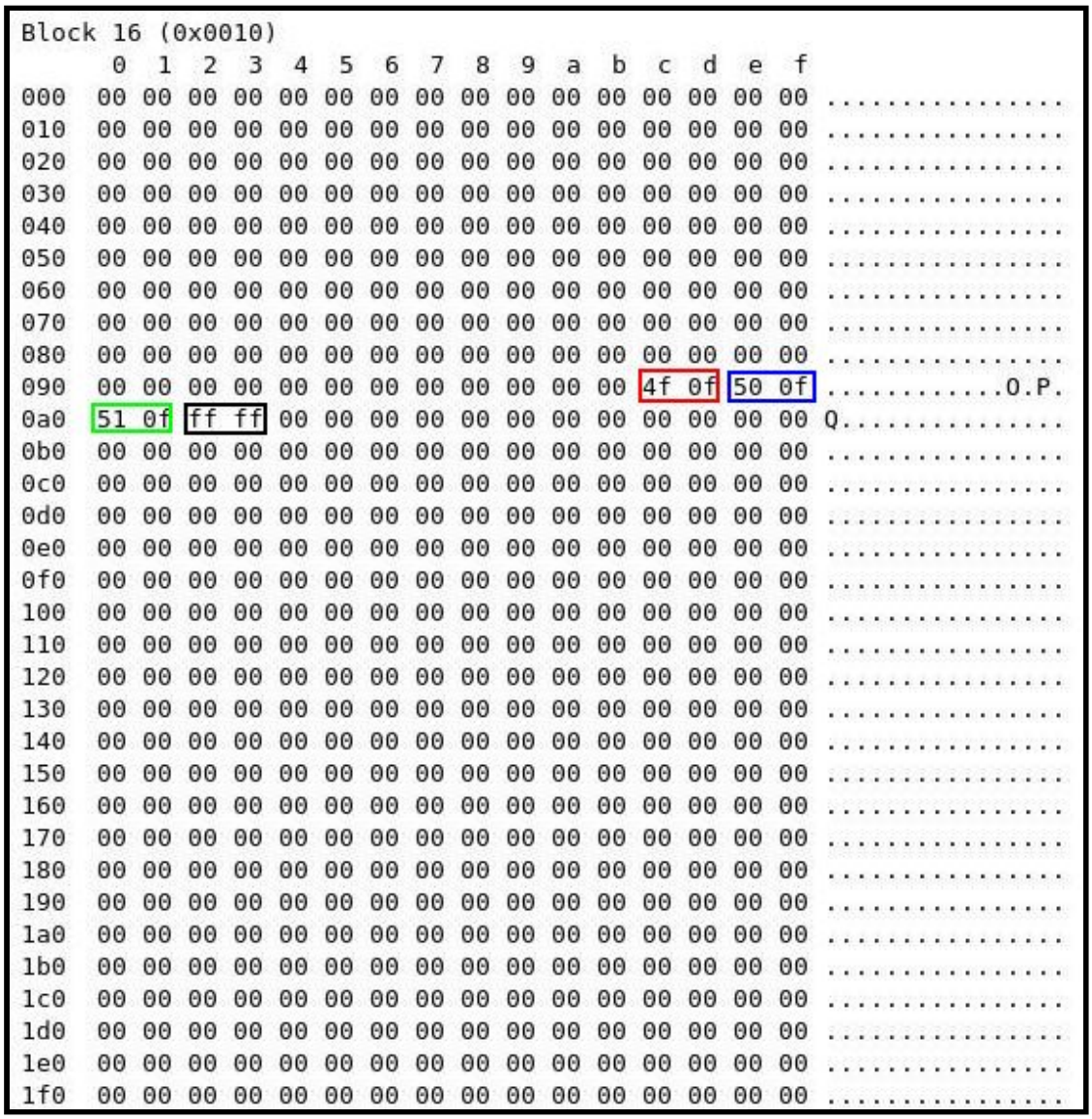
```
0 0 0 0  0 0 0 0  0 0 0 0  1 1 1 1   =>  0x0f
```

So the entry we want is in block 0x0f of the FAT. The remainder from our division is of course all the bits we lost when we shifted:

```
0  1 0 0 1  1 1 0 0     =>  0x9c
```

so this is the byte offset of the entry within the FAT block.

