The Acorn DFS Osword commands - by - Gordon Horsington

Module 0. Introduction

```
      +-----------------------------------------------------------+
      | All the DFS modules in  this  series  use programs which |
      | experiment  with the format and contents of discs. These |
      | experiments may  have disasterous effects if you use any |
      | of the programs  on  discs  which store programs or data |
      | which you cannot afford to lose.  You  should  first try |
      | out  the  programs  using  discs  that  have either been |
      | duplicated or, better still, have not been used at all.  |
      +-----------------------------------------------------------+
```

There are eight modules in this part of the Opcodes series. The modules are in files named T/DFS00 to T/DFS07. The following topics will be covered in these series modules.

Module 0. This module. Introduction.
Programs: IDSDUMP, VERIFY

Module 1. The DFS Osword commands (Part 1).
Programs: CYCLES, HOWMANY, STATUS, OUTPUT

Module 2. The DFS Osword commands (Part 2).
Programs: FORM10, NOTFORM, WRITE10, READ10

Module 3. Formatting single density discs.
Program: OFFSET, IDSDUMP, VERIFY

Module 4. Converting 40 track discs to run on 80 track disc drives.
Program: CONVERT

Module 5. Creating discs compatible with both 40 and 80 track drives.
Program: DUALDFS, OFFSET

Module 6. Creating copy-protected single density discs.
Programs: SECTOR5, ENCODE, DECODE, IDSDUMP, VERIFY

Module 7. Duplicating copy-protected single density discs.
Programs: COPYDFS, COPYALL, DEFORM, IDSDUMP, VERIFY

The later modules develop the ideas and techniques described in the earlier modules and for this reason the series needs to be worked through from beginning to end rather than used as a reference guide.

Introduction to single density discs

There are three levels at which data can be written to and read from single density discs. The highest level, with which all disc users should be familiar, is through using the DFS star commands, filenames and the range of BGET, BPUT and other filing system commands. This level has a large ammount of filing system independence so that, for example, the same *SAVE command syntax can be used with both tape and disc. When this high level access to the DFS is used, both the programmer and the program user are tied to the restrictions of the particular DFS ROM so that, for example, non-standard disc formats are not available.

A lower level of access to the DFS ROM is provided by the DFS Osword commands. Oswords &7D to &7F are used by the DFS and Oswords &70 to &73 are used by the ADFS.

The ADFS is only available on BBC computers which use the Western Digital 1770 (or 1772) disc controller. The BBC B DFS was designed to use the Intel 8271 disc controller and cannot support the ADFS without a 1770 upgrade. The 1770 disc controller is fitted as standard to the BBC B+ and Master series computers. The 1770 disc controller can support both single density and double density disc formats but the 8271 can only support the single density format. All disc based BBC computers are capable of using

Oswords &7D to &7F but only those with the ADFS can use Oswords &70 to &73. Only Oswords &7D to &7F will be considered in detail in this series.

The lowest level of access to discs can be achieved by programing the disc contoller directly.

The hardware which makes up a BBC microcomputer system is memory mapped. This means that the usable registers of all the hardware devices available to the I/O processor are mapped onto the main memory address space used by the 6502 CPU. Page &FE, ie. memory from &FE00 to &FEFF, is known as Sheila and this page is reserved for the hardware on the I/O processor's circuit board.

Sheila addresses &80 to &9F are available to the floppy disc controller. Five of the BBC B's 8271 registers are mapped onto the Sheila addresses from &FE80 to &FE82. Three of the five registers can only be written into and the other two can only be read from. For this reason only three Sheila addresses need to be used to access the five registers. These three addresses can be used to communicate with the 8271 and to instruct it to execute a wide range of functions. Sheila address &FE84 is used to pass data to, and to read data from, the disc controller. The mapping of the 8271 registers onto Sheila addresses is shown in figure 1.

8271 register	Sheila address	read or write
status	&FE80	read
result	&FE81	read
command	&FE80	write
parameter	&FE81	write
reset	&FE82	write

Figure 1. The 8271 registers mapped onto Sheila addresses

The 8271 has twelve other registers, known as the Special Registers, which are not mapped onto the Sheila addresses. Access to these registers can be achieved indirectly using the 8271 Read Special Register command or the 8271 Write Special Register command, both of which can be sent to the disc controller using the registers in figure 1. The Sheila addresses can be peeked or poked using the indirection operator but to produce Tube-compatible code it is necessary to use Osbyte &96 to read the Sheila addresses and Osbyte &97 to write to them.

Using Osbytes &96 and &97 will ensure that the code is Tube-compatible but it not the easiest or the best way to program the disc controller. Osword &7F executes a single 8271 command through all its phases and relieves the programmer of the problems associated with techniques such as non-maskable interupt handling which must be used when programming the 8271. Osword &7F uses the Sheila addresses from &FE80 to &FE84 but the programmer does not have to be concerned with, or even be aware of, the detailed use of these addresses. Osword &7F provides a standard interface on the disc controller and is the method of accessing the disc hardware used in this series.

Before looking at the single density Oswords in detail it is necessary to understand the format used by single density discs.

The BBC computer is capable of using 3 1/2 inch, 5 1/4 inch and 8 inch discs, although 8 inch discs are something of a rarity these days. Because 8 inch discs are so uncommon they will not be discussed in this series. Both 3 1/2 inch and 5 1/4 inch discs use the same format and are available in both single and double sided versions with either 40 or 80 tracks per side.

Each track is subdivided into a number of sectors each of which has an identification field (usually called an ID field) and a data field.

There are a number of gaps associated with each track. The gaps are a variable number of bytes between the ID and data fields and are used to space out the fields to prevent the sectors overwriting each other when the disc speed varies.

