OPERATING SYSTEMS TUTORIAL 9

University of Victoria

Content

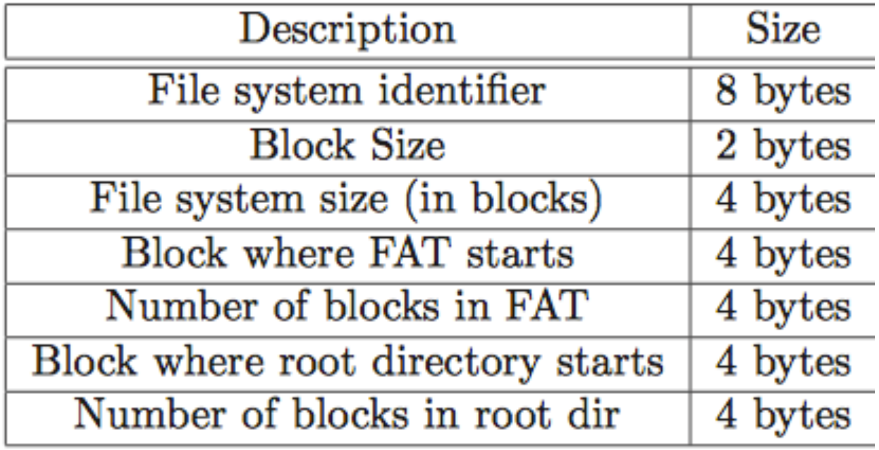
- Objectives of P3 - FAT (specification)
- Hints on programming
- 3 Exercise Questions
- DEMO

File System Specifications

3 major components of an FAT File System:

- **Super Block,**
- **File Allocation Table**
- **Directory Structure.**

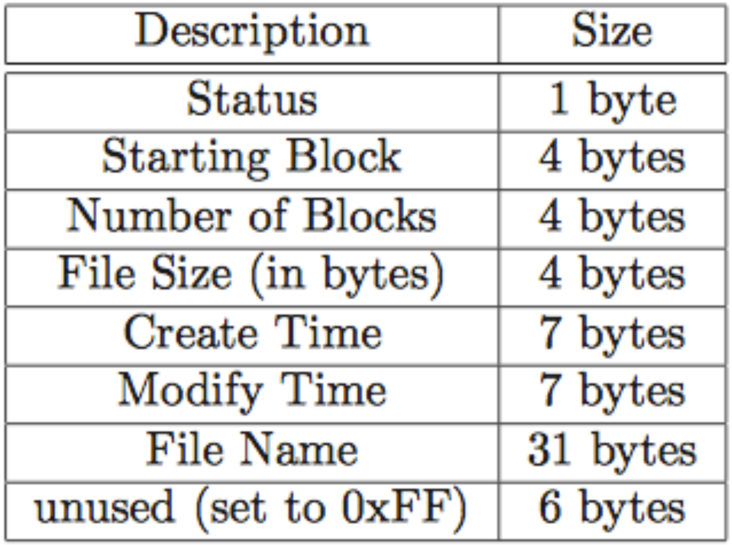
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

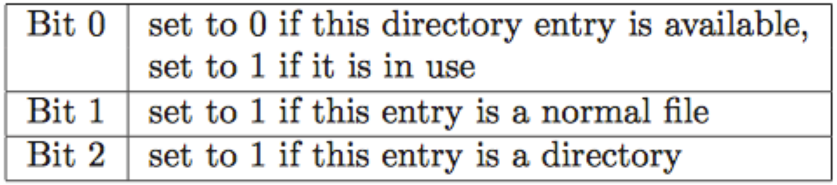
The first block (**512 B**) is reserved to contain information about the file system

Directory Entry

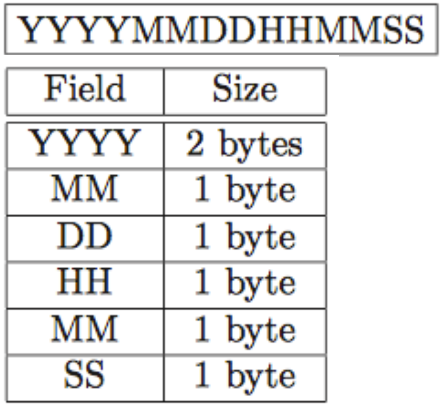


Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Takes up **64 B**, which implies there are 8 directory entries per **512 B block**



Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory



YYYYMMDDHHMMSS

Field	Size
YYYY	2 bytes
MM	1 byte
DD	1 byte
HH	1 byte
MM	1 byte
SS	1 byte

Objectives

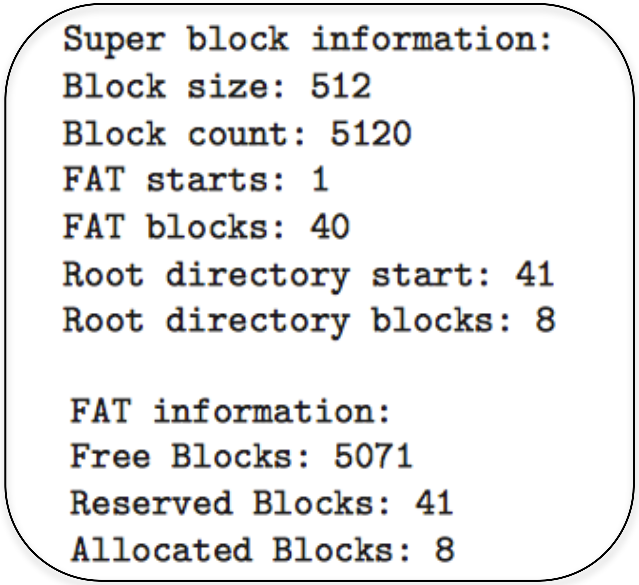
Implementing utilities that perform operations on a File System (e.g. FAT)

Since we are dealing with **Binary Data (0 | 1),** functions intended for string manipulation such as **strcpy()** do NOT work, and it is necessary to use functions intended for binary data such as **memcpy().**

Part 1 (3 points)

Read the file system **Super Block** and use the information to read the FAT.

./diskinfo test.img

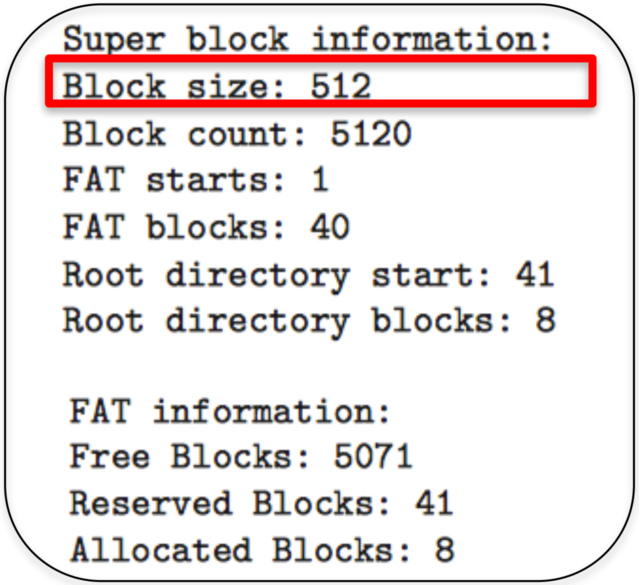


```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

Please Use the Same Output Format
In Your Own Code.

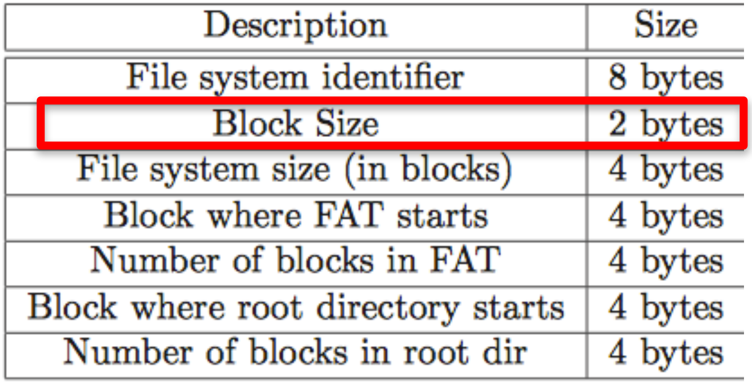
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

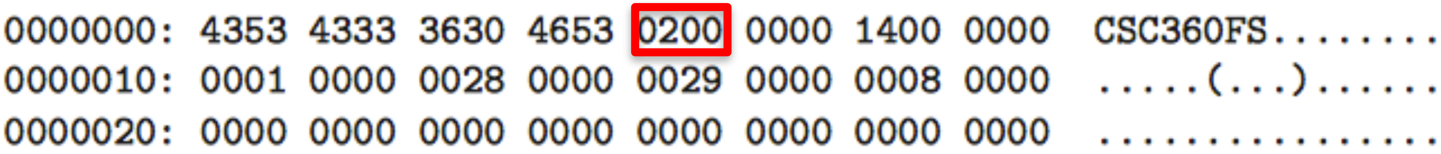
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

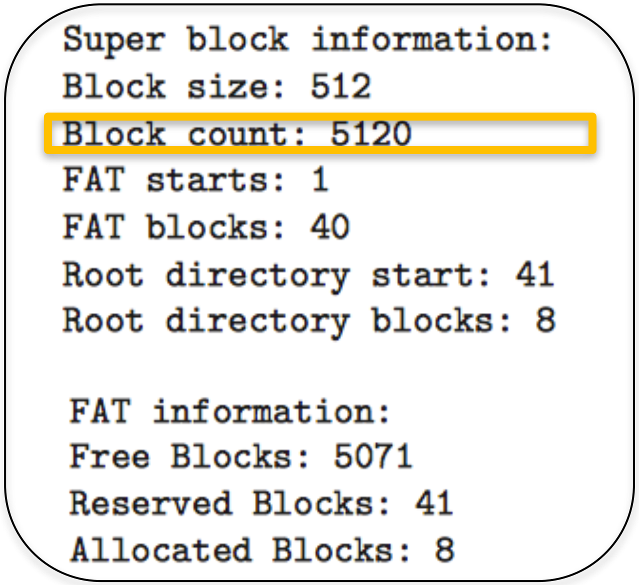
SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

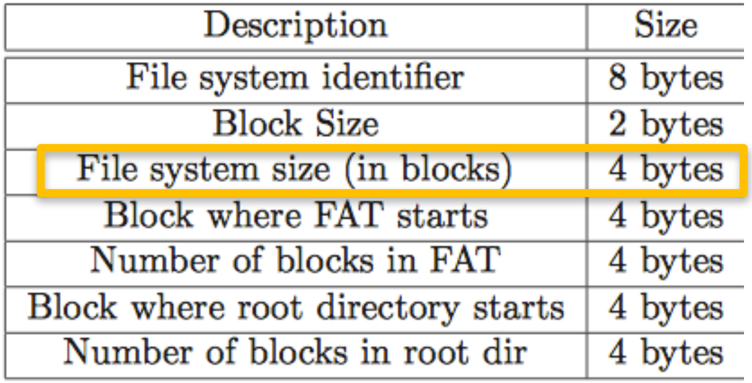
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

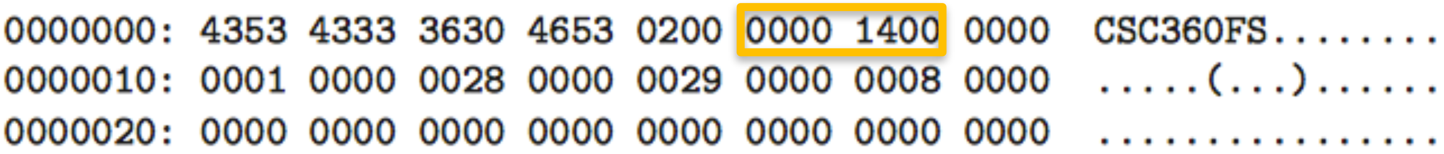
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

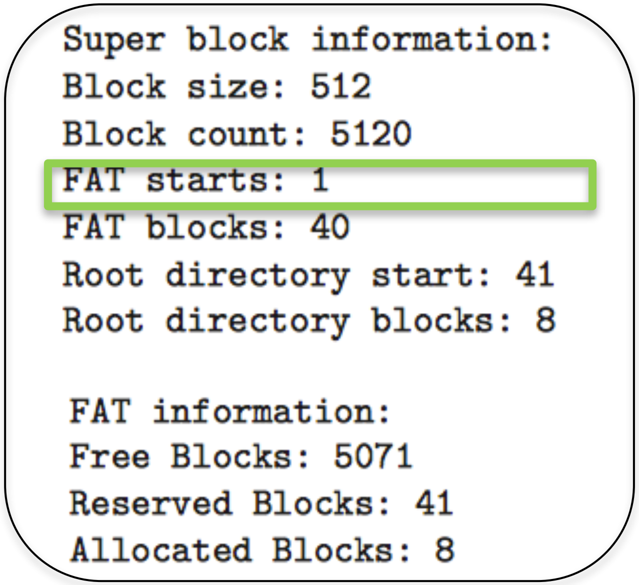
SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

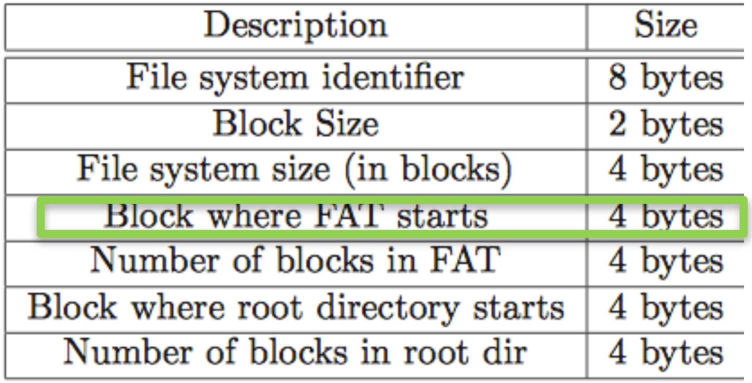
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

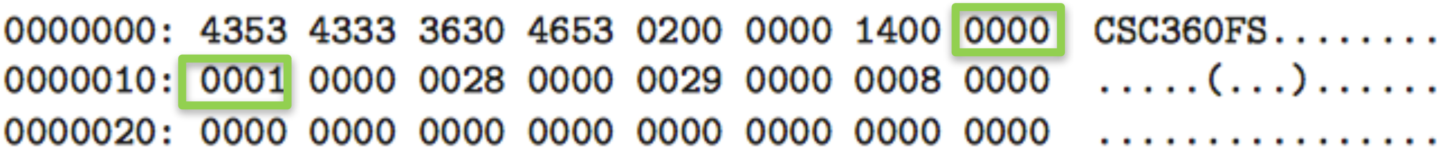
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

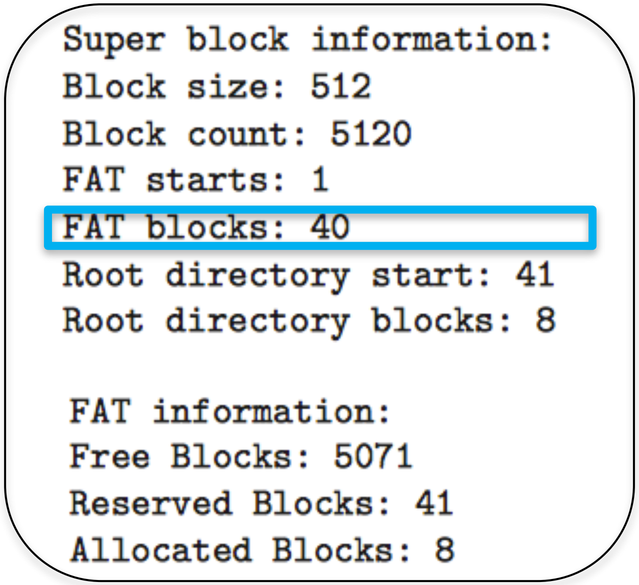
SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

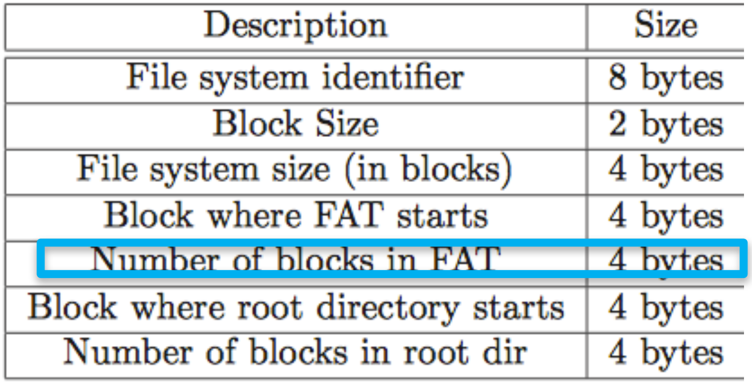
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

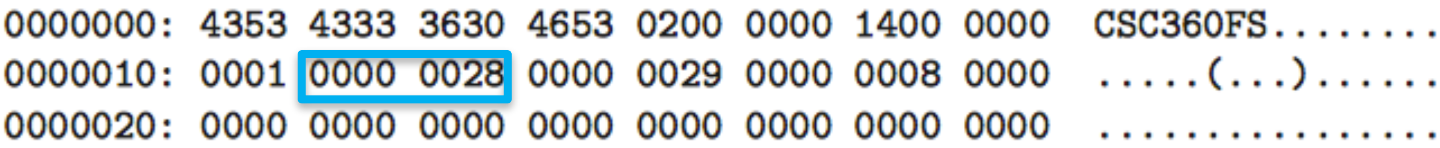
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

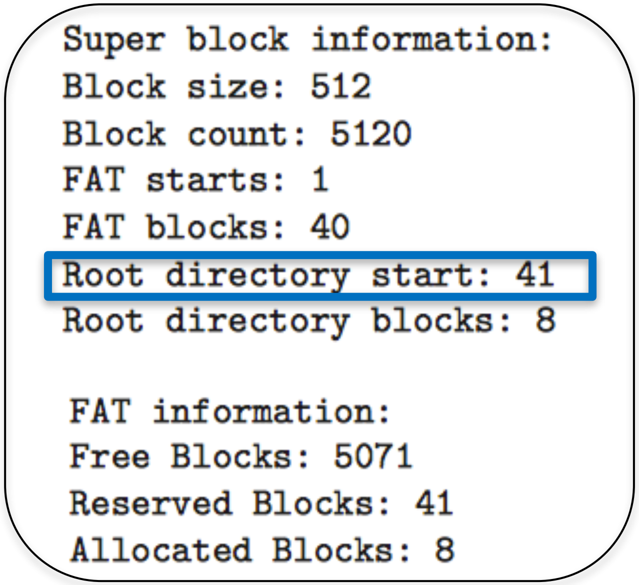
SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

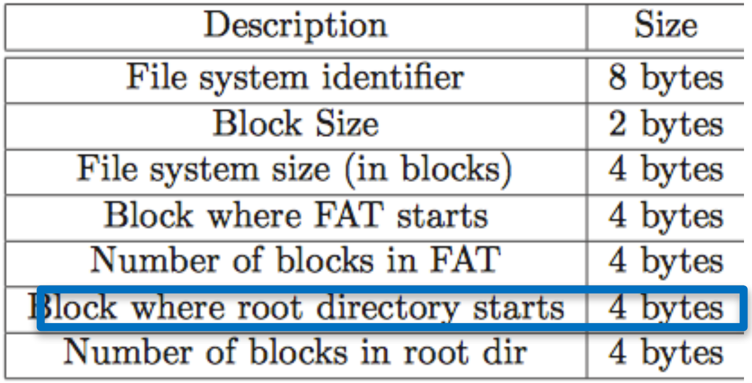
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

 FAT information:
 Free Blocks: 5071
 Reserved Blocks: 41
 Allocated Blocks: 8
```