We need to find FAT block 0x0f. We know the FAT starts in block 1 of the disk (see earlier), so block 0x0f of the FAT will be in disk block 0x0f+1, or block 0x10. Let's look at that block:



```
Block 16 (0x0010)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
090   00 00 00 00 00 00 00 00 00 00 00 00 4f 0f 50 0f ............O.P.
0a0   51 0f ff ff 00 00 00 00 00 00 00 00 00 00 00 00 Q...............
0b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

We need to look at the FAT entry (two bytes) at offset 0x9c; this is highlighted in red above, and resolves to the 16 bit value 0x0f4f. This is actually the very next cluster, numerically, from the one we have just looked at (this will not always be the case), so we can apply a bit of common sense and deduce that the second data block of the file appears immediately after the first; thus, the first two blocks are at 0xf65 and 0xf66. Here is block 0xf66:

```
Block 3942 (0x0f66)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  74 74 65 72 2e 0a 0a 20 20 20 54 68 65 72 65 20 tter...   There
010  73 74 6f 6f 64 20 61 74 20 74 68 65 20 74 68 72 stood at the thr
020  65 73 68 6f 6c 64 20 77 69 74 68 20 50 43 20 69 eshold with PC i
030  6e 20 74 6f 77 2c 0a 20 20 20 20 20 41 6e 20 41 n tow,.     An A
040  52 50 41 4e 45 54 20 68 61 63 6b 65 72 2c 20 61 RPANET hacker, a
050  6c 6c 20 72 65 61 64 79 20 74 6f 20 67 6f 2e 0a ll ready to go..
060  20 20 20 49 20 63 6f 75 6c 64 20 73 65 65 20 66    I could see f
070  72 6f 6d 20 74 68 65 20 63 72 65 61 73 65 73 20 rom the creases
080  74 68 61 74 20 63 6f 76 65 72 65 64 20 68 69 73 that covered his
090  20 62 72 6f 77 2c 0a 20 20 20 20 20 68 65 27 64  brow,.     he'd
0a0  20 63 6f 6e 71 75 65 72 20 74 68 65 20 63 72 69  conquer the cri
0b0  73 69 73 20 63 6f 6e 66 72 6f 6e 74 69 6e 67 20 sis confronting
0c0  68 69 6d 20 6e 6f 77 2e 0a 20 20 20 4d 6f 72 65 him now..   More
0d0  20 72 61 70 69 64 20 74 68 61 6e 20 65 61 67 6c  rapid than eagl
0e0  65 73 2c 20 68 65 20 63 68 65 63 6b 65 64 20 65 es, he checked e
0f0  61 63 68 20 61 6c 61 72 6d 0a 20 20 20 20 20 61 ach alarm.     a
100  6e 64 20 73 63 72 75 74 69 6e 69 7a 65 64 20 65 nd scrutinized e
110  61 63 68 20 66 6f 72 20 69 74 73 20 70 6f 74 65 ach for its pote
120  6e 74 69 61 6c 20 68 61 72 6d 2e 0a 0a 20 20 20 ntial harm...
130  4f 6e 20 4c 41 50 42 2c 20 6f 6e 20 4f 53 49 2c On LAPB, on OSI,
140  20 58 2e 32 35 21 0a 20 20 20 20 20 54 43 50 2c  X.25!.     TCP,
150  20 53 4e 41 2c 20 56 2e 33 35 21 0a 0a 20 20 20  SNA, V.35!..
160  48 69 73 20 65 79 65 73 20 77 65 72 65 20 61 66 His eyes were af
170  69 72 65 20 77 69 74 68 20 74 68 65 20 73 74 72 ire with the str
180  65 6e 67 74 68 20 6f 66 20 68 69 73 20 67 61 7a ength of his gaz
190  65 3b 0a 20 20 20 20 20 6e 6f 20 62 75 67 20 63 e;.     no bug c
1a0  6f 75 6c 64 20 68 69 64 65 20 6c 6f 6e 67 3b 20 ould hide long;
1b0  6e 6f 74 20 66 6f 72 20 68 6f 75 72 73 20 6f 72 not for hours or
1c0  20 64 61 79 73 2e 0a 20 20 20 41 20 77 69 6e 6b  days..   A wink
1d0  20 6f 66 20 68 69 73 20 65 79 65 20 61 6e 64 20  of his eye and
1e0  61 20 74 77 69 74 63 68 20 6f 66 20 68 69 73 20 a twitch of his
1f0  68 65 61 64 2c 0a 20 20 20 20 20 73 6f 6f 6e 20 head,.     soon
```

which certainly looks like the continuation of the poem. If we look at the FAT entry for this new cluster (which, since it's the next block, will also be the next cluster and thus in the next FAT entry), it is highlighted in blue above, and contains the value 0x0f50. This is the very next block and cluster:

```
Block 3943 (0x0f67)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  67 61 76 65 20 6d 65 20 74 6f 20 6b 6e 6f 77 20 gave me to know
010  49 20 68 61 64 20 6c 69 74 74 6c 65 20 74 6f 20 I had little to
020  64 72 65 61 64 2e 0a 20 20 20 48 65 20 73 70 6f dread..   He spo
030  6b 65 20 6e 6f 74 20 61 20 77 6f 72 64 2c 20 62 ke not a word, b
040  75 74 20 77 65 6e 74 20 73 74 72 61 69 67 68 74 ut went straight
050  20 74 6f 20 68 69 73 20 77 6f 72 6b 2c 0a 20 20  to his work,.
060  20 20 20 66 69 78 69 6e 67 20 61 20 6e 65 74 20    fixing a net
070  74 68 61 74 20 68 61 64 20 67 6f 6e 65 20 70 6c that had gone pl
080  75 6d 62 20 62 65 72 73 65 72 6b 3b 0a 20 20 20 umb berserk;.
090  41 6e 64 20 6c 61 79 69 6e 67 20 61 20 66 69 6e And laying a fin
0a0  67 65 72 20 6f 6e 20 6f 6e 65 20 73 75 73 70 65 ger on one suspe
0b0  63 74 20 6c 69 6e 65 2c 0a 20 20 20 20 20 68 65 ct line,.     he
0c0  20 65 6e 74 65 72 65 64 20 61 20 70 61 74 63 68  entered a patch
0d0  20 61 6e 64 20 74 68 65 20 6e 65 74 20 63 61 6d  and the net cam
0e0  65 20 75 70 20 66 69 6e 65 21 0a 0a 20 20 20 54 e up fine!..   T
0f0  68 65 20 70 61 63 6b 65 74 73 20 66 6c 6f 77 65 he packets flowe
100  64 20 6e 65 61 74 6c 79 20 61 6e 64 20 70 72 6f d neatly and pro
110  74 6f 63 6f 6c 73 20 6d 61 74 63 68 65 64 3b 0a tocols matched;.
120  20 20 20 20 20 74 68 65 20 68 6f 73 74 73 20 69      the hosts i
130  6e 74 65 72 66 61 63 65 64 20 61 6e 64 20 73 68 nterfaced and sh
140  69 66 74 2d 72 65 67 69 73 74 65 72 73 20 6c 61 ift-registers la
150  74 63 68 65 64 2e 0a 20 20 20 48 65 20 74 65 73 tched..   He tes
160  74 65 64 20 74 68 65 20 73 79 73 74 65 6d 20 66 ted the system f
170  72 6f 6d 20 47 61 74 65 77 61 79 20 74 6f 20 50 rom Gateway to P
180  41 44 3b 0a 20 20 20 20 20 6e 6f 74 20 6f 6e 65 AD;.     not one
190  20 62 69 74 20 77 61 73 20 64 72 6f 70 70 65 64  bit was dropped
1a0  3b 20 6e 6f 20 63 68 65 63 6b 73 75 6d 20 77 61 ; no checksum wa
1b0  73 20 62 61 64 2e 0a 20 20 20 41 74 20 6c 61 73 s bad..   At las
1c0  74 20 68 65 20 77 61 73 20 66 69 6e 69 73 68 65 t he was finishe
1d0  64 20 61 6e 64 20 77 65 61 72 69 6c 79 20 73 69 d and wearily si
1e0  67 68 65 64 0a 20 20 20 20 20 61 6e 64 20 74 75 ghed.     and tu
1f0  72 6e 65 64 20 74 6f 20 65 78 70 6c 61 69 6e 20 rned to explain
```

We continue this (again, it's the next block and cluster) and we find 0x0f51 as the cluster number (highlighted in green above). Here is that block:

```
Block 3944 (0x0f68)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000   77 68 79 20 74 68 65 20 73 79 73 74 65 6d 20 68 why the system h
010   61 64 20 64 69 65 64 2e 0a 20 20 20 49 20 74 77 ad died..   I tw
020   69 73 74 65 64 20 6d 79 20 66 69 6e 67 65 72 73 isted my fingers
030   20 61 6e 64 20 63 6f 75 6e 74 65 64 20 74 6f 20  and counted to
040   74 65 6e 3b 0a 20 20 20 20 20 61 6e 20 6f 66 66 ten;.     an off
050   2d 62 79 2d 6f 6e 65 20 69 6e 64 65 78 20 68 61 -by-one index ha
060   64 20 64 6f 6e 65 20 69 74 20 61 67 61 69 6e 2e d done it again.
070   2e 2e 0a 0a 20 20 20 56 69 6e 74 20 43 65 72 66 ....   Vint Cerf
080   0a 20 20 20 44 65 63 65 6d 62 65 72 20 31 39 38 .   December 198
090   35 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5...............
0a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Lastly, we look at the FAT entry for this block/cluster (highlighted in black). This time the entry is 0xffff, which indicates that there are no more blocks in the file. We have finished!

Conclusion

If you've managed to get this far (and understood it all) you have a good working understanding of the 16-bit FAT file system. You should be able to analyse a disk, and see if it is corrupted. You may even be able to repair it!