Physical tracks and sectors are identified by their physical position on a disc. Physical track 0 is the outermost track and physical sector 0 is the first sector on a track after the index pulse hole. Every sector is given a one-byte logical track number and a one-byte logical sector number. The physical track numbers and logical track numbers are the same on discs formatted for the Acorn DFS but the physical and logical sector numbers do not have to be the same. One of the many ways of copy-protecting discs is to make the physical and logical track numbers different, this effectivly disables the DFS *BACKUP command.

The number of sectors, the size of the data fields, and the gap sizes are determined when the disc is formatted. In module 3 you will have the opportunity to vary these parameters but, whatever the format, each single density track has the layout shown in figure 2 below.

```
  +------------+----------+------+----------+
  | Sync bytes | Data mark | data | data CRC |
  +------------+----------+------+----------+
              \             /
Index          \  Data     /                                  End of
mark            \ field   /                                    track
+-----+----+-----+------+-----+----+-----+------+-----+     +-----+-----+
| Gap | ID | Gap | Data | Gap | ID | Gap | Data | Gap | ... | Gap | Gap |
|  1  |    |  2  |  0   |  3  |    |  2  |  1   |  3  |     |  4  |  5  |
+-----+----+-----+------+-----+----+-----+------+-----+     +-----+-----+
     / ID   \
    / field  \
   /          \
  +-------+------+---------+--------+---------+-----------+--------+
  | Sync  | ID   | Logical | Head   | Logical | Data size | Sector |
  | bytes | mark | track   | number | sector  | code      | ID CRC |
  +-------+------+---------+--------+---------+-----------+--------+
```

Figure 2. The layout of a DFS track.

Figure 2 illustrates a complete track with one of the data fields expanded above the centre line and one of the ID fields expanded below. If figure 2 is taken as an illustration of a 10 sector track, then sectors 2 - 9 have been left out.

The index mark at the beginning of the track has its position determined by the physical index pulse hole in the disc. This mark is followed by Gap 1. Gap 1 occurs once per track between the index mark and the start of physical sector 0. It should always be 16 (&10) bytes long.

Gap 1 is followed by the ID field for physical sector 0. The ID field starts with 6 sync bytes. These sync bytes synchronise the controller to the rotational speed of the disc. The sync bytes are followed by a sector ID mark, which simply marks the start of the sector. This in turn is followed by the logical track number. The logical track number is normally the same as the physical track number but, on non-standard discs, it does not have to be the same. If you use the demonstration progran IDSDUMP on a copy-protected disc you may find that the physical track numbers and the logical track numbers of the protected tracks are different.

The logical track number is followed by the head number. This should be &00 for drives 0 and 1 (which use the under side of the disc), and &01 for drives 2 and 3 (which use the top side of the disc) but most formatting programs use a head number of &00 for all disc surfaces. A head number of &00 seems to be ignored by the disc controller.

Next comes the logical sector number. The logical sector number does not have to be the same as the physical sector number and, as will be demonstrated in module 3, there can be a small advantage to be gained by not using the same logical and physical sector numbers.

The data size code follows the logical sector number and it indicates the number of data bytes in the data field (see figure 3 below). This is followed by a two byte cyclic redundancy check (CRC). The CRC is used to check for errors in the data stored within the ID field.

The ID field is followed by Gap 2. Gap 2 should always be 11 (&0B) bytes long and it is positioned between the ID field and the data field of each sector on the track. Each data field is followed by Gap3. The Gap 3 after the last data field is followed by Gap 4 which in turn is followed by Gap

5. The relationship between the sector sizes and the gap sizes for 3 1/2 inch and 5 1/4 inch discs is shown in figure 3. All the numbers in figure 3 are in decimal.

No. Sectors	Size code	Length	Gap1	Gap2	Gap3	Gap4	Gap5
18	0	128	16	11	11	24	0
10	1	256	16	11	21	30	0
5	2	512	16	11	74	88	0
2	3	1024	16	11	255	740	0
1	4	2048	16	11	0	1028	0

Figure 3. The relationship between sector size code, length and gap size.

The data field for each sector starts with 6 sync bytes which are used to synchronise the disc controller with the rotational speed of the disc. The sync bytes are followed by the data mark which identifies the start of the data and also indicates if the data are marked as "deleted". Deleted data are not physically deleted from the disc, they are simply marked as deleted. This type of data marking will be used in module 6 to help produce copy-protected discs. The data follow the data mark and are terminated with two data CRC bytes.