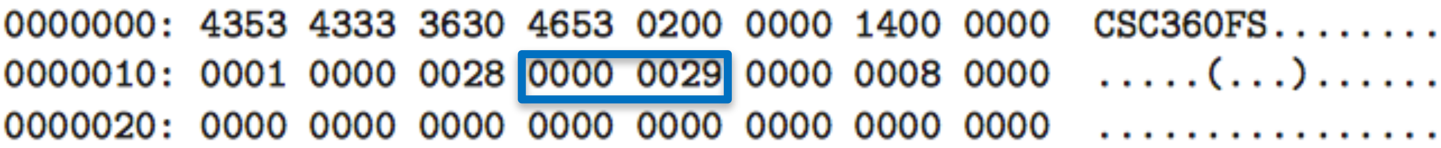
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

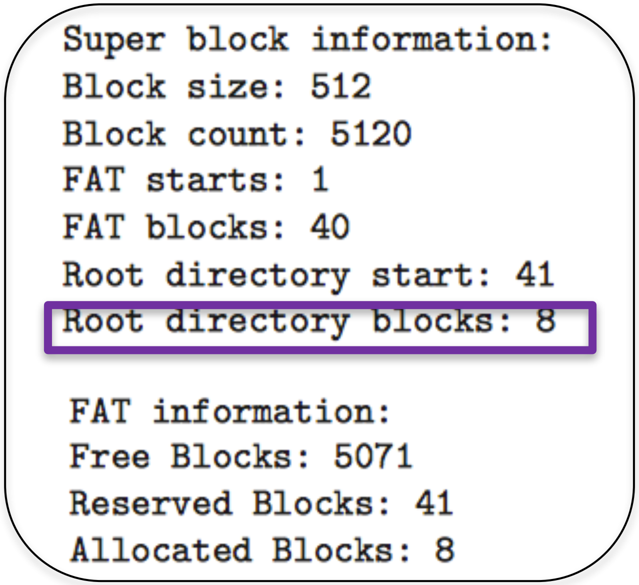
SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

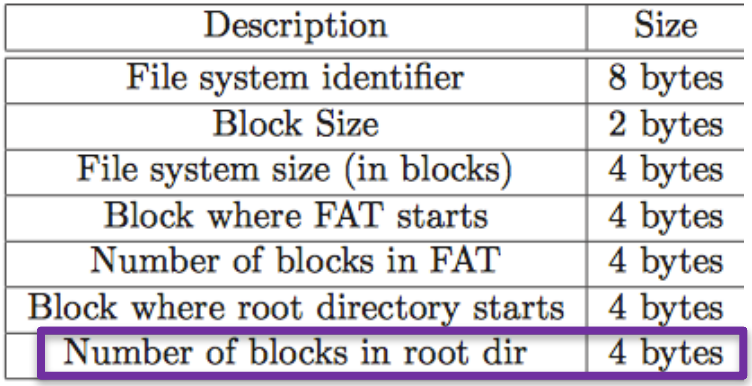
diskinfo

./diskinfo test.img



```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

FAT information:
Free Blocks: 5071
Reserved Blocks: 41
Allocated Blocks: 8
```

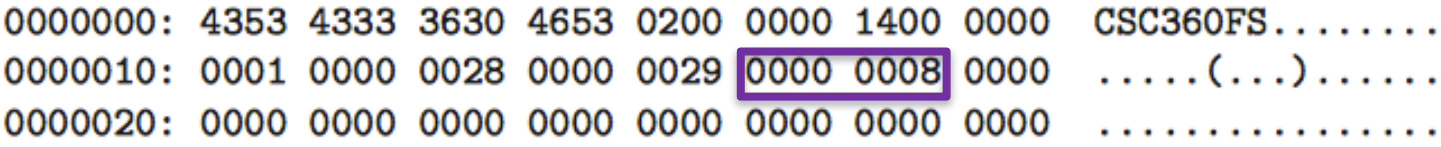
Super Block



Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

SUPERBLOCK:



```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

diskinfo

./diskinfo test.img

```
Super block information:
Block size: 512
Block count: 5120
FAT starts: 1
FAT blocks: 40
Root directory start: 41
Root directory blocks: 8

FAT information:
Free Blocks: 5071
Reserved Blocks: 41
Allocated Blocks: 8
```

Super Block

Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes

The first block (**512 B**) is reserved to contain information about the file system

Value of FAT entry

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002– 0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

SUPERBLOCK:

```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Objectives

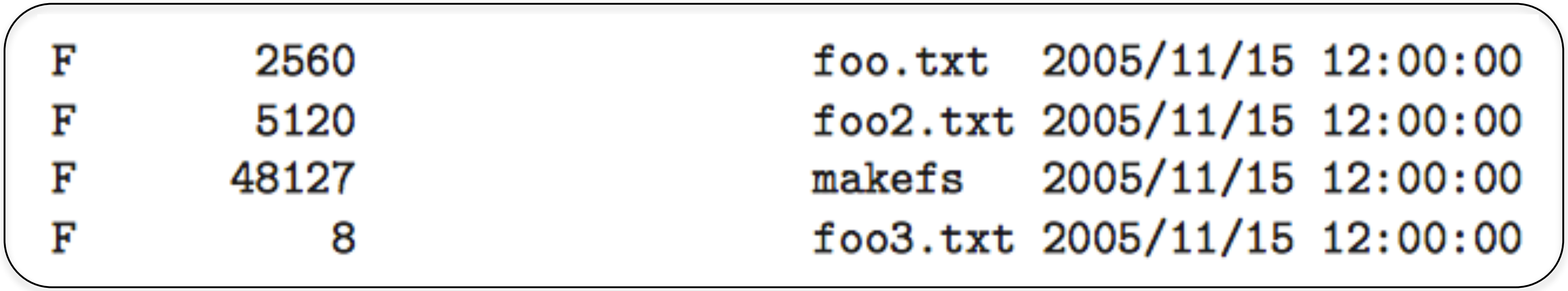
Implementing utilities that perform operations on a File System (e.g. FAT)

Since we are dealing with **Binary Data (0 | 1),** functions intended for string manipulation such as **strcpy()** do NOT work, and it is necessary to use functions intended for binary data such as **memcpy().**

Part 2 (3 points)

Displays the contents of the root directory or a given sub-directory in the file system.

./disklist test.img /sub_dir

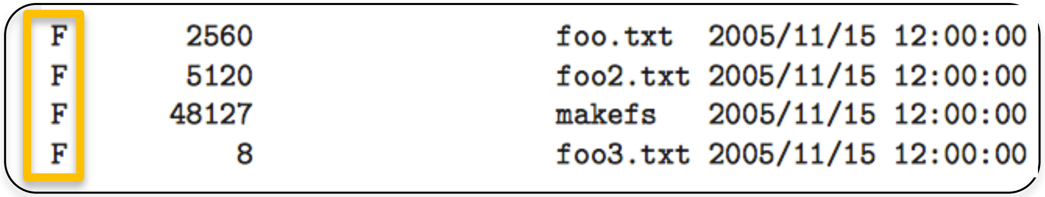


```
F       2560                   foo.txt  2005/11/15 12:00:00
F       5120                   foo2.txt 2005/11/15 12:00:00
F      48127                   makefs   2005/11/15 12:00:00
F          8                   foo3.txt 2005/11/15 12:00:00
```

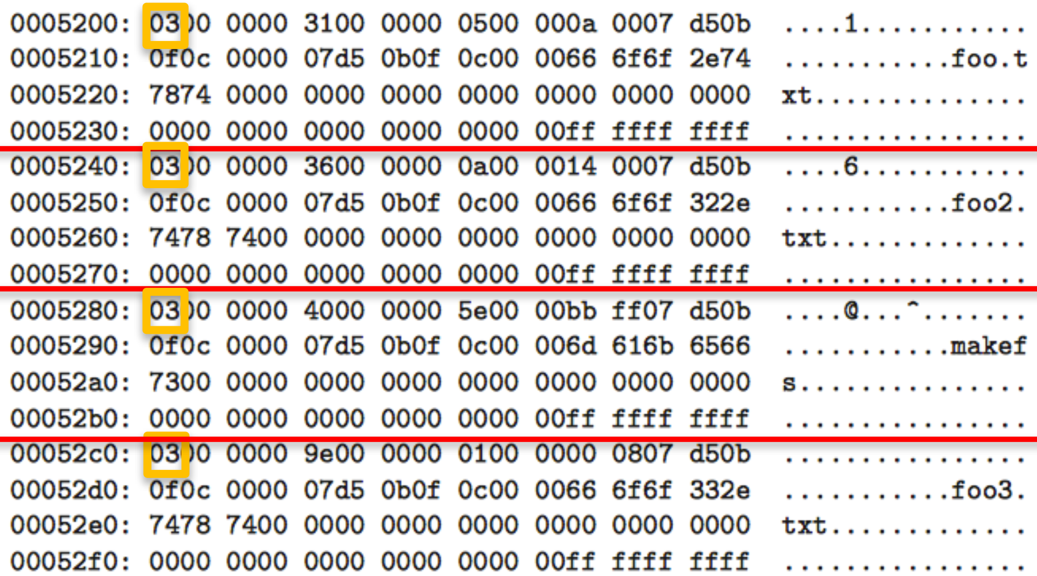
Please Use the Same Output Format
In Your Own Code.

disklist

./disklist test.img /subdir

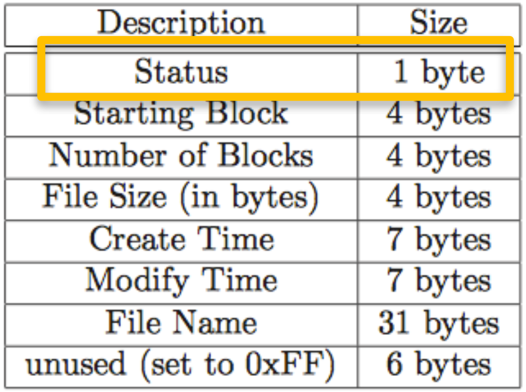


```
F        2560                foo.txt  2005/11/15 12:00:00
F        5120                foo2.txt 2005/11/15 12:00:00
F       48127                makefs   2005/11/15 12:00:00
F           8                foo3.txt 2005/11/15 12:00:00
```