The program IDSDUMP can be used to print the ID field for every sector on a single density disc. I will not explain how the program works because it uses techniques that will be covered in the next two modules. The program is commented so that you can come back to it after reading modules 1 and 2 when you should be able to understand how it works.

```
   10 REM: IDSDUMP
   20 zeropage=&70
   30 osasci=&FFE3
   40 osnewl=&FFE7
   50 osword=&FFF1
   60 osbyte=&FFF4
   70 DIM buffer &50
   80 DIM mcode &200
   90 FOR pass=0 TO 2 STEP 2
  100 P%=mcode
  110 [       OPT pass
  120         LDA #14       \ paged mode
  130         JSR osasci
  140 .mainloop
  150         JSR escape    \ check escape flag
  160         JSR firstsector \ read sector id first sector
  170         BNE notformatted \ if error, track not formatted
  180         JSR tracknumber \ print track number
  190         JSR sectorids \ read all sector ids
  200 .notformatted
  210         INC physical  \ increment physical track number
  220         LDA physical  \ load physical track number
  230         CMP last      \ all done?
  240         BNE mainloop  \ if not copy next track
  250         JSR osnewl
  260         RTS           \ return to BASIC
  270 .escape
  280         LDA &FF       \ escape flag
  290         BMI pressed   \ bit 7 set if pressed
  300         RTS
  310 .pressed
  320         LDA #&7E
  330         JSR osbyte    \ acknowledge Escape
  340         BRK
  350         BRK
  360         EQUS "Escape"
  370         BRK
  380 .firstsector
  390         LDA physical  \ physical track number
  400         STA idsblock+7 \ store physical track
  410         LDA #1        \ one sector
  420         STA idsblock+9 \ number of ids
```

```
 430         LDA #&7F
 440         LDX #idsblock MOD 256
 450         LDY #idsblock DIV 256
 460         JSR osword
 470         LDA idsblock+10 \ result
 480         AND #&1E      \ = 0 if formatted
 490         RTS
 500 .sectorids
 510         LDX buffer+3   \ load data size code
 520         LDA sizes,X   \ load number of sectors
 530         STA idsblock+9 \ store number of sectors
 540         ASL A         \ *2
 550         ASL A         \ *4
 560         STA sectornumber \ store index on sectors
 570         LDA #&7F
 580         LDX #idsblock MOD 256
 590         LDY #idsblock DIV 256
 600         JSR osword
 610         LDA idsblock+10 \ result
 620         AND #&1E
 630         BNE idserror  \ = 0 if OK
 640         LDX #0
 650 .next
 660         LDY #0
 670 .printloop
 680         LDA buffer,X
 690         JSR printbyte \ print every byte of sector table
 700         INX
 710         INY
 720         CPY #4        \ 4 bytes per line
 730         BNE printloop
 740         JSR osnewl
 750         CPX sectornumber \ last byte?
 760         BCC next      \ go back for more
 770         RTS
 780 .idserror
 790         BRK
 800         BRK
 810         EQUS "Sector ID Error"
 820         BRK
 830 .tracknumber
 840         JSR osnewl
 850         LDX #title MOD 256
 860         LDY #title DIV 256
 870         JSR printtext \ print "Track &"
 880         LDA physical  \ load physical track number
 890         JSR printbyte \ print track number
 900         LDX #header MOD 256
 910         LDY #header DIV 256
 920         JMP printtext \ print "LT HN LS DS"
 930 .printtext
 940         STX zeropage
 950         STY zeropage+1
 960         LDY #0
 970 .textloop
 980         LDA (zeropage),Y
 990         BEQ endtext
1000         JSR osasci
1010         INY
1020         BNE textloop
1030 .endtext
1040         RTS
1050 .printbyte
1060         PHA
1070         LSR A
1080         LSR A
1090         LSR A
1100         LSR A
1110         JSR nybble    \ print MS nybble
1120         PLA
1130         JSR nybble    \ print LS nybble
1140         LDA #ASC(" ")
1150         JSR osasci    \ print space
1160         JMP osasci    \ print space
1170 .nybble
1180         AND #&0F
1190         SED
```

```
 1200          CLC
 1210          ADC #&90
 1220          ADC #&40
 1230          CLD
 1240          JMP osasci    \ print nybble and return
 1250 .idsblock
 1260          EQUB &FF      \ current drive
 1270          EQUD buffer   \ address of buffer
 1280          EQUD &00005B03 \ read sector ids
 1290          EQUW 0
 1300 .sizes
 1310          EQUB 18
 1320          EQUB 10
 1330          EQUB 5
 1340          EQUB 2
 1350          EQUB 1
 1360 .title
 1370          EQUS "  Track  &"
 1380          BRK
 1390 .header
 1400          EQUB &0D
 1410          EQUS "  ----------"
 1420          EQUB &0D
 1430          EQUS "LT  HN  LS  DS"
 1440          EQUB &0D
 1450          EQUS "--------------"
 1460          EQUB &0D
 1470          BRK
 1480 .physical
 1490          EQUB &00
 1500 .sectornumber
 1510          EQUB &00
 1520 .last
 1530          EQUB &00
 1540 ]
 1550 NEXT
 1560 INPUT'"Number of tracks (40/80) "tracks$
 1570 IF tracks$="40" ?last = 40 ELSE ?last = 80
 1580 PRINT'"Insert ";?last;" track disc into current drive"
 1590 PRINT"and press Spacebar to print sector IDs"
 1600 REPEAT
 1610 UNTIL GET=32
 1620 PRINT'"Press Shift to scroll"
 1630 CALL mcode
```

Chain the program IDSDUMP and, when prompted, put a suitable disc in the current drive. Then press the spacebar to print the sector IDs for every track on the disc. The track number is displayed and the logical track (LT), head number (HN), logical sector (LS) and data size code (DS) are printed for every physical sector on each track, starting with physical sector 0.

It is quite interesting to use this program with a copy-protected disc. You will almost certainly find that some of the physical and logical track numbers are different and you may also find some unexpected logical sector numbers. Figure 4 is a part of the output I produced with the 40 track single density disc version of the game "Grand Prix Construction Set".

```
  Track  &0A
  ----------
LT  HN  LS  DS
--------------
14  00  00  01
14  00  01  01
14  00  02  01
14  00  03  01
14  00  04  01
14  00  05  01
14  00  06  01
14  00  07  01
14  00  08  01
14  00  09  01
```

Figure 4. Part of the output from the program IDSDUMP

You should notice that physical track &0A uses logical track number &14 but the physical sector numbers, indicated by the order of the sectors, are the same as the logical sector numbers. This use of different physical and logical track numbers is sufficient to prevent the *BACKUP command duplicating the disc. As if to make sure it can't be copied, the disc also uses deleted data to re-inforce the same effect. Deleted data markers cannot be displayed with the program IDSDUMP and so I have provided a program called VERIFY which will verify copy-protected discs and indicate which tracks use deleted data.

The deleted data mark is a part of the data field and can be read using the verify command. This will also be explained in detail in a later module but, for now, you should find it interesting to use the program VERIFY to find the deleted data on a copy-protected disc.

```
   10 REM: VERIFY
   20 REM: for copy-protected discs
   30 osnewl=&FFE7
   40 oswrch=&FFEE
   50 osword=&FFF1
   60 osbyte=&FFF4
   70 DIM buffer &50
   80 DIM mcode &500
   90 FOR pass=0 TO 2 STEP 2
  100 P%=mcode
  110 [       OPT pass
  120         JSR osnewl
  130 .mainloop
  140         JSR escape    \ check escape flag
  150         JSR seek      \ seek physical tracks 0 - 40
  160         JSR firstsector \ read sector id first sector
  170         BNE notverify \ if error track not formatted
  180         JSR sectorids \ read all sector ids
  190         JSR verify    \ verify all sectors
  200 .notverify
  210         JSR printbyte \ print track number
  220         INC physical  \ increment physical track number
  230         LDA physical  \ load physical track number
  240         CMP last      \ all done?
  250         BNE mainloop  \ if not copy next track
  260         JSR osnewl
  270         RTS           \ return to BASIC
  280 .escape
  290         LDA &FF       \ escape flag
  300         BMI pressed   \ bit 7 set if pressed
  310         RTS
  320 .pressed
  330         LDA #&7E
  340         JSR osbyte    \ acknowledge Escape
  350         BRK
  360         BRK
  370         EQUS "Escape"
  380         BRK
  390 .seek
  400         LDA physical  \ physical track number
  410         STA seekblock+7
  420         LDA #&7F
  430         LDX #seekblock MOD 256
  440         LDY #seekblock DIV 256
  450         JSR osword
  460         LDA seekblock+8 \ result
  470         BNE seekerror \ = 0 if OK
  480         RTS
  490 .seekerror
  500         BRK
  510         BRK
  520         EQUS "Seek error"
  530         BRK
  540 .firstsector
  550         LDA physical  \ physical track number
  560         STA idsblock+7 \ store physical track
  570         LDA #1        \ one sector
  580         STA idsblock+9 \ number of ids
```

```
 590         LDA #&7F
 600         LDX #idsblock MOD 256
 610         LDY #idsblock DIV 256
 620         JSR osword
 630         LDA idsblock+10 \ = 0 if formatted
 640         RTS
 650 .sectorids
 660         LDX buffer+3  \ load data size code
 670         LDA sizes,X   \ load number of sectors
 680         STA idsblock+9 \ store number of sectors
 690         ASL A         \ *2
 700         ASL A         \ *4
 710         SEC
 720         SBC #4        \ sectors*4-4
 730         STA sectornumber \ store index on sectors
 740         TXA           \ transfer data size code
 750         ASL A         \ *2
 760         ASL A         \ *4
 770         ASL A         \ *8
 780         ASL A         \ *16
 790         ASL A         \ *32
 800         ORA idsblock+9 \ add number of sectors
 810         STA verblock+9 \ store for verify
 820         LDA #&7F
 830         LDX #idsblock MOD 256
 840         LDY #idsblock DIV 256
 850         JSR osword
 860         LDA idsblock+10 \ result
 870         BNE idserror  \ = 0 if OK
 880         RTS
 890 .idserror
 900         BRK
 910         BRK
 920         EQUS "Sector ID Error"
 930         BRK
 940 .verify
 950         LDX sectornumber \ load index on table
 960         LDA buffer+2,X \ load logical sector number
 970         STA verblock+8 \ store for verify
 980 .lowest
 990         DEX
1000         DEX
1010         DEX
1020         DEX
1030         BMI finished
1040         LDA buffer+2,X \ load logical sector number
1050         CMP verblock+8 \ is it lower than the last one?
1060         BCS lowest    \ branch if not lowest sector
1070         STA verblock+8 \ store if it is lower
1080         BCC lowest    \ look for lower sector number
1090 .finished
1100         LDA buffer    \ load logical track number
1110         STA verblock+7 \ and store for verify
1120         JSR register  \ write track register
1130         LDA #&7F
1140         LDX #verblock MOD 256
1150         LDY #verblock DIV 256
1160         JSR osword
1170         LDA physical  \ physical track number
1180         JSR register  \ write track register
1190         LDA verblock+10
1200         AND #&1E      \ isolate error bits
1210         BNE vererror
1220         RTS
1230 .vererror
1240         BRK
1250         BRK
1260         EQUS "Verify error"
1270         BRK
1280 .register
1290         STA regblock+8 \ value to put in register
1300         LDA #&7F
1310         LDX #regblock MOD 256
1320         LDY #regblock DIV 256
1330         JSR osword
1340         LDA regblock+9
1350         BNE regerror
```

```
 1360         RTS
 1370 .regerror
 1380         BRK
 1390         BRK
 1400         EQUS "Special register error"
 1410         BRK
 1420 .printbyte
 1430         LDA physical  \ print physical track number
 1440         PHA
 1450         LSR A
 1460         LSR A
 1470         LSR A
 1480         LSR A
 1490         JSR nybble    \ print MS nybble
 1500         PLA
 1510         JSR nybble    \ print LS nybble
 1520         LDA #ASC(" ")
 1530         LDX verblock+10 \ load deleted data flag
 1540         BEQ space     \ if =0 not deleted
 1550         LDA #ASC("d") \ deleted data mark
 1560 .space
 1570         JSR oswrch    \ print space or "d"
 1580         LDA #ASC(" ")
 1590         JMP oswrch    \ print space
 1600 .nybble
 1610         AND #&0F
 1620         SED
 1630         CLC
 1640         ADC #&90
 1650         ADC #&40
 1660         CLD
 1670         JMP oswrch    \ print nybble and return
 1680 .seekblock
 1690         EQUB &FF      \ current drive
 1700         EQUD &00      \ does not matter
 1710         EQUD &00006901 \ seek, 1 parameter
 1720 .idsblock
 1730         EQUB &FF      \ current drive
 1740         EQUD buffer   \ address of buffer
 1750         EQUD &00005B03 \ read sector ids
 1760         EQUW &00
 1770 .verblock
 1780         EQUB &FF      \ current drive
 1790         EQUD &00      \ does not matter
 1800         EQUD &00005F03 \ verify multi sector
 1810         EQUW &00
 1820 .regblock
 1830         EQUB &FF      \ current drive
 1840         EQUD &00      \ does not matter
 1850         EQUD &00127A02
 1860         EQUB &00      \ result
 1870 .sizes
 1880         EQUB 18
 1890         EQUB 10
 1900         EQUB 5
 1910         EQUB 2
 1920         EQUB 1
 1930 .physical
 1940         EQUB &00
 1950 .sectornumber
 1960         EQUB &00
 1970 .last
 1980         EQUB &00
 1990 ]
 2000 NEXT
 2010 INPUT'"Number of tracks (40/80) "tracks$
 2020 IF tracks$="40" ?last = 40 ELSE ?last = 80
 2030 PRINT"Insert ";?last;" track disc into current drive"
 2040 PRINT"and press the Spacebar to verify"
 2050 REPEAT
 2060 UNTIL GET=32
 2070 CALL mcode
```