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

Directory Entry

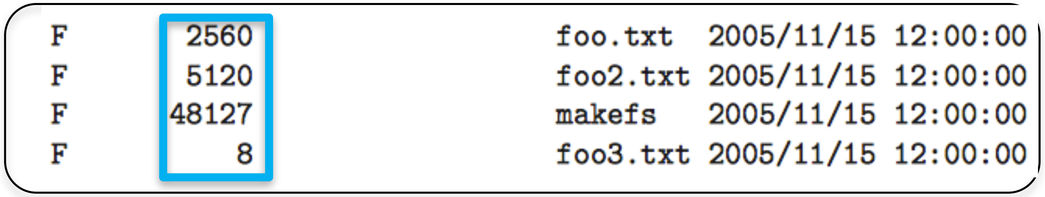


Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

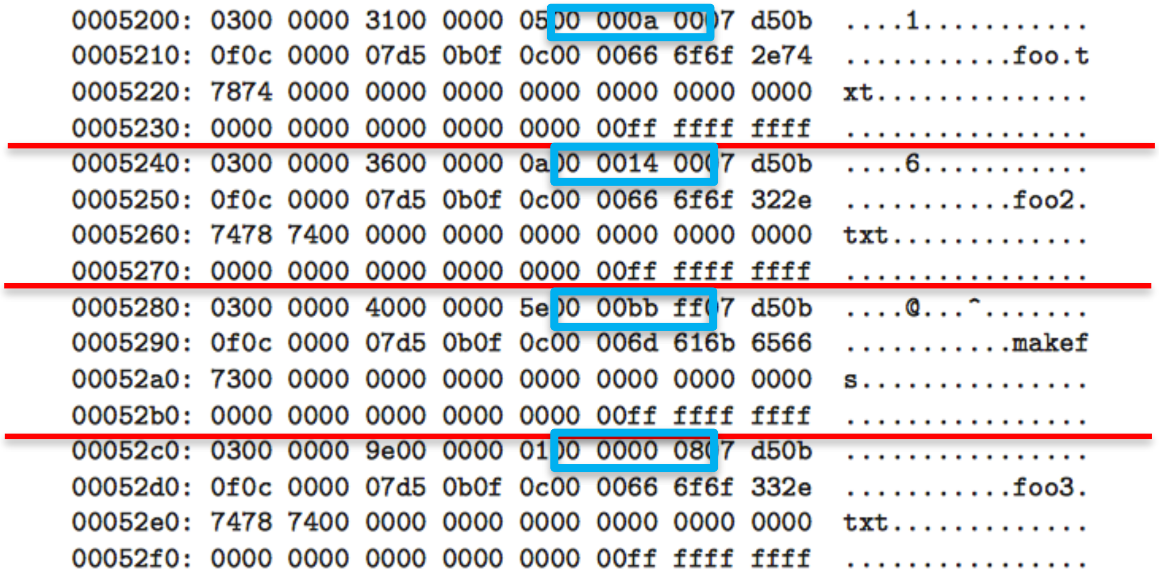
Takes up **64 B**, which implies there are 8 directory entries per **512 B block**

disklist

./disklist test.img /subdir

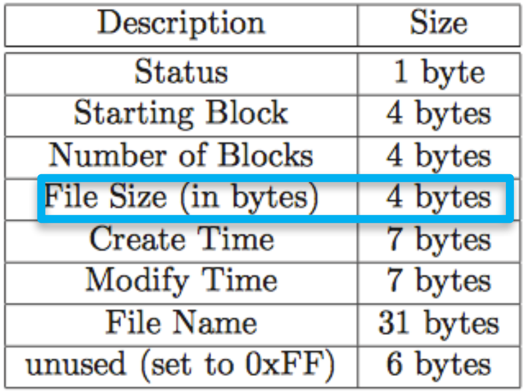


```
F        2560                foo.txt  2005/11/15 12:00:00
F        5120                foo2.txt 2005/11/15 12:00:00
F       48127                makefs   2005/11/15 12:00:00
F           8                foo3.txt 2005/11/15 12:00:00
```



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

Directory Entry

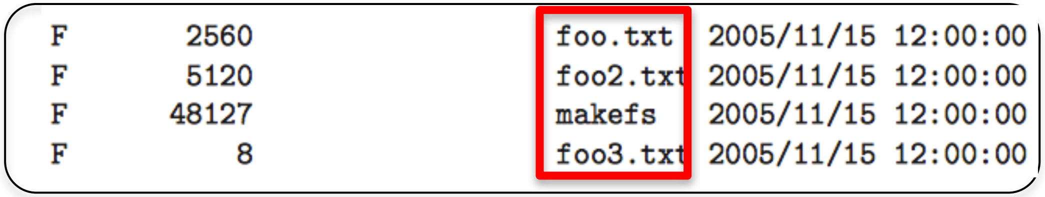


Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

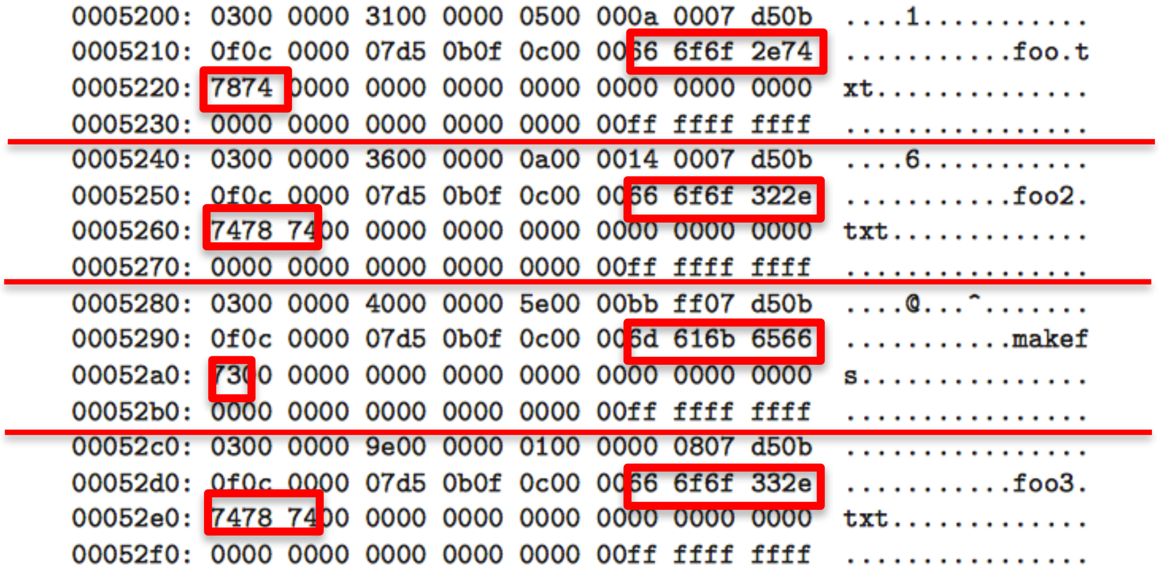
Takes up **64 B**, which implies there are 8 directory entries per **512 B block**

disklist

./disklist test.img /subdir

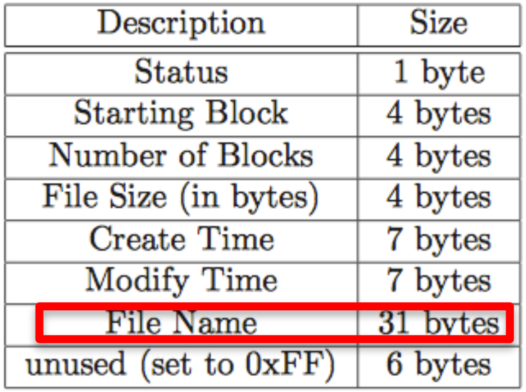


```
F        2560                 foo.txt  2005/11/15 12:00:00
F        5120                 foo2.txt 2005/11/15 12:00:00
F       48127                 makefs   2005/11/15 12:00:00
F           8                 foo3.txt 2005/11/15 12:00:00
```



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

Directory Entry

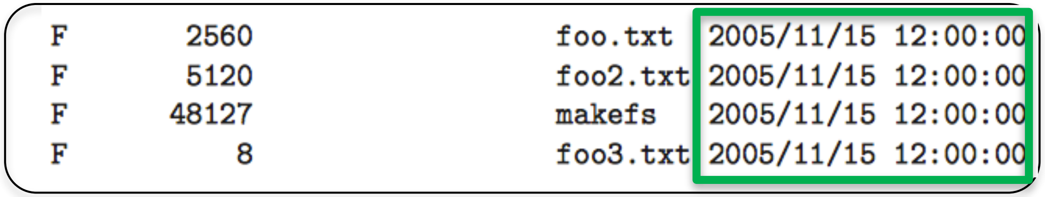


Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

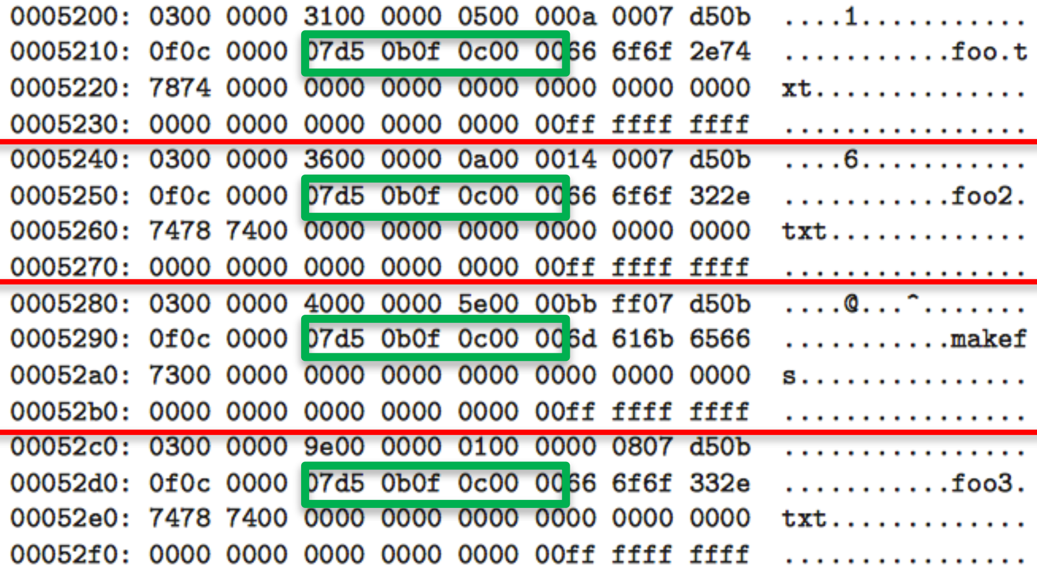
Takes up **64 B**, which implies there are 8 directory entries per **512 B block**

disklist

./disklist test.img /subdir

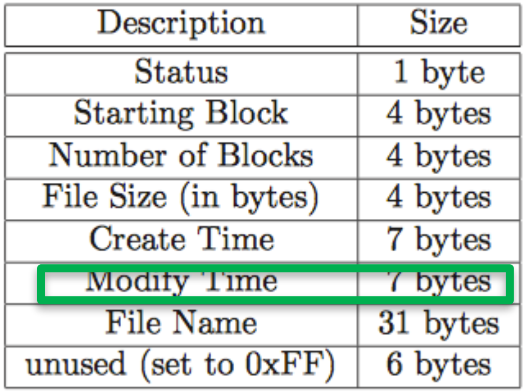


```
F       2560              foo.txt 2005/11/15 12:00:00
F       5120              foo2.txt 2005/11/15 12:00:00
F      48127              makefs  2005/11/15 12:00:00
F          8              foo3.txt 2005/11/15 12:00:00
```



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

Directory Entry



Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Takes up **64 B**, which implies there are 8 directory entries per **512 B block**

Objectives

Implementing utilities that perform operations on a File System (e.g. FAT)

Since we are dealing with **Binary Data (0 | 1),** functions intended for string manipulation such as **strcpy()** do NOT work, and it is necessary to use functions intended for binary data such as **memcpy().**

Part 3 (3 points)

Write a program that copies a file from the file system to the current directory in your operating system (Linux). If the specified file is not found in the root directory (of test.img) or a given subdirectory of the file system, you should output the message **File not found** and exit.

```
./diskget  test.img  /sub_dir/foo2.txt   foo.txt
```

Objectives

Implementing utilities that perform operations on a File System (e.g. FAT)

Since we are dealing with **Binary Data (0 | 1),** functions intended for string manipulation such as **strcpy()** do NOT work, and it is necessary to use functions intended for binary data such as **memcpy().**

Part 4 (3 points)

Write a program that copies a file from the current directory into the file system, at the root directory or a given sub-directory. If the specified file is not found, you should output the message **File not found** on a single line and exit.

```
./diskput  test.img  foo.txt   /sub_dir/foo3.txt
```

But file system size does NOT change

Objectives

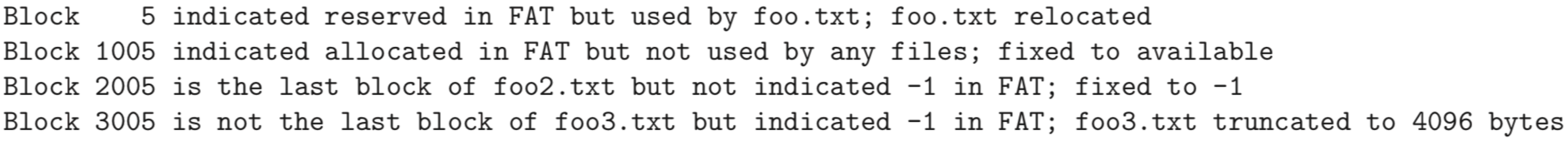
Implementing utilities that perform operations on a File System (e.g. FAT)

Since we are dealing with **Binary Data (0 | 1),** functions intended for string manipulation such as **strcpy()** do NOT work, and it is necessary to use functions intended for binary data such as **memcpy().**

Part 5 (3 points)

Go through the disk image according to the file system specification, including the super block, FDT, FAT and data blocks, find inconsistent information among them and fix these issues when possible.

./diskfix test.img



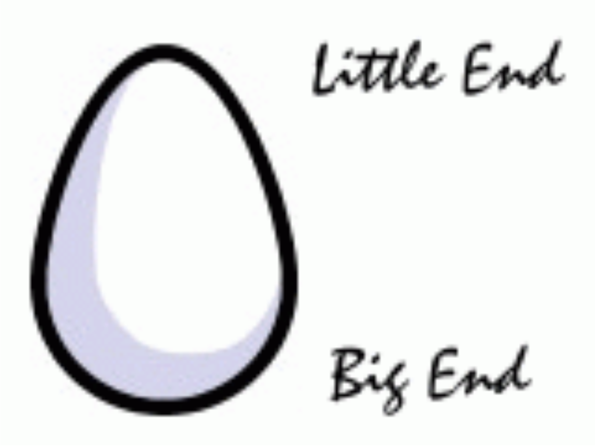
```
Block    5 indicated reserved in FAT but used by foo.txt; foo.txt relocated
Block 1005 indicated allocated in FAT but not used by any files; fixed to available
Block 2005 is the last block of foo2.txt but not indicated -1 in FAT; fixed to -1
Block 3005 is not the last block of foo3.txt but indicated -1 in FAT; foo3.txt truncated to 4096 bytes
```

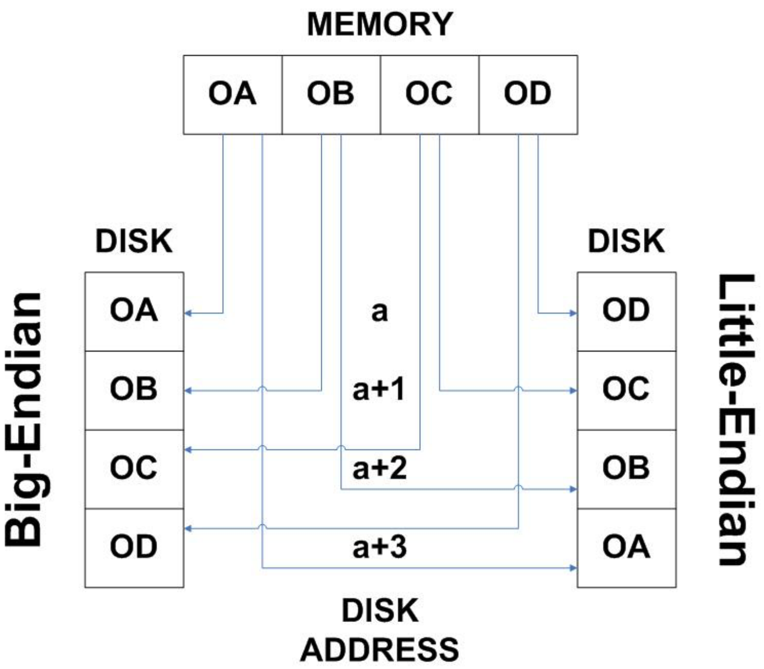
Generating multiple binaries from a single source

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>
#include <limits.h>
#include <assert.h>
#include <time.h>

int main(int argc, char* argv[])
{
#if defined(PART1)
        diskinfo(argc, argv);
#elif defined(PART2)
        disklist(argc, argv);
#elif defined(PART3)
        diskget(argc, argv);
#elif defined(PART4)
        diskput(argc,argv);
#elif defined(PART5)
        diskfix(argc,argv);
#else
#       error “PART[12345] must be defined"
#endif
        return 0;
}
```

```
.PHONY all:
all:
        gcc -Wall -D PART1 parts.c -o diskinfo
        gcc -Wall -D PART2 parts.c -o disklist
        gcc -Wall -D PART3 parts.c -o diskget
        gcc -Wall -D PART4 parts.c -o diskput
        gcc -Wall -D PART5 parts.c -o diskfix

.PHONY clean:
clean:
        -rm diskinfo disklist diskget diskput diskfix
```

Byte Ordering





Consider the large integer **0xDEADBEEF**

Stored in memory as

Little Endian

EF BE AD DE

DE AD BE EF

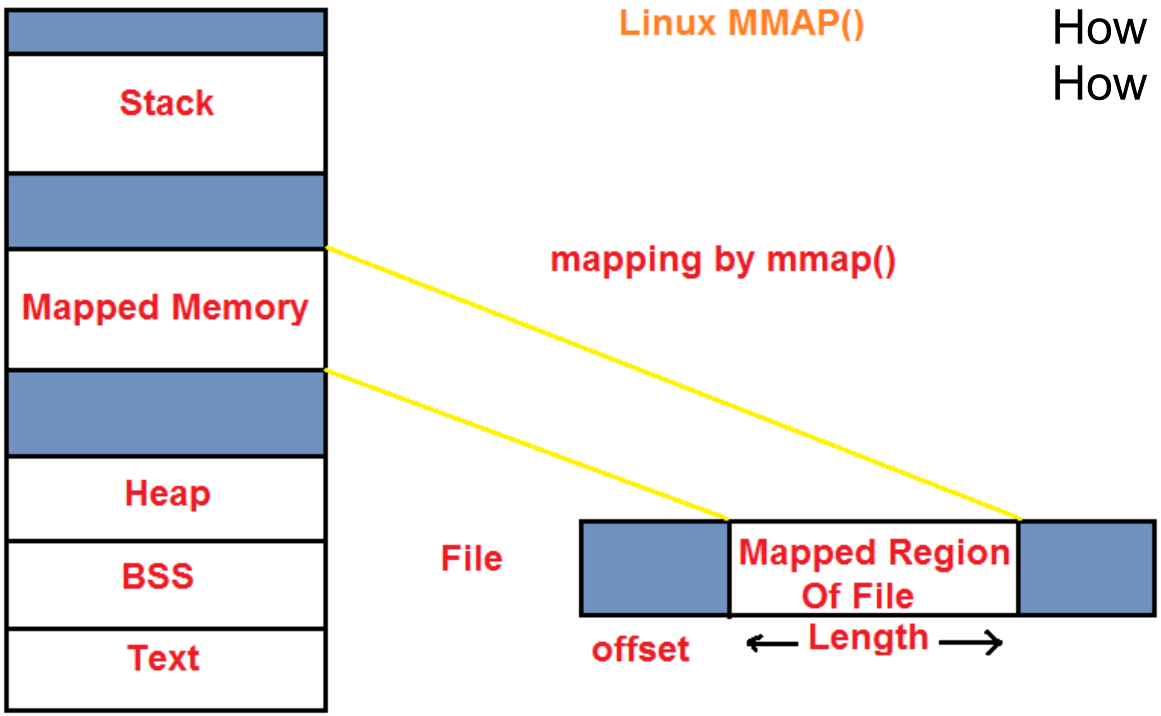
Big Endian

Hints on programming

mmap:

```
void *mmap(void *addr, size_t length, int prot,
int flags, int fd, off_t offset);
```

http://man7.org/linux/man-pages/man2/mmap.2.html



How to know the size of input file?
How to get file descriptor?

http://www.tutorialsdaddy.com/courses/linux-device-driver/lessons/mmap/

Useful Structures

```
// Super block
struct __attribute__((__packed__)) superblock_t {
  uint8_t   fs_id [8];
  uint16_t block_size;
  uint32_t file_system_block_count;
  uint32_t fat_start_block;
  uint32_t fat_block_count;
  uint32_t root_dir_start_block;
  uint32_t root_dir_block_count;
};
```

```
// Time and date entry
struct __attribute__((__packed__)) dir_entry_timedate_t {
  uint16_t year;
  uint8_t  month;
  uint8_t  day;
  uint8_t  hour;
  uint8_t  minute;
  uint8_t  second;
};
```

```
// Directory entry
struct __attribute__((__packed__)) dir_entry_t {
  uint8_t                    status;
  uint32_t                   starting_block;
  uint32_t                   block_count;
  uint32_t                   size;
  struct   dir_entry_timedate_t create_time;
  struct   dir_entry_timedate_t modify_time;
  uint8_t                    filename[31];
  uint8_t                    unused[6];
};
```

"__attribute__((__packed__))" is important and needed, otherwise, compiler optimizes for byte alignment

A An Exercise

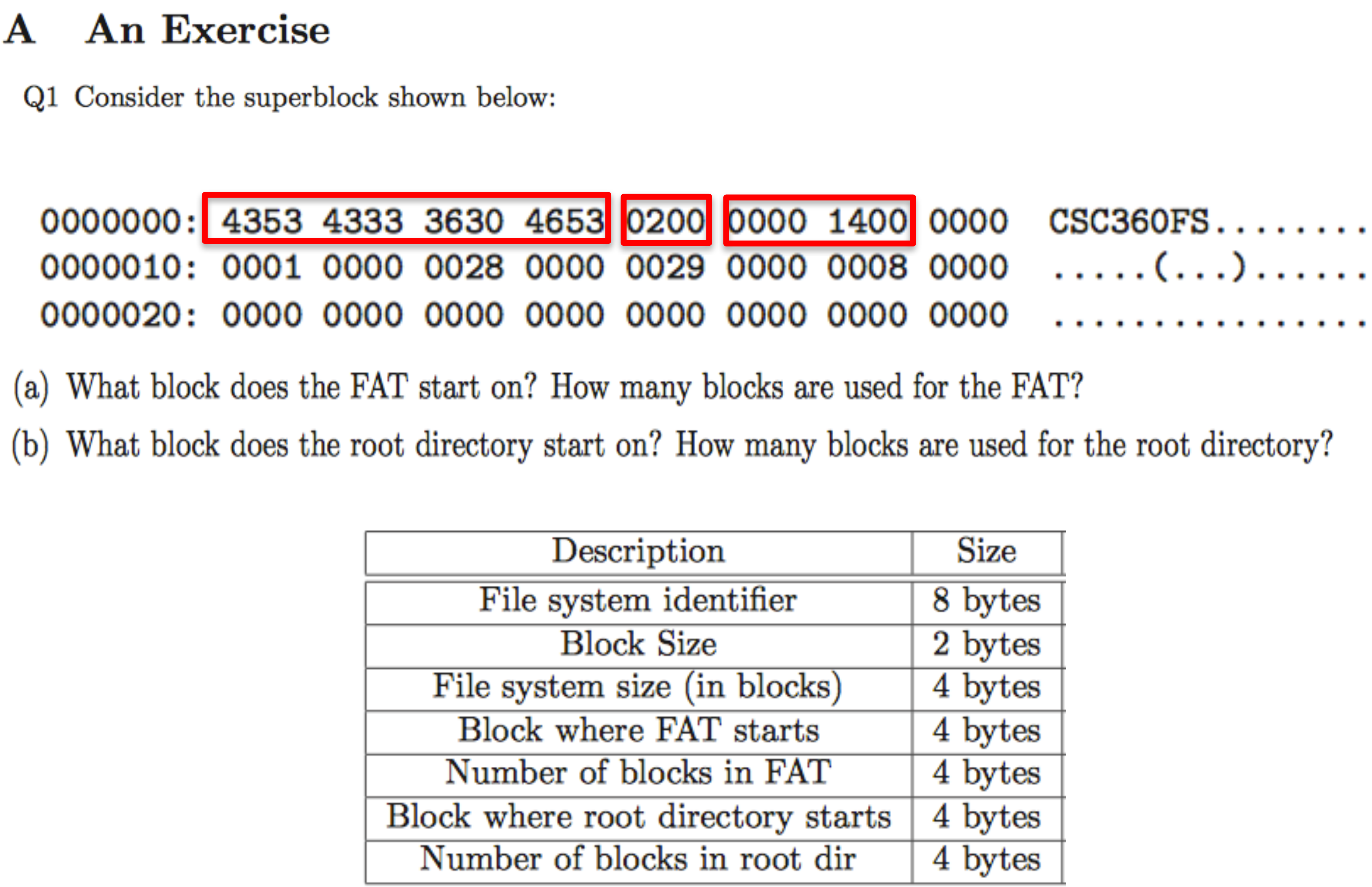
Q1 Consider the superblock shown below:

```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

(a) What block does the FAT start on? How many blocks are used for the FAT?

(b) What block does the root directory start on? How many blocks are used for the root directory?

Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes



A An Exercise

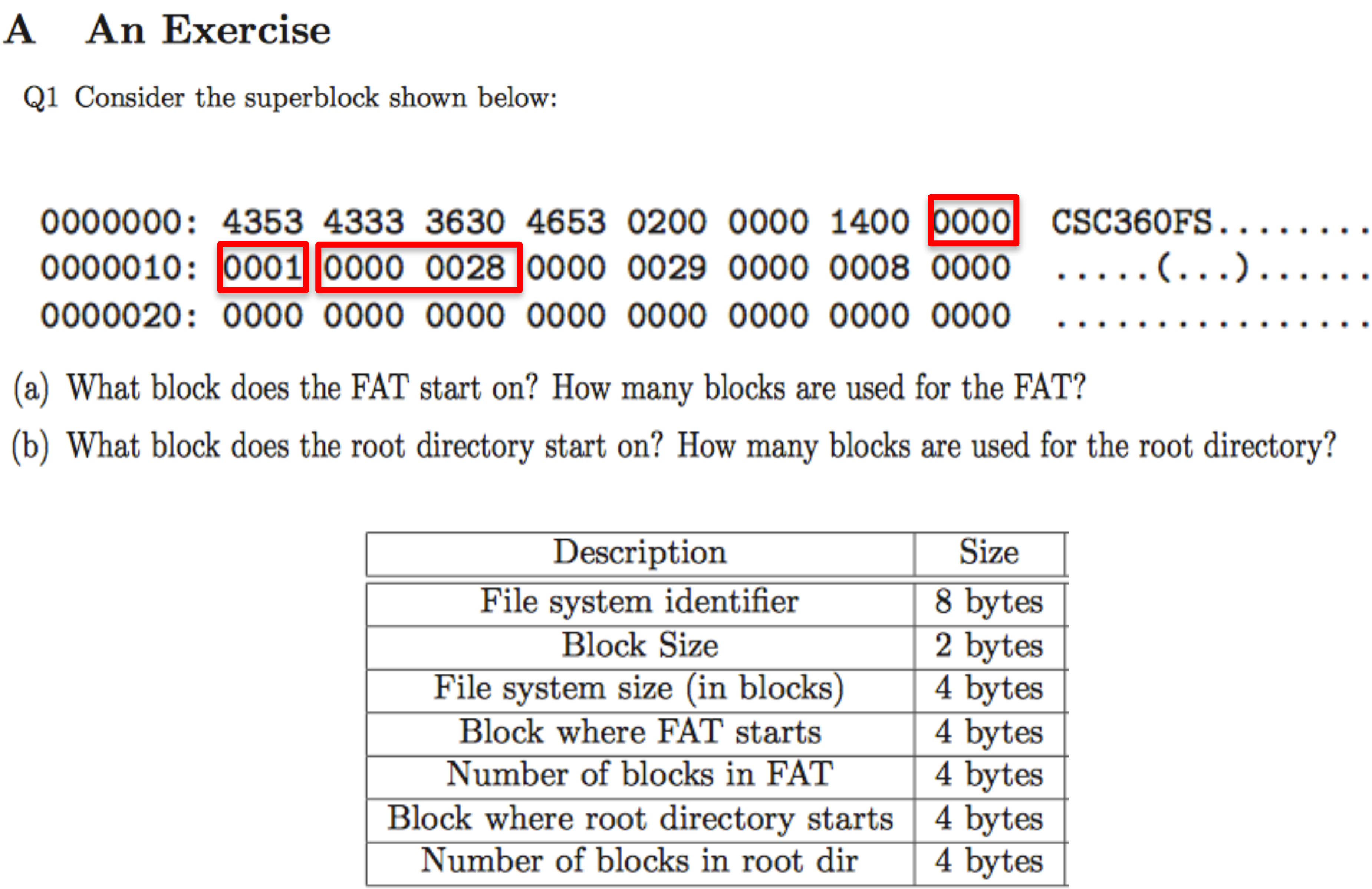
Q1 Consider the superblock shown below:

```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

(a) What block does the FAT start on? How many blocks are used for the FAT?

(b) What block does the root directory start on? How many blocks are used for the root directory?

Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes



A An Exercise

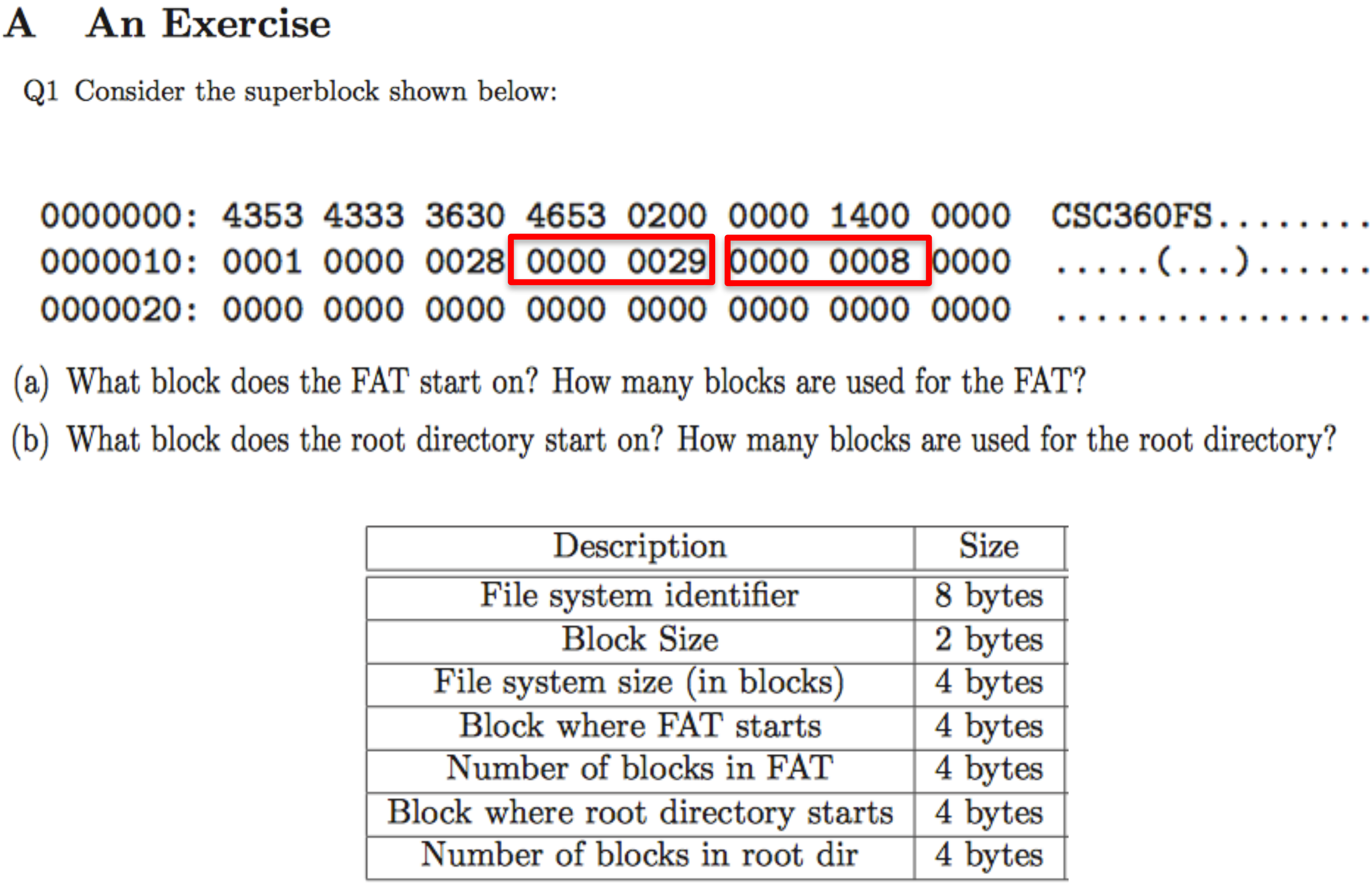
Q1 Consider the superblock shown below:

```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

(a) What block does the FAT start on? How many blocks are used for the FAT?