The program is commented to explain how it works and you might like to come back to it after reading module 2. When the program VERIFY was used on the "Grand Prix Construction Set" disc the output shown in figure 5 was produced.

```
>LO."VERIFY"
>RUN

Number of tracks (40/80) 40
Insert 40 track disc into current drive
and press the Spacebar to verify

00  01  02  03  04  05  06  07  08  09
0A  0B  0C  0Dd 0Ed 0Fd 10d 11d 12  13
14  15  16  17  18  19d 1Ad 1Bd 1Cd 1Dd
1Ed 1Fd 20d 21d 22d 23  24  25  26  27

>
```

Figure 5. The output from the program VERIFY

The lower case letter d following tracks &0D to &11 and tracks &19 to &22 indicates that these tracks have deleted data stored on them. The game on this disc loads in two parts and I would not be at all surprised to find that the two parts are stored on the tracks with deleted data. If the program VERIFY is used with a standard DFS disc it will not produce a lower case d to indicate the use of the deleted data marker.

The Acorn DFS Osword commands - by - Gordon Horsington

Module 1. The DFS Osword commands (part 1)

```
+----------------------------------------------------------+
| All the DFS modules in  this  series  use programs which |
| experiment  with the format and contents of discs. These |
| experiments may  have disasterous effects if you use any |
| of the programs  on  discs  which store programs or data |
| which you cannot afford to lose.  You  should  first try |
| out  the  programs  using  discs  that  have either been |
| duplicated or, better still, have not been used at all.  |
+----------------------------------------------------------+
```

The DFS ROM intercepts and recognises three Osword calls. Osword &7D reads the number of times a disk has been written, Osword &7E reads the number of sectors on a disc, and Osword &7F executes the 8271 disc controller commands.

Osword &7D

Osword &7D reads the catalogue for the current default disc, as specified by the most recent *DRIVE command, and extracts the number of disc cycles from the catalogue. The number of disc cycles is a BCD number in the range from 0 to 99. This number gives some indication of how many times the disc has been written. Because The disc cycle number restarts at 0 after reaching 99 this is not a reliable count.

The catalogue is read by Osword &7D and stored in the first two pages of the paged ROM absolute workspace (pages &0E and &0F with OS 1.2). The result is stored in a one byte parameter block specified by X and Y registers on entry. The program CYCLES demonstrates how Osword &7D can be used to read the disc cycles.

```
   10 REM: CYCLES
   20 osword=&FFF1
   30 DIM mcode &100
   40 FOR pass = 0 TO 2 STEP 2
   50 P%=mcode
   60 [       OPT pass
   70         LDA #&7D
   80         LDX #result MOD 256
   90         LDY #result DIV 256
  100         JSR osword
  110         RTS
  120 .result
  130         EQUB &00
  140 ]
  150 NEXT
  160 CALL mcode
  170 PRINT"Disc cycles = ";~?result
```

Osword &7E

Osword &7E reads the catalogue for the current default disc, as specified by the most recent *DRIVE command, and extracts the number of sectors available on the disc. There are 800 (&320) sectors on an 80 track Acorn DFS disc and 400 (&190) sectors on a 40 track Acorn DFS disc. Non-standard and copy-protected discs may have different numbers of available sectors.

The catalogue is read by Osword &7E and stored in the first two pages of the paged ROM absolute workspace (pages &0E and &0F with OS 1.2). The result is stored in a four byte parameter block specified by the X and Y registers on entry. The program HOWMANY demonstrates how Osword &7E can be used to read the number of available sectors on a DFS disc. The least significant byte of the number of sectors will be in byte &01 of the result and the most significant byte in byte &02 of the result. Bytes &00 and &03 of the result should always be zero.

```
   10 REM: HOWMANY
   20 osword=&FFF1
   30 DIM mcode &100
   40 FOR pass = 0 TO 2 STEP 2
   50 P%=mcode
   60 [       OPT pass
   70         LDA #&7E
   80         LDX #result MOD 256
   90         LDY #result DIV 256
  100         JSR osword
  110         RTS
  120 .result
  130         EQUD &00
  140 ]
  150 NEXT
  160 CALL mcode
  170 PRINT"&";~result?2;~result?1;" Sectors"
```

Osword &7F

Osword &7F executes the 8271 disc controller commands. If the disc interface uses a 1770 disc controller then Osword &7F emulates the 8271 command set using the 1770. The complete command set executed by Osword &7F is shown in figure 1. Not all the 8271 commands can be emulated by the 1770 and the Osword &7F commands from &6C to &7D are not fully implemented with the Acorn 1770 disc interface.

When you write software which uses the Osword &7F commands from &6C to &7D you should take care to ensure that your programs will work with the 1770 disc controller as well as with the 8271 interface. Some of the example programs used in this module use these partly implemented Osword &7F commands and may not work as expected with all 1770 disc interfaces. The programs used to illustrate the other Osword &7F commands all work with the Acorn 1770 DFS.

Command number	Param- eters	1770	Action
&4A	2	Yes	Write data 128 bytes
&4B	3	Yes	Write data multi-sector
&4E	2	Yes	Write deleted data 128 bytes
&4F	3	Yes	Write deleted data multi-sector
&52	2	Yes	Read data 128 bytes
&53	3	Yes	Read data multi-sector
&56	2	Yes	Read data and deleted data 128 bytes
&57	3	Yes	Read data and deleted data multi-sector
&5B	3	Yes	Read sector ids
&5E	2	Yes	Verify data and deleted data 128 bytes
&5F	3	Yes	Verify data and deleted data multi-sector
&63	5	Yes	Format track
&69	1	Yes	Seek
&6C	0	Part	Read drive status
&75	4	Part	Initialise 8271
&75	4	Part	Load bad tracks
&7A	2	Part	Write special register
&7D	1	Part	Read special register

Figure 1. The Osword &7F command set

Osword &7F uses a variable length parameter block the address of which is specified by the X and Y registers on entry.

Byte &00 of the parameter block is used to store the number of the disc drive to be used with the command. If a negative drive number is used (&80 to &FF) then Osword &7F uses the currently selected drive as specified by the most recent *DRIVE command.

Bytes &01 to &04 of the parameter block store the address of a buffer area into which any data to be read will be placed, or from which any data to be written will be taken. A buffer is not needed by all the commands but bytes &01 to &04 of the parameter block are always assigned to a buffer address even if a buffer is not used.

Column 2 of figure 1 shows that the Osword &7F commands use from 0 to 5 parameters. The number of parameters used by a call is stored in byte &05 of its parameter block. Byte &06 of the parameter block stores the command number (column 1, figure 1), and byte &07 onwards the parameters required by the command. The last byte of the parameter block is a result byte. Unless it is used to read the 8271 registers, Osword &7F will normally return the number zero in the result byte when a command has been executed sucessfully but it returns the number &20 (ie. bit 5 set) if deleted data have been sucessfully written to, read from, or verified on a disc.

The result is returned in byte &07 of the parameter block for Osword &7F command number &6C, which uses no parameters. It is returned in byte &08 of the parameter block for command numbers &69 and &7D, which use one parameter, and so on up to byte &0C of the parameter block for command number &63 which uses 5 parameters.

Errors are reported in bits 1 to 4 of the result byte. The error codes can be isolated by ANDing the result byte with #&1E and the error codes can be interpreted as shown in figure 2.

Result	Interpretation
&02	Scan met equal **
&04	Scan met not equal **
&08	Clock error
&0A	Late DMA **
&0C	Sector ID CRC error
&0E	Data CRC error
&10	Drive not ready
&12	Disc write protected
&14	Physical track 0 not found
&16	Write fault
&18	Sector not found
Errors marked ** should not occur	

Figure 2. The error codes returned in the result byte

When designing software which uses Osword &7F you can test the result byte for specific errors after ANDing the result with #&1E to isolate the error bits. ANDing with #&1E excludes the deleted data bit which is not really an error at all. Testing for specific errors is not essential because all the above errors are fatal and any error should be used to halt your program. Testing for specific errors can always give some useful extra information when a routine fails to work as expected.

In the rest of this module and whole of the next module I will discuss the commands executed by Osword &7F and give short examples of the some of them. Later in the series these commands will be used to show you how to create disc utility programs.

The order in which the commands will be discussed is not the order in which they are listed in figure 1. I will cover the Osword &7F command numbers &69 to &7D in this module. With the exception of the Seek command, these commands are not fuly implemented with the 1770 disc interface but they are effectivly incorporated in other commands such as Write Data, Read Data, and Verify. The Osword &7F command numbers &4A to &63 will be covered in the next module.

Osword &7F Read Drive Status

The Osword &7F Read Drive Status command copies the 8271 drive control input register to the parameter block result byte. To use this command set up the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &00 (no command parameters)
Parameter block &06       = &6C (read drive status command)
Parameter block &07       = result byte
```

Each bit of the result byte has the following meaning:

```
bit 7 = unused
bit 6 = READY1
bit 5 = FAULT
bit 4 = INDEX
bit 3 = WR PROTECT
bit 2 = READY0
bit 1 = TRK0
bit 0 = COUNT/OP1
```

These results are only really useful for tracking down hardware errors. You can use the demonstration program STATUS with both write protected and write enabled discs to see the effect that write protection has on the bits of the result register. You could use this command to test for a write protection tab on a disc but the only unique use for the Osword &7F Read Drive Status command is to clear a "not ready" signal. It is used by the DFS for this purpose. This is not one of the most useful commands and you will probably not need to use it in any of your programs.

```
   10 REM: STATUS
   20 oswrch=&FFEE
   30 osword=&FFF1
   40 DIM mcode &100
   50 FOR pass = 0 TO 2 STEP 2
   60 P%=mcode
   70 [       OPT pass
   80         LDA #&7F
   90         LDX #block MOD 256
  100         LDY #block DIV 256
  110         JSR osword
  120         RTS
  130 .block
  140         EQUB &FF      \ current drive
  150         EQUD &00      \ does not matter
  160         EQUB &00      \ 0 parameters
  170         EQUB &6C      \ read status command
  180 .result
  190         EQUB &00      \ result byte
  200 .binary
  210         LDX #8
  220 .loop
  230         LDA #ASC("0")
  240         ASL result
  250         ADC #&00
  260         JSR oswrch
  270         DEX
  280         BNE loop
  290         RTS
  300 ]
  310 NEXT
  320 CALL mcode
  330 PRINT"Result = &";~?result;", %";
  340 CALL binary
  350 PRINT
```

Osword &7F Initialise 8271

Osword &7F Initialise 8271 is not fully implemented with the 1770 disc interface. For this reason you should avoid using it and use the equivalent Osbyte &FF which is available from the operating system of all BBC microcomputers.

The hardware default setting is equivalent to *FX 255,0,255 and this gives access to the slowest disc drives. The Osbyte calls in figure 3 can be used to give access to faster drives but, if you are writing software to be used on unknown drives, it may be a good idea to select the slowest time.

Osbyte &FF passes the values of the drive step time, settlement time, and head load time to the disc controller on soft break and they will remain in force until a hard break.

Step time	Settle time	Load time	Osbyte &FF
4	16	0	*FX 255,0,207
6	16	0	*FX 255,0,223
6	50	32	*FX 255,0,239
24	20	64	*FX 255,0,255

Figure 3. Disc access timings

If you need to use the Osword &7F Initialise 8271 command then the following parameter block must be used.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &04 (4 command parameters)
Parameter block &06       = &75 (initialise 8271 command)
Parameter block &07       = &0D (init 8271 marker)
Parameter block &08       = drive step time (milliseconds / 2)
Parameter block &09       = head settlement time (miliseconds / 2)
Parameter block &0A       = head unload/load time (two 4 bit numbers)
Parameter block &0B       = result byte
```

The drive step time should be in the range from &01 to &FF, representing 2 to 510 milliseconds in 2 millisecond steps. A drive step time of zero indicates that the drive will provide its own step pulses.

The head settlement time should be in the range &00 to &FF representing a delay of 0 to 512 miliseconds in 2 millisecond steps.

The four most significant bits of the head unload/load time should be in the range %0000 to %1110 (0-14) representing the number of complete disc revolutions before the head is unloaded. %1111 specifies that the head should not be unloaded at all. The four least significant bits of the head unload/load time specify the time taken to load the head in 8 milisecond intervals. This is a number in the range %0000 to %1111 (0-15) representing head load times of 0 to 120 milliseconds.

The following code could be used to initialise the 8271.

```
        LDA #&7F
        LDX #block MOD 256
        LDY #block DIV 256
        JSR &FFF1
        RTS
.block
        EQUB &FF          \ current drive
        EQUD &00          \ buffer address (not used)
        EQUB &04          \ 4 parameters
        EQUB &75          \ init 8271 command
        EQUB 12           \ 24 milliseconds step time
        EQUB 10           \ 20 milliseconds settle time
        EQUB &C8          \ Unload = 12 revs, load =64 milliseconds
        EQUB &00          \ result byte
```

Osword &7F Seek

The Osword &7F Seek command uses the appropriate track register as a base from which to seek a specified physical track. Register number &12 is used for drive 0/2 and register number &1A is used for drive 1/3. This command does not load the head and does not check the sector IDs. If track 0 is specified the seek command steps the head outwards until it trips the track 0 switch. If the TRK0 signal is missing after 255 attempts to find it, the command reports error &14 in the result byte. Error &14 is physical track zero not found (see figure 2).

Seek track 0 can be used to find a base from which to seek any other physical track. This can be useful if the track register contains an unknown or incorrect physical track number.

The following parameter block is used to seek a physical track.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &01 (1 command parameter)
Parameter block &06       = &69 (seek command)
Parameter block &07       = physical track number
Parameter block &08       = result byte
```

The Osword &7F Seek command is useful if, for example, you want to write data onto a copy-protected disc which uses different physical and logical track numbers. You would use it to seek the physical track number and then use the Osword &7F Write Special Register command to write the logical track number into the appropriate track register. After writing to the disc you should then either rewrite the physical track number into the appropriate track register or seek track 0.

The following code could be used to seek track 0 on the current drive.

```
        LDA #&7F
        LDX #block MOD 256
        LDY #block DIV 256
        JSR &FFF1
        RTS
.block
        EQUB &FF          \ current drive
        EQUD &00          \ buffer address (not used)
        EQUB &01          \ 1 parameter
        EQUB &69          \ seek command
        EQUB &00          \ track 0
        EQUB &00          \ result byte
```

Osword &7F Load Bad Tracks

This command is used to tell the 8271 that there are one or two "bad tracks" on a disc. This command is not fully implemented in the 1770 disc interface and is not used by either DFS. Because it is not used by the DFS it can be used for copy-protecting discs when the DFS *BACKUP command will give the "disc fault" error. Copy-protection will be covered in detail in module 6.

Two bad track registers are available for each disc surface and they are used to specify which tracks are to be totally ignored by the 8271. For example, if track 1 is bad the 8271 will use track 3 when track 2 is specified. As far as the disc controller is concerned physical track 3 is seen as physical track 2, physical track 4 is seen as physical track 3, and so on.

When bad tracks are used their track number IDs should be set to &FF when the disc is formatted. Track 0 must not be set as a bad track. The command lasts until a hard reset.

The following parameter block is used by Osword &7F Load Bad Tracks.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &04 (4 command parameters)
Parameter block &06       = &75 (load bad tracks command)
Parameter block &07       = drive pair (&10 = drive 0/2, &18 = drive 1/3)
```

```
Parameter block &08       = bad track number 1
Parameter block &09       = bad track number 2
Parameter block &0A       = current physical track
Parameter block &0B       = result byte
```

The following code could be used to load bad tracks 1 and 2. Osword &7F Seek command is used to position the head over a known physical track, in this case track zero.

```
        JSR seekzero      \ seek track zero
        LDA #&7F
        LDX #block MOD 256
        LDY #block DIV 256
        JSR &FFF1
        RTS
.block
        EQUB &00          \ drive 0
        EQUD &00          \ buffer address (not used)
        EQUB &04          \ 4 parameters
        EQUB &75          \ load bad tracks command
        EQUB &10          \ drive 0/2
        EQUB &01          \ track 1
        EQUB &02          \ track 2
        EQUB &00          \ current track
        EQUB &00          \ result byte
```

Osword &7F Write Special Register

The internal registers of the 8271 can be overwritten using the Osword &7F Write Special Register command. The contents of all the registers in figure 4 can be altered but you are advised to limit yourself to altering the track registers, numbers &12 and &1A. The bad track registers have their own Osword &7F Load Bad Tracks command described above but they can also be altered with the Osword &7F Write Special Register command. If you decide to alter any other registers the results are likely to be disasterous - you have been warned!

Register no.	Register
&06	Scan sector register
&10	Bad track register 1, drive 0/2
&11	Bad track register 2, drive 0/2
&12	Track register, drive 0/2
&13	Scan count register (LSB)
&14	Scan count register (MSB)
&17	DMA mode register
&18	Bad track register 1, drive 1/3
&19	Bad track register 2, drive 1/3
&1A	Track register, drive 1/3
&22	Drive control input register
&23	Drive control output register

Figure 4. The 8271 registers

The following parameter block must be used with the Osword &7F Write Special Register command.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &7A (write special register command)
Parameter block &07       = register number (from figure 4)
Parameter block &08       = value to put in register
Parameter block &09       = result byte
```

The track registers &12 (drive 0/2) and &1A (drive 1/3) are used by the disc controller to identify its current track position. These registers contain the number of the physical track. If the logical track number stored in the sector ID is not the same as the physical track number you should always use the write special register command to store the logical track number in the track register before attempting to read from or write to the disc. You must always reset the track register to its original value after reading or writing. As you have seen in the module 0, using different logical and physical track numbers is yet another technique for copy-protecting discs.

The following code could be used to seek track 0 and then load the track register for drive 0/2 with the number 1. Track zero will then be seen as physical track 1.

```
        JSR seekzero      \ seek track zero
        LDA #&7F
        LDX #block MOD 256
        LDY #block DIV 256
        JSR &FFF1
        RTS
.block
        EQUB &00          \ drive 0
        EQUD &00          \ buffer address (not used)
        EQUB &02          \ 2 parameters
        EQUB &7A          \ write special register command
        EQUB &12          \ track register drive 0/2
        EQUB &01          \ track 1
        EQUB &00          \ result byte
```

Osword &7F Read Special Register

The internal registers of the 8271 can be read using Osword &7F Read Special Register. The contents of all the registers in figure 4 can be read but not all the registers give useful results. The drive control input register can be read using Osword &7F Read Drive Status command but the drive control output register, and all the other registers, have to be read with the Osword &7F Read Special Register command.

The following parameter block must be used with the Osword &7F Read Special Register command.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &01 (1 command parameter)
Parameter block &06       = &7D (read special register command)
Parameter block &07       = register number (from figure 4)
Parameter block &08       = result byte
```

The program OUTPUT demonstrates how to read the contents of the drive control output register for the current drive. The result is printed in hexadecimal and binary. Each bit of the result has the following meaning:

```
bit 7 = SELECT 1
bit 6 = SELECT 0
bit 5 = FAULT RESET?OP0
bit 4 = LOW CURRENT
bit 3 = LOAD HEAD
bit 2 = DIRECTION
bit 1 = SEEK/STEP
bit 0 = WR ENABLE
```

It is interesting to run the program OUTPUT with drive 0 selected (using *DRIVE 0) and then select drive 1 (with *DRIVE 1) and run the program again to see the difference it makes to the result. If you want to read any other register change the value &23 in line 180 for another register number taken from figure 4.

```
   10 REM: OUTPUT
   20 oswrch=&FFEE
   30 osword=&FFF1
```

```
 40 DIM mcode &100
 50 FOR pass = 0 TO 2 STEP 2
 60 P%=mcode
 70 [       OPT pass
 80         LDA #&7F
 90         LDX #block MOD 256
100         LDY #block DIV 256
110         JSR osword
120         RTS
130 .block
140         EQUB &FF      \ current drive
150         EQUD &00      \ does not matter
160         EQUB &01      \ 1 parameter
170         EQUB &7D      \ read special register command
180         EQUB &23      \ drive control output register
190 .result
200         EQUB &00      \ result byte
210 .binary
220         LDX #8
230 .loop
240         LDA #ASC("0")
250         ASL result
260         ADC #&00
270         JSR oswrch
280         DEX
290         BNE loop
300         RTS
310 ]
320 NEXT
330 CALL mcode
340 PRINT"Result = &";~?result;", %";
350 CALL binary
360 PRINT
```

The Acorn DFS Osword commands - by - Gordon Horsington

Module 2. The DFS Osword commands (part 2)

```
+-----------------------------------------------------------+
| All the DFS