(b) What block does the root directory start on? How many blocks are used for the root directory?

Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes



A An Exercise

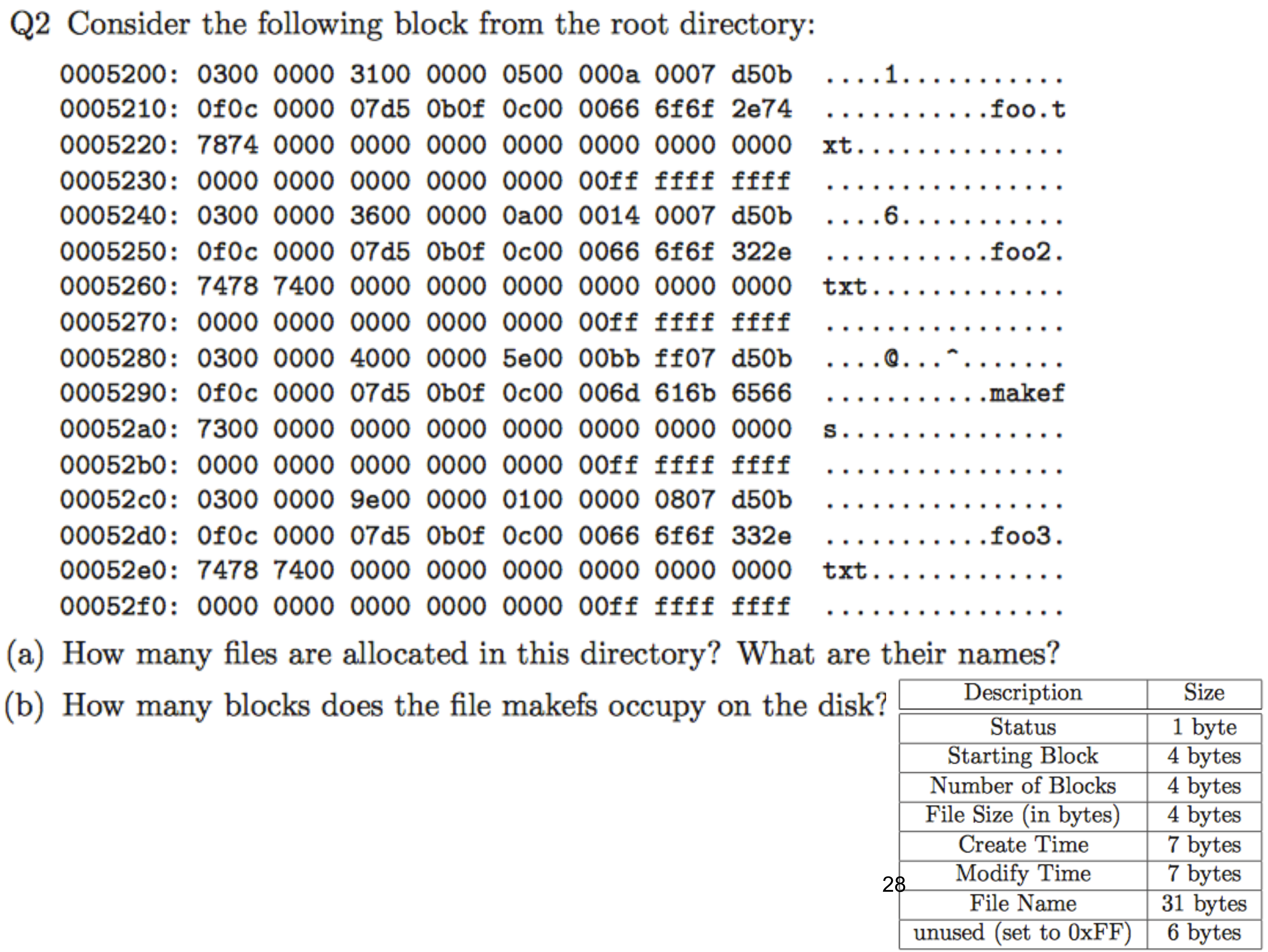
Q1 Consider the superblock shown below:

```
0000000: 4353 4333 3630 4653 0200 0000 1400 0000  CSC360FS........
0000010: 0001 0000 0028 0000 0029 0000 0008 0000  .....(...)......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

(a) What block does the FAT start on? How many blocks are used for the FAT?

(b) What block does the root directory start on? How many blocks are used for the root directory?

Description	Size
File system identifier	8 bytes
Block Size	2 bytes
File system size (in blocks)	4 bytes
Block where FAT starts	4 bytes
Number of blocks in FAT	4 bytes
Block where root directory starts	4 bytes
Number of blocks in root dir	4 bytes



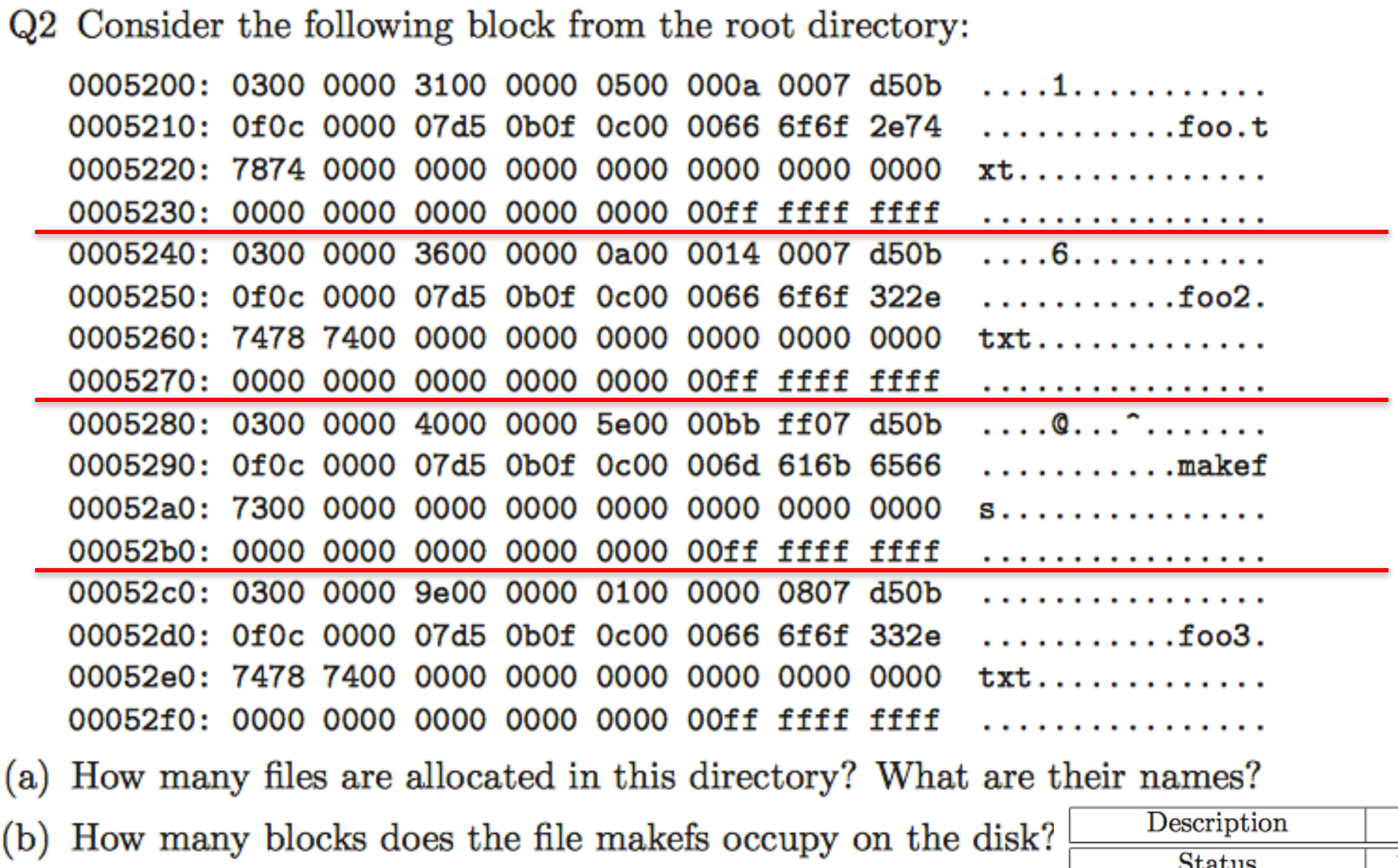
Q2 Consider the following block from the root directory:

```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes



Q2 Consider the following block from the root directory:

```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

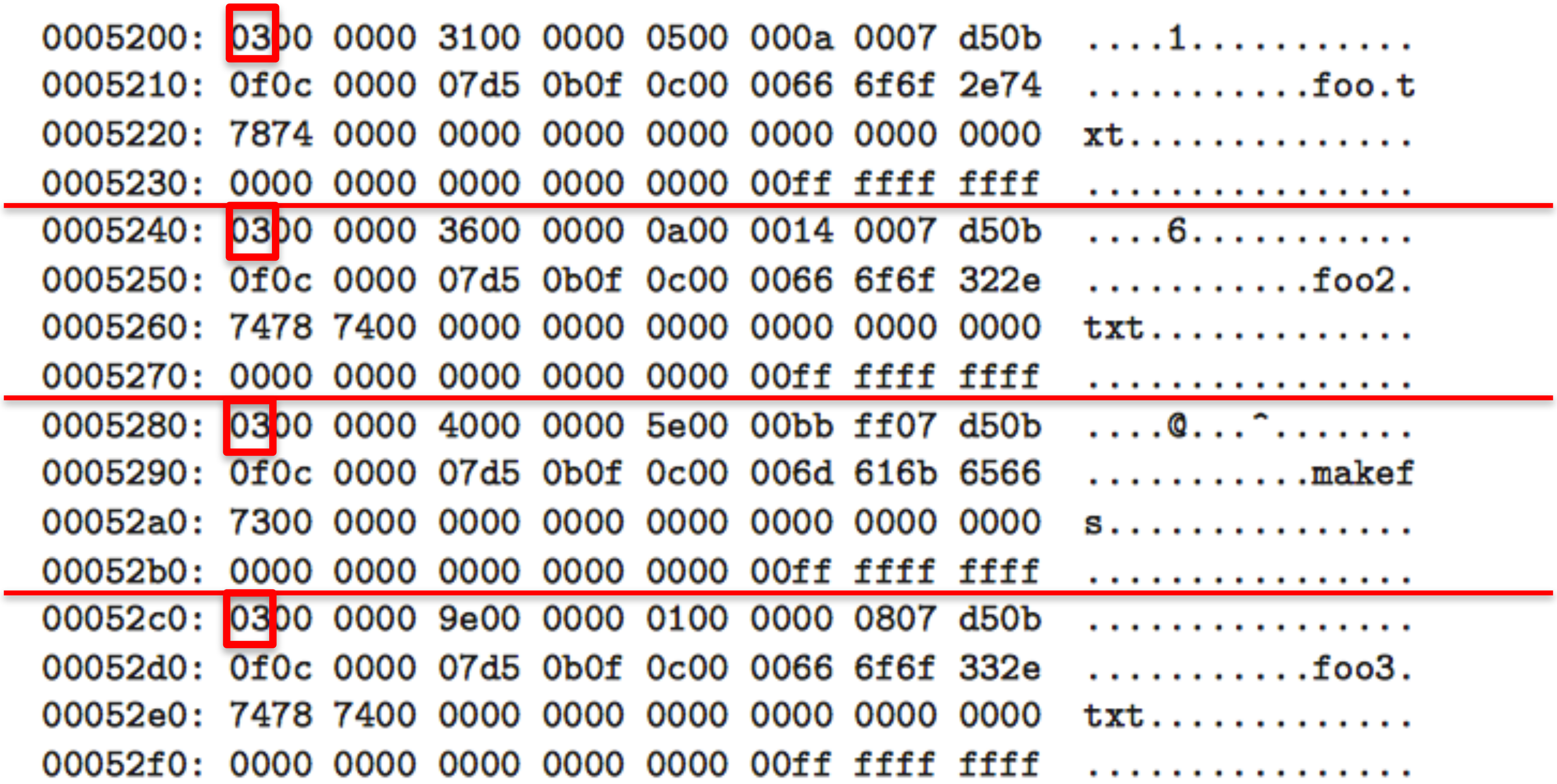
(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Each directory entry takes 64 B

Q2 Consider the following block from the root directory:



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ............foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ............foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ............makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ............foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

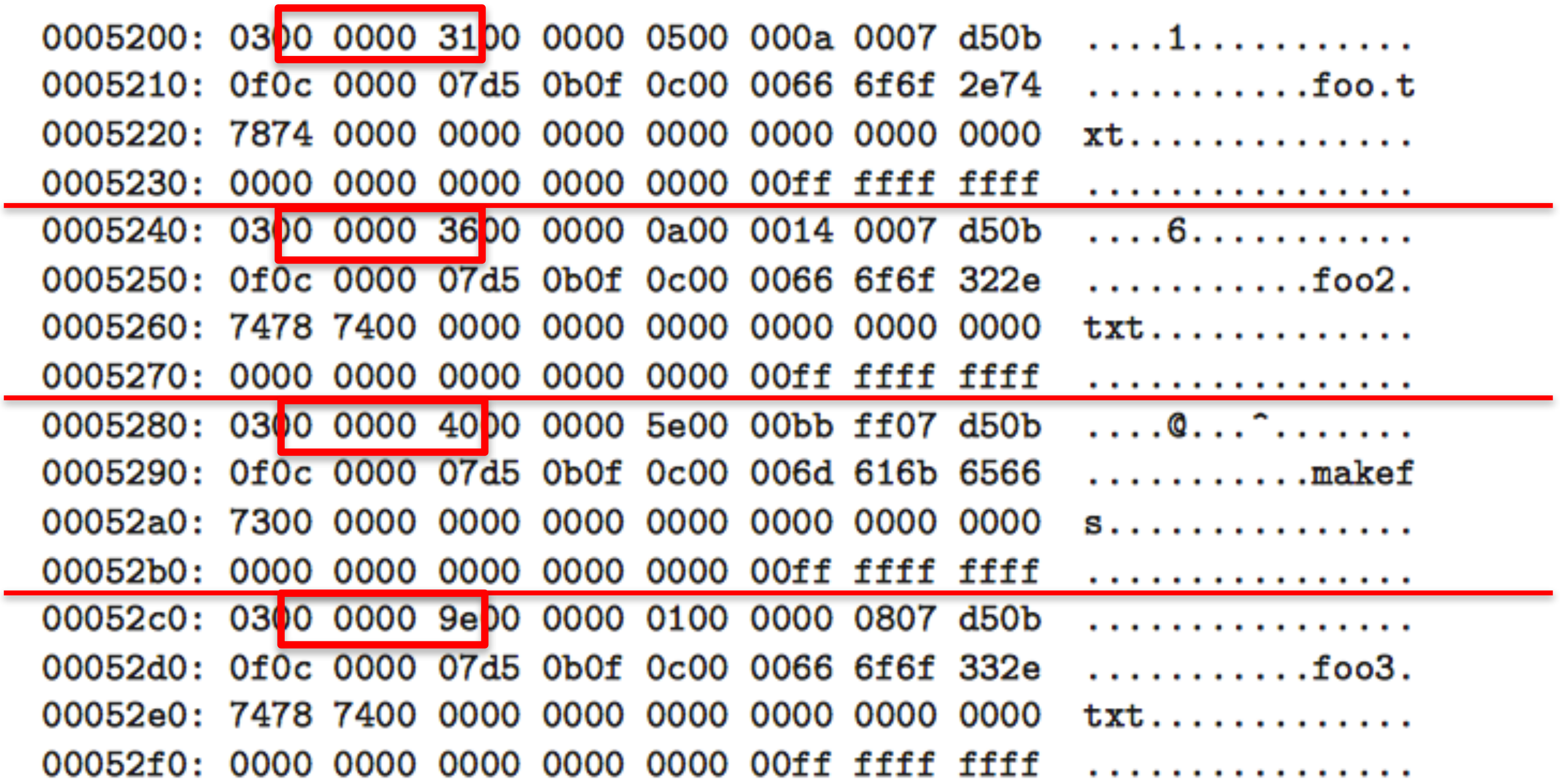
Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

64 B

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Q2 Consider the following block from the root directory:



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ............foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ............foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ............makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ............foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

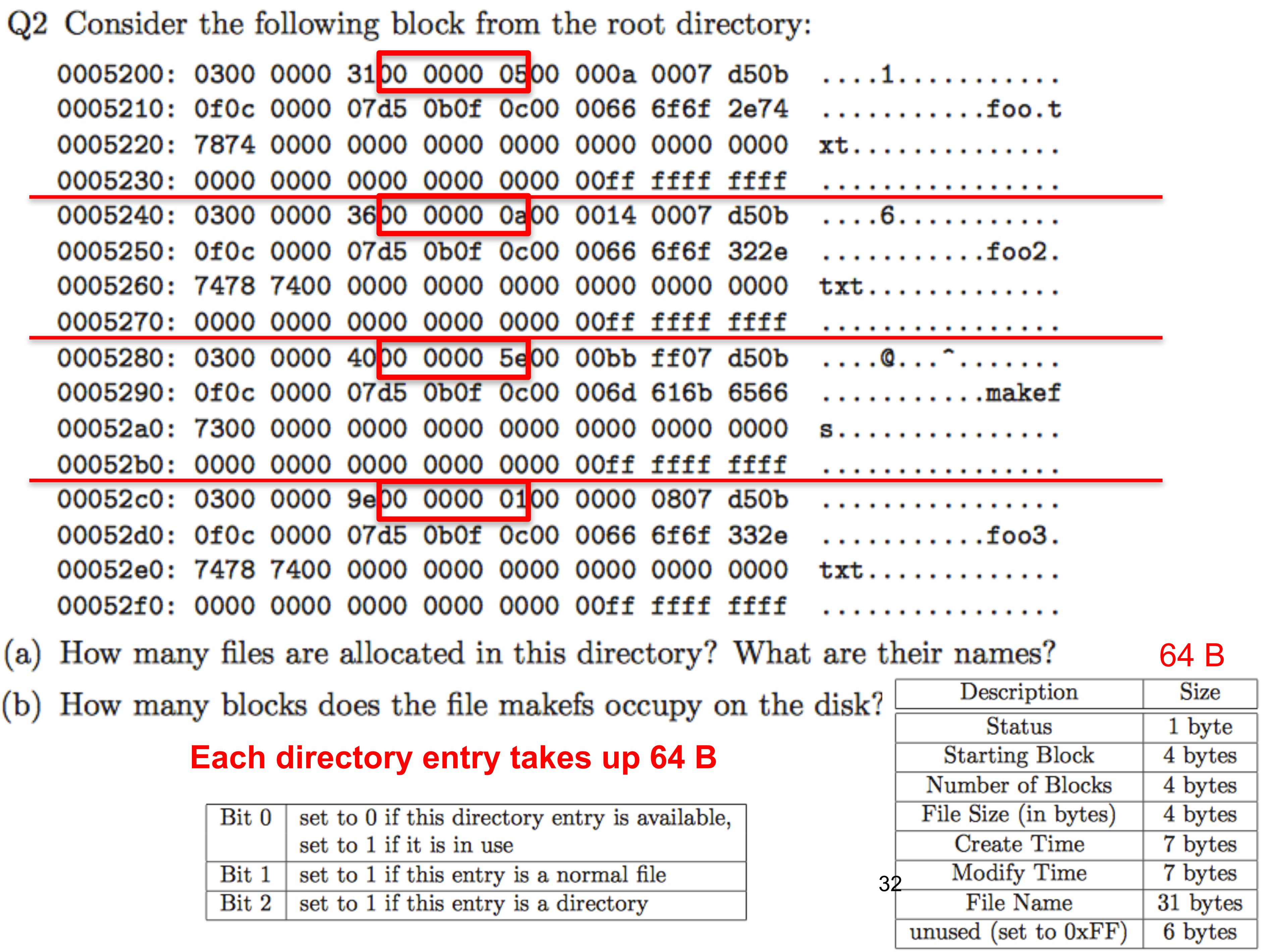
(b) How many blocks does the file makefs occupy on the disk?

Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

64 B

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes



Q2 Consider the following block from the root directory:

```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 00ff ffff ffff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

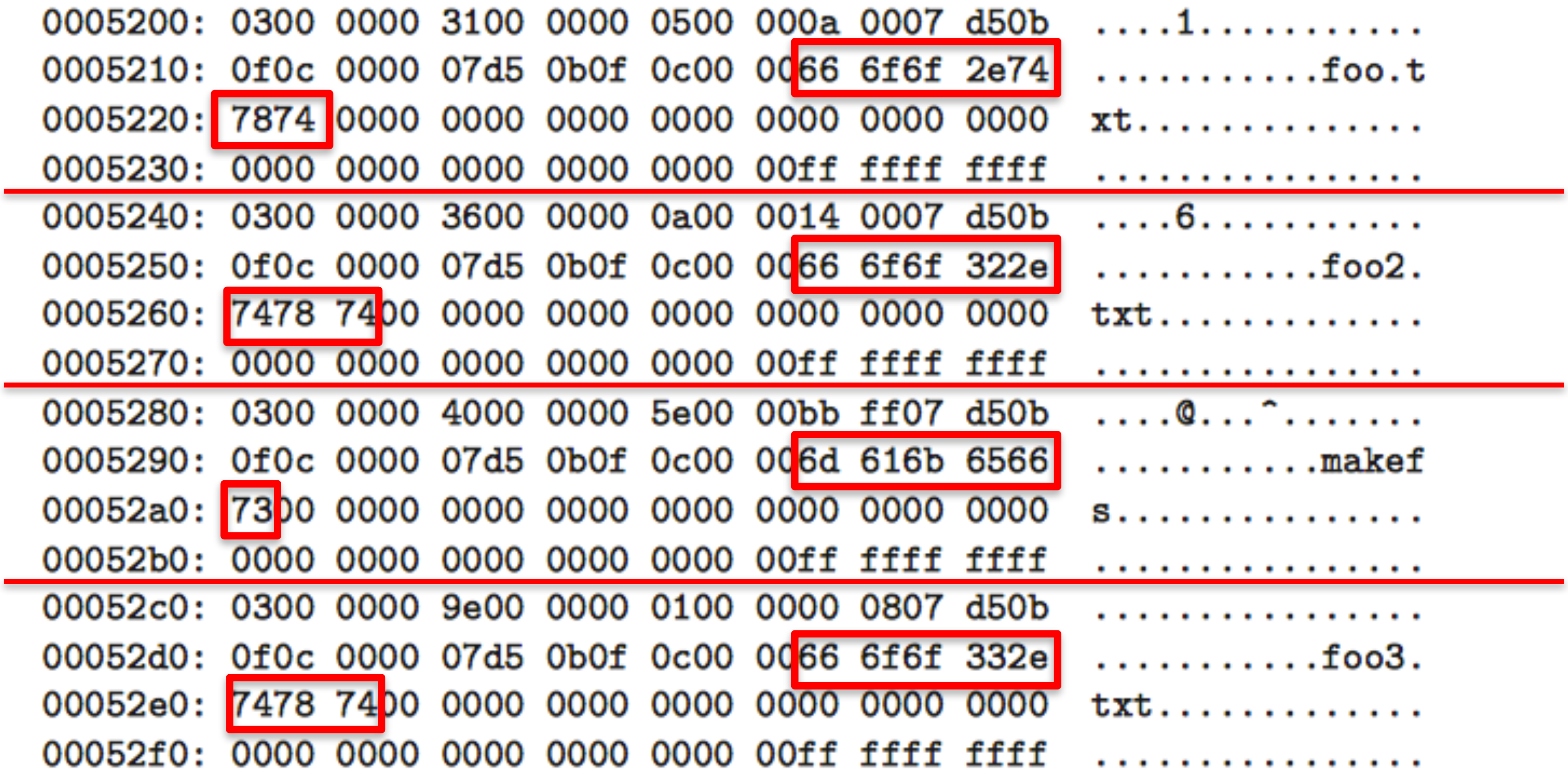
Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

64 B

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Q2 Consider the following block from the root directory:



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

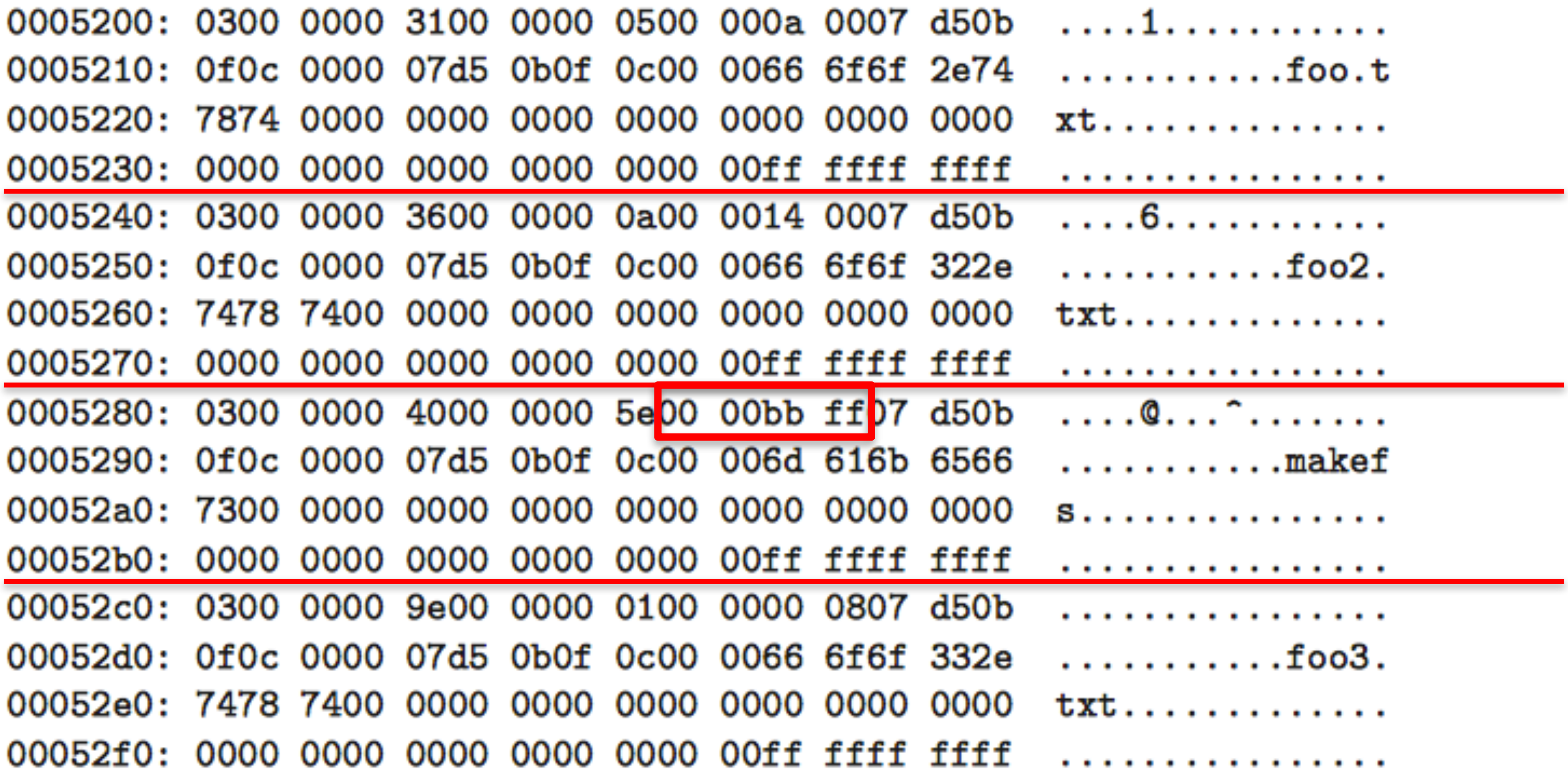
64 B

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

Q2 Consider the following block from the root directory:



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

(a) How many files are allocated in this directory? What are their names?

(b) How many blocks does the file makefs occupy on the disk?

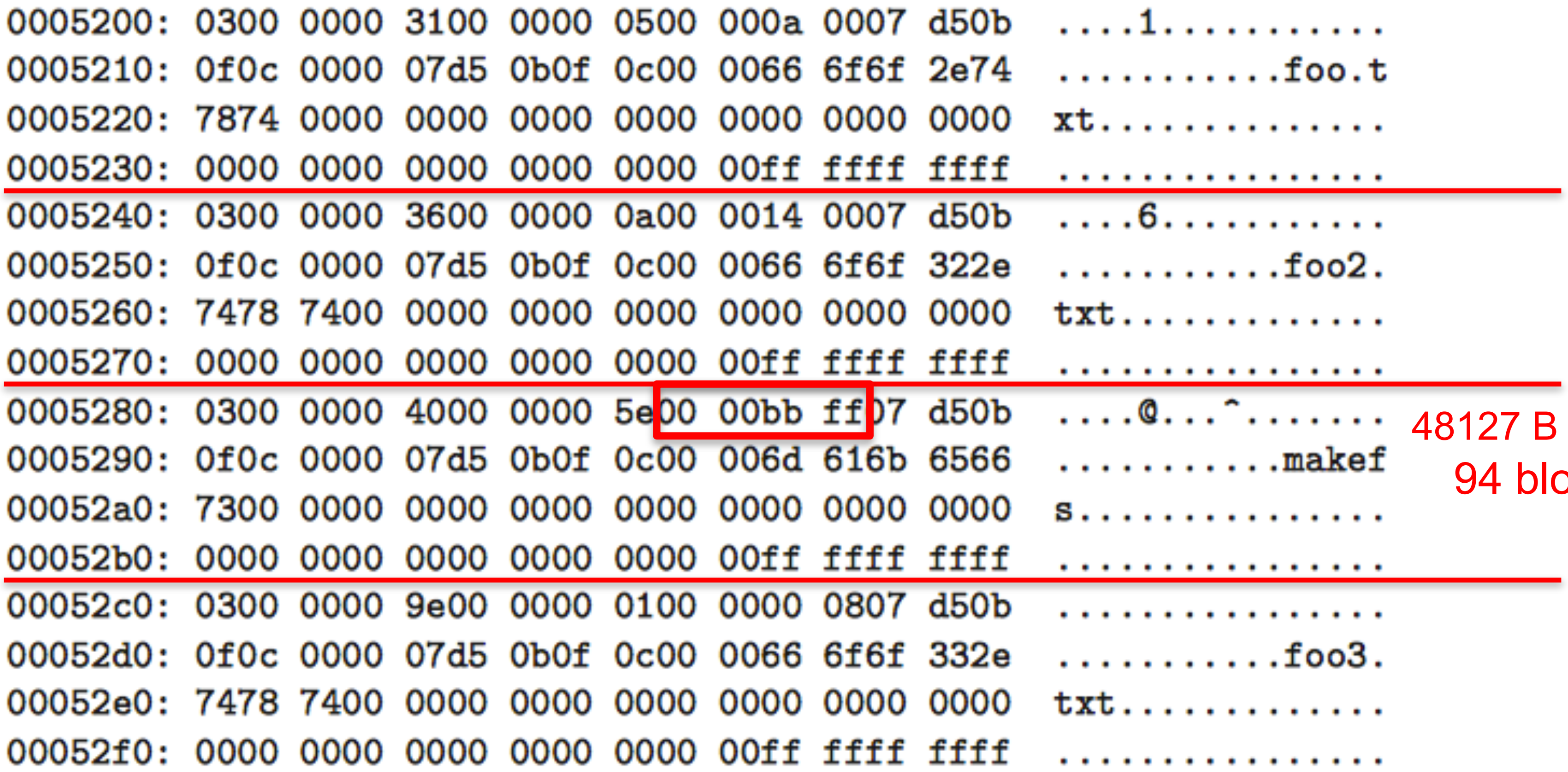
Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

64 B

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Q2 Consider the following block from the root directory:



```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

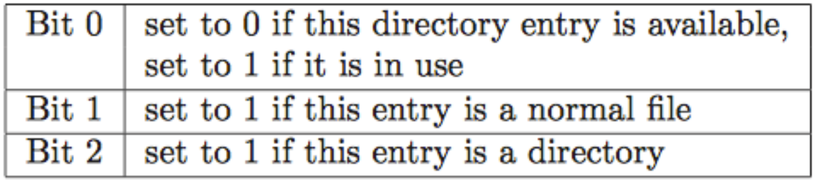
48127 B / 512 B
94 blocks

(a) How many files are allocated in this directory? What are their names?

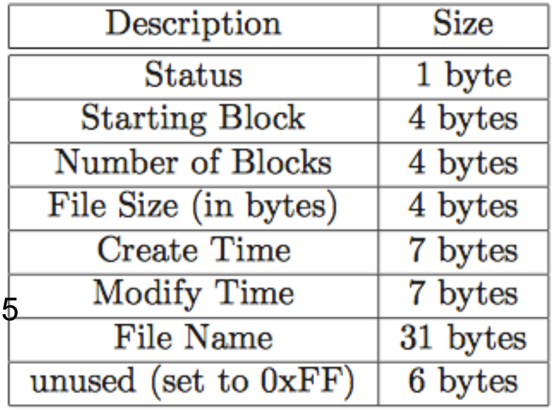
64 B

(b) How many blocks does the file makefs occupy on the disk?

Each directory entry takes up 64 B



Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory



Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes

Q2 Consider the following block from the root directory:

```
0005200: 0300 0000 3100 0000 0500 000a 0007 d50b  ....1...........
0005210: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 2e74  ...........foo.t
0005220: 7874 0000 0000 0000 0000 0000 0000 0000  xt..............
0005230: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005240: 0300 0000 3600 0000 0a00 0014 0007 d50b  ....6...........
0005250: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 322e  ...........foo2.
0005260: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0005270: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
0005280: 0300 0000 4000 0000 5e00 00bb ff07 d50b  ....@...^.......
0005290: 0f0c 0000 07d5 0b0f 0c00 006d 616b 6566  ...........makef
00052a0: 7300 0000 0000 0000 0000 0000 0000 0000  s...............
00052b0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
00052c0: 0300 0000 9e00 0000 0100 0000 0807 d50b  ................
00052d0: 0f0c 0000 07d5 0b0f 0c00 0066 6f6f 332e  ...........foo3.
00052e0: 7478 7400 0000 0000 0000 0000 0000 0000  txt.............
00052f0: 0000 0000 0000 0000 0000 00ff ffff ffff  ................
```

48127 B / 512 B

94 blocks

5e = 94

(a) How many files are allocated in this directory? What are their names?

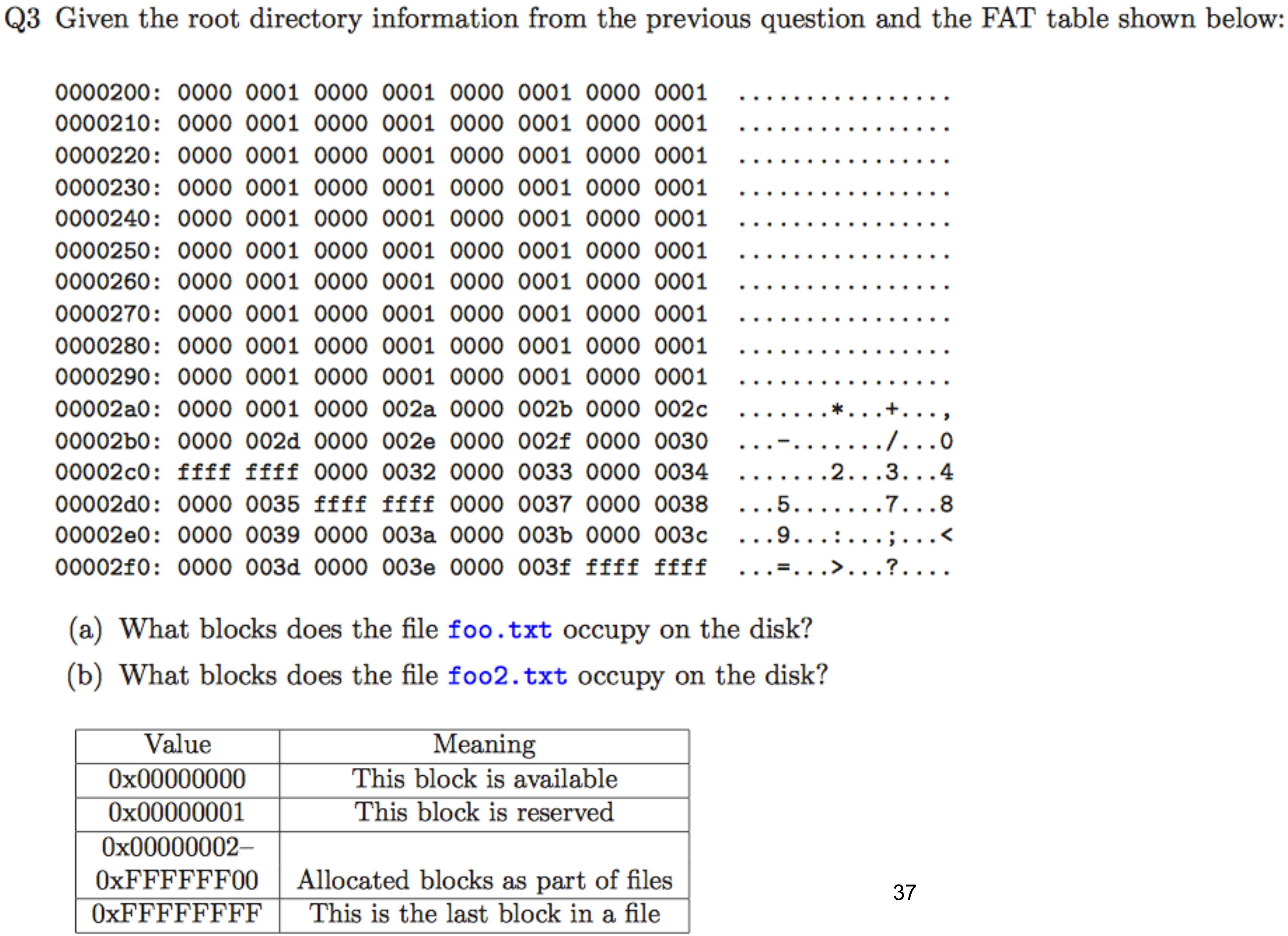
64 B

(b) How many blocks does the file makefs occupy on the disk?

Each directory entry takes up 64 B

Bit 0	set to 0 if this directory entry is available, set to 1 if it is in use
Bit 1	set to 1 if this entry is a normal file
Bit 2	set to 1 if this entry is a directory

Description	Size
Status	1 byte
Starting Block	4 bytes
Number of Blocks	4 bytes
File Size (in bytes)	4 bytes
Create Time	7 bytes
Modify Time	7 bytes
File Name	31 bytes
unused (set to 0xFF)	6 bytes



Q3 Given the root directory information from the previous question and the FAT table shown below:

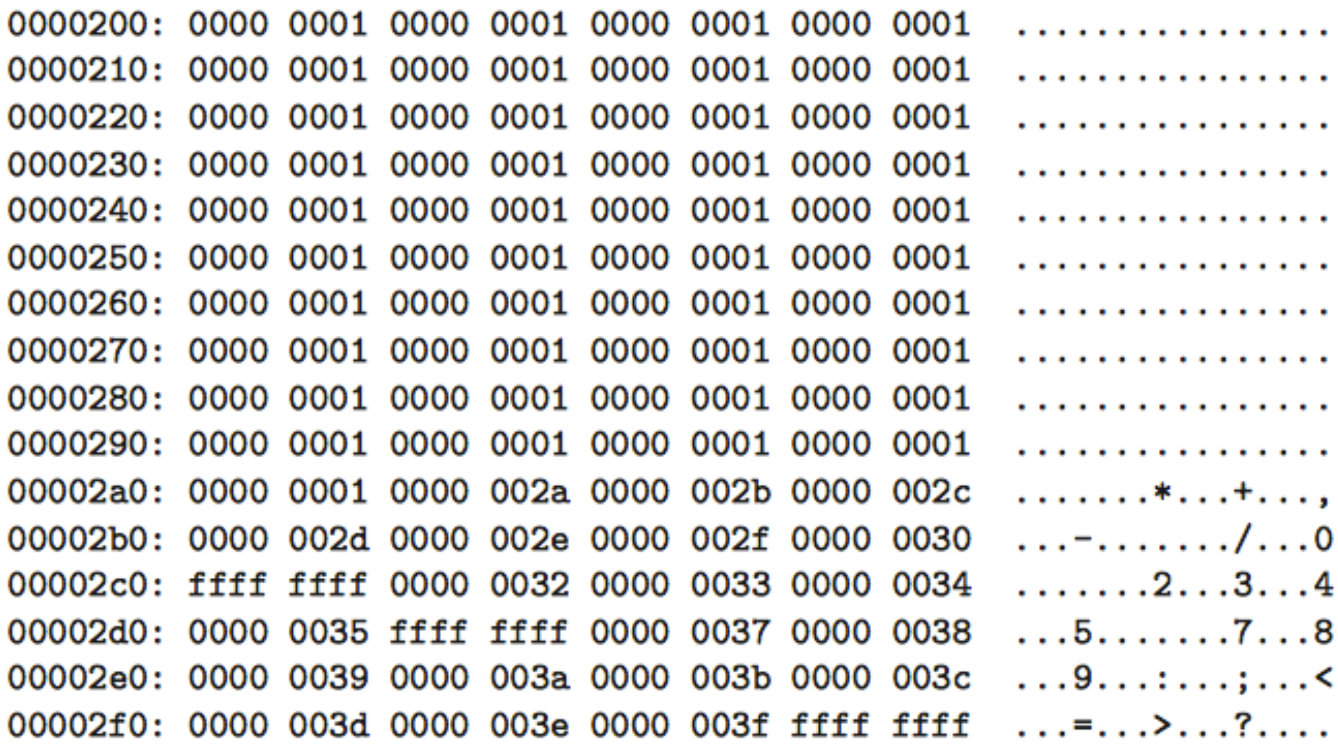
```
0000200: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000210: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000220: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000230: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000240: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000250: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000260: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000270: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000280: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000290: 0000 0001 0000 0001 0000 0001 0000 0001  ................
00002a0: 0000 0001 0000 002a 0000 002b 0000 002c  .......*...+...,
00002b0: 0000 002d 0000 002e 0000 002f 0000 0030  ...-......./...0
00002c0: ffff ffff 0000 0032 0000 0033 0000 0034  .......2...3...4
00002d0: 0000 0035 ffff ffff 0000 0037 0000 0038  ...5.......7...8
00002e0: 0000 0039 0000 003a 0000 003b 0000 003c  ...9...:...;...<
00002f0: 0000 003d 0000 003e 0000 003f ffff ffff  ...=...>...?....
```

(a) What blocks does the file `foo.txt` occupy on the disk?

(b) What blocks does the file `foo2.txt` occupy on the disk?

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002– 0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

Q3 Given the root directory information from the previous question and the FAT table shown below:



```
0000200: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000210: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000220: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000230: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000240: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000250: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000260: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000270: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000280: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000290: 0000 0001 0000 0001 0000 0001 0000 0001  ................
00002a0: 0000 0001 0000 002a 0000 002b 0000 002c  .......*...+...,
00002b0: 0000 002d 0000 002e 0000 002f 0000 0030  ...-......./...0
00002c0: ffff ffff 0000 0032 0000 0033 0000 0034  .......2...3...4
00002d0: 0000 0035 ffff ffff 0000 0037 0000 0038  ...5.......7...8
00002e0: 0000 0039 0000 003a 0000 003b 0000 003c  ...9...:...;...<
00002f0: 0000 003d 0000 003e 0000 003f ffff ffff  ...=...>...?....
```

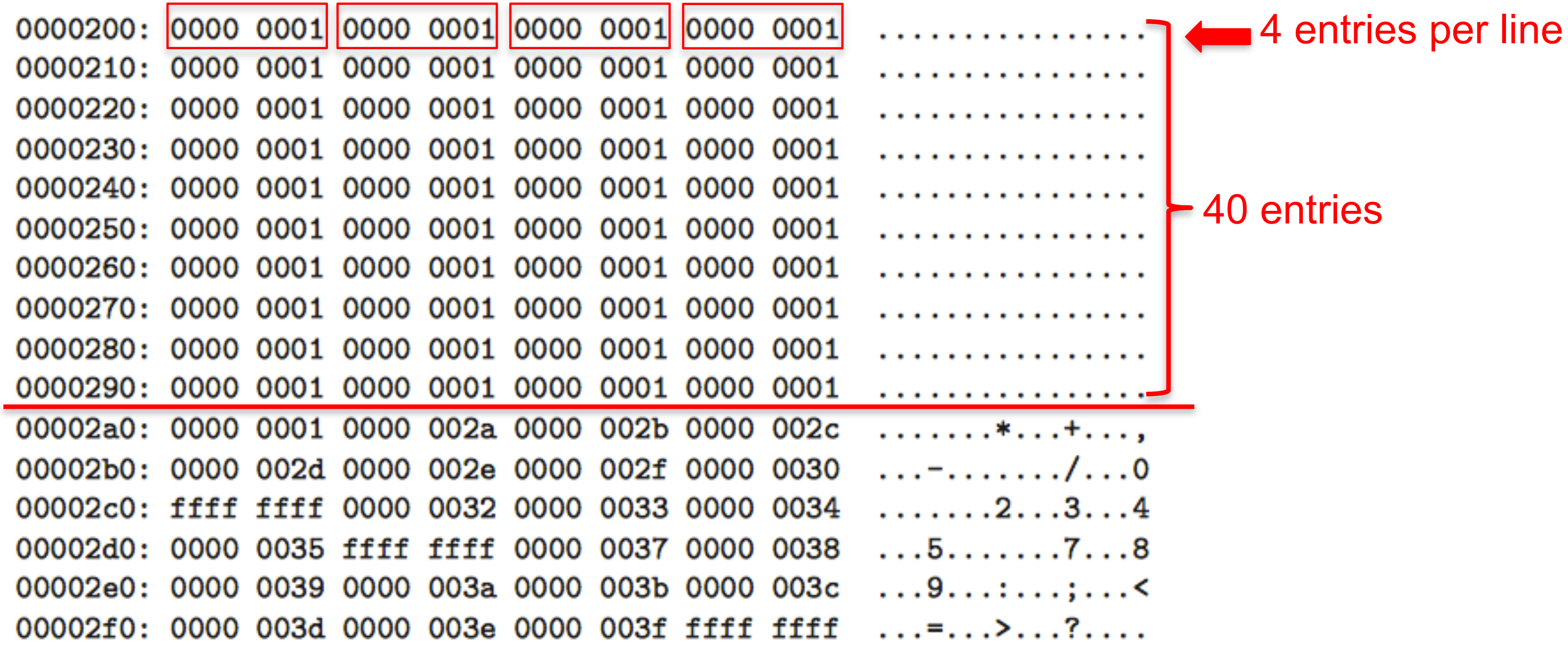
(a) What blocks does the file foo.txt occupy on the disk?   0x0000 0031 → entry 49

(b) What blocks does the file foo2.txt occupy on the disk?

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002– 0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

FAT entries are 4 B long (32 bits)

Q3 Given the root directory information from the previous question and the FAT table shown below:



```
0000200: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000210: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000220: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000230: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000240: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000250: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000260: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000270: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000280: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000290: 0000 0001 0000 0001 0000 0001 0000 0001  ................
00002a0: 0000 0001 0000 002a 0000 002b 0000 002c  .......*...+...,
00002b0: 0000 002d 0000 002e 0000 002f 0000 0030  ...-......./...0
00002c0: ffff ffff 0000 0032 0000 0033 0000 0034  .......2...3...4
00002d0: 0000 0035 ffff ffff 0000 0037 0000 0038  ...5.......7...8
00002e0: 0000 0039 0000 003a 0000 003b 0000 003c  ...9...:...;...<
00002f0: 0000 003d 0000 003e 0000 003f ffff ffff  ...=...>...?....
```

4 entries per line

40 entries

(a) What blocks does the file foo.txt occupy on the disk?

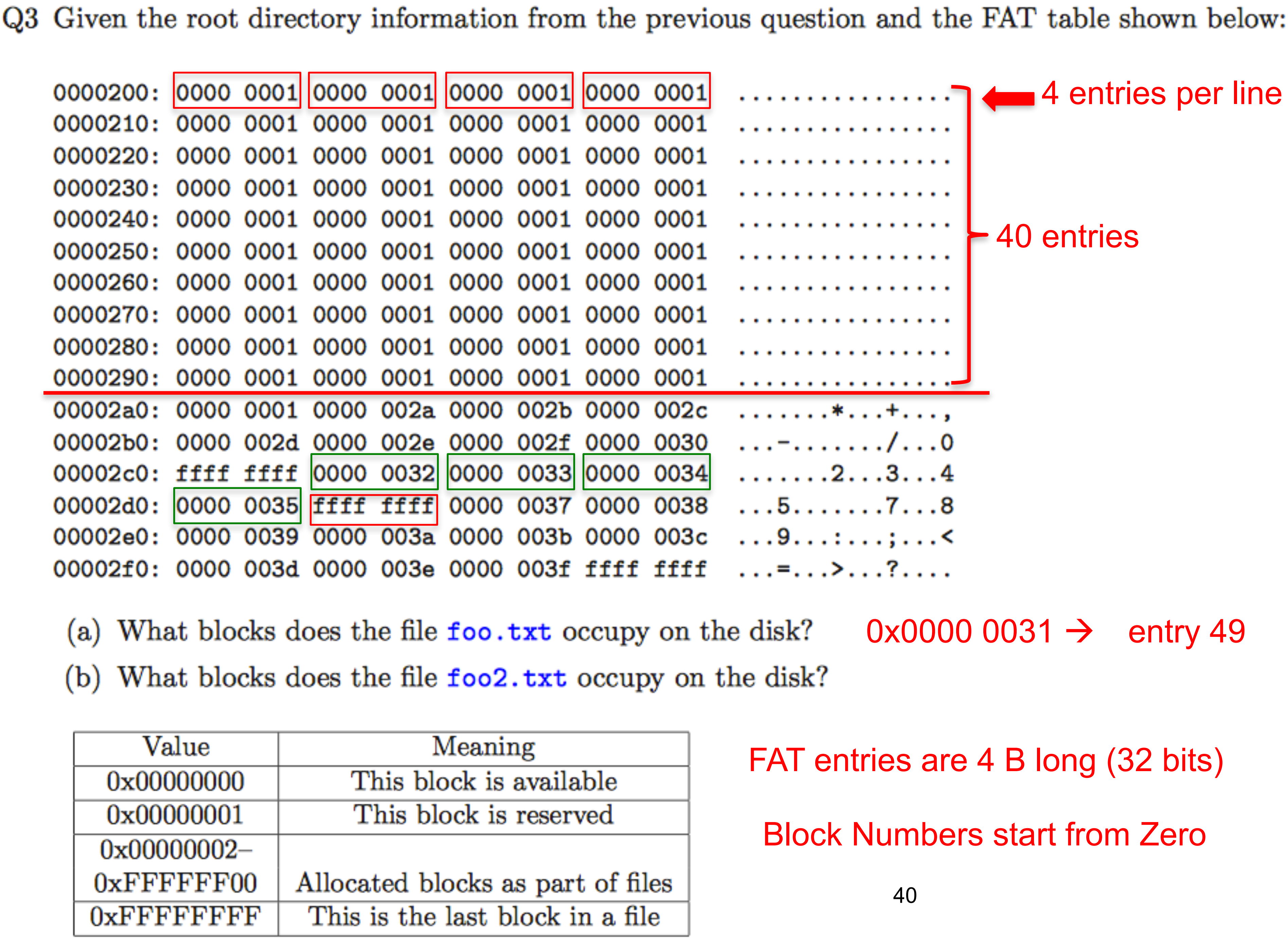
0x0000 0031 → entry 49

(b) What blocks does the file foo2.txt occupy on the disk?

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002– 0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

FAT entries are 4 B long (32 bits)

Block Numbers start from Zero



Q3 Given the root directory information from the previous question and the FAT table shown below:

```
0000200: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000210: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000220: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000230: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000240: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000250: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000260: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000270: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000280: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000290: 0000 0001 0000 0001 0000 0001 0000 0001  ................
00002a0: 0000 0001 0000 002a 0000 002b 0000 002c  .......*...+...,
00002b0: 0000 002d 0000 002e 0000 002f 0000 0030  ...-......./...0
00002c0: ffff ffff 0000 0032 0000 0033 0000 0034  ........2...3...4
00002d0: 0000 0035 ffff ffff 0000 0037 0000 0038  ...5.......7...8
00002e0: 0000 0039 0000 003a 0000 003b 0000 003c  ...9...:...;...<
00002f0: 0000 003d 0000 003e 0000 003f ffff ffff  ...=...>...?....
```

4 entries per line

40 entries

(a) What blocks does the file foo.txt occupy on the disk?

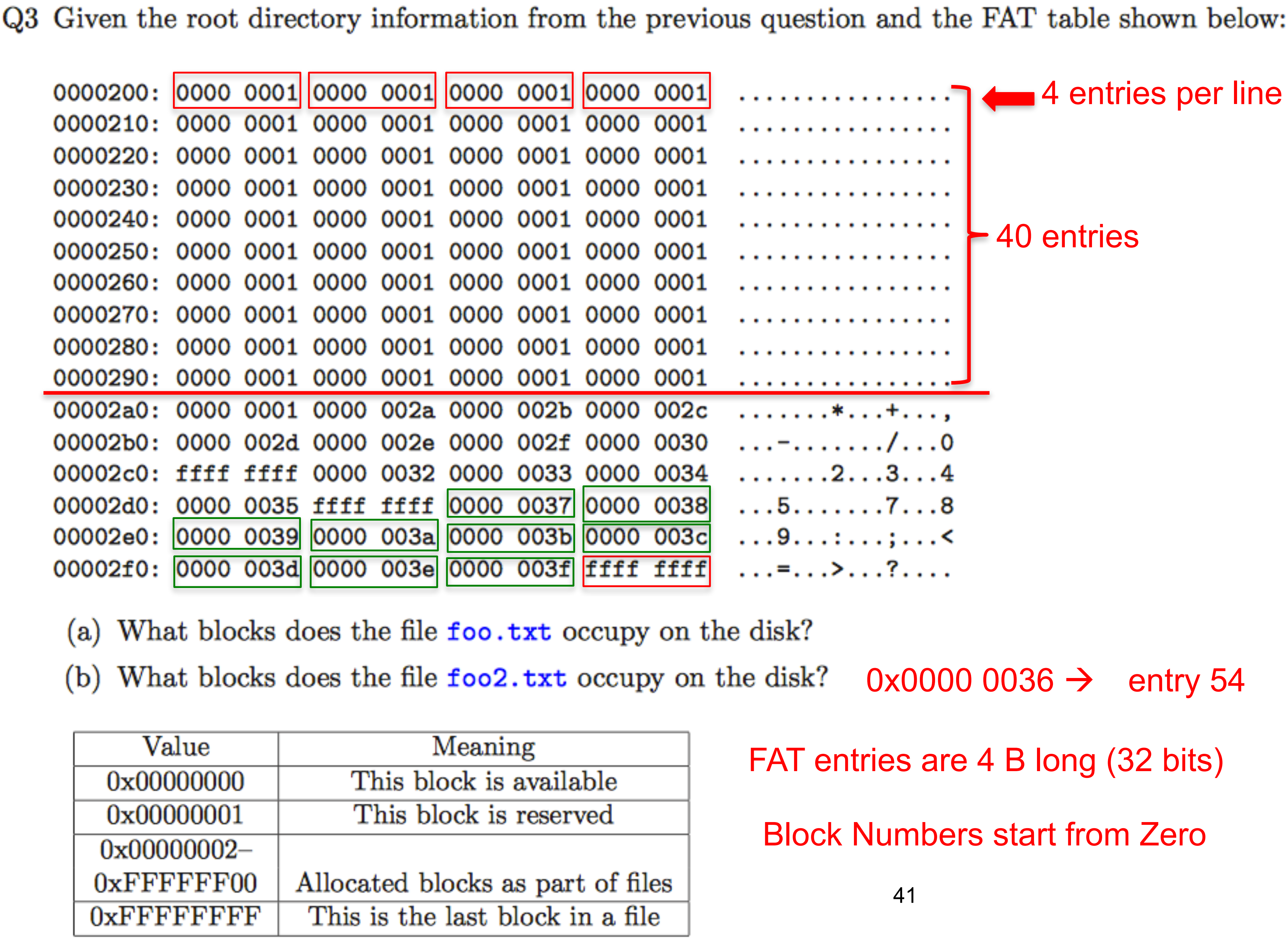
0x0000 0031 → entry 49

(b) What blocks does the file foo2.txt occupy on the disk?

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002– 0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

FAT entries are 4 B long (32 bits)

Block Numbers start from Zero



Q3 Given the root directory information from the previous question and the FAT table shown below:

```
0000200: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000210: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000220: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000230: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000240: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000250: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000260: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000270: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000280: 0000 0001 0000 0001 0000 0001 0000 0001  ................
0000290: 0000 0001 0000 0001 0000 0001 0000 0001  ................
00002a0: 0000 0001 0000 002a 0000 002b 0000 002c  .......*...+...,
00002b0: 0000 002d 0000 002e 0000 002f 0000 0030  ...-......./...0
00002c0: ffff ffff 0000 0032 0000 0033 0000 0034  .......2...3...4
00002d0: 0000 0035 ffff ffff 0000 0037 0000 0038  ...5.......7...8
00002e0: 0000 0039 0000 003a 0000 003b 0000 003c  ...9...:...;...<
00002f0: 0000 003d 0000 003e 0000 003f ffff ffff  ...=...>...?....
```

4 entries per line

40 entries

(a) What blocks does the file foo.txt occupy on the disk?

(b) What blocks does the file foo2.txt occupy on the disk?

0x0000 0036 → entry 54

Value	Meaning
0x00000000	This block is available
0x00000001	This block is reserved
0x00000002–0xFFFFFF00	Allocated blocks as part of files
0xFFFFFFFF	This is the last block in a file

FAT entries are 4 B long (32 bits)

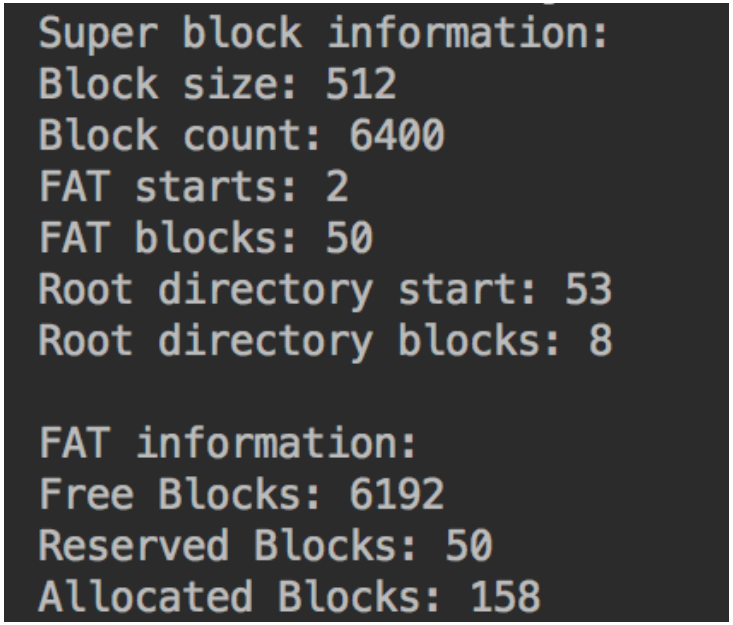
Block Numbers start from Zero

Conclusion

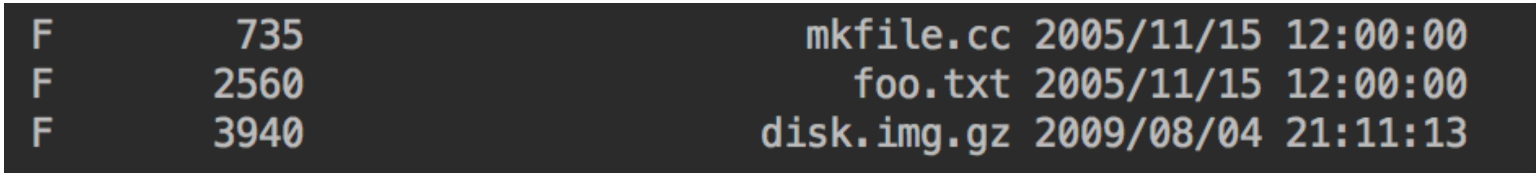
FAT only knows what the next block is.
Directory helps finding the starting block.
Root is the starting of all the directories and files.

There exercise questions are related:
In Q1, we can see the FAT starts from 0x01 and has 0x28 blocks.
In Q2, address starts from block 0x29.
Corresponds to Q3.

info on the test.img



```
Super block information:
Block size: 512
Block count: 6400
FAT starts: 2
FAT blocks: 50
Root directory start: 53
Root directory blocks: 8

FAT information:
Free Blocks: 6192
Reserved Blocks: 50
Allocated Blocks: 158
```



```
F        735                   mkfile.cc 2005/11/15 12:00:00
F       2560                     foo.txt 2005/11/15 12:00:00
F       3940                 disk.img.gz 2009/08/04 21:11:13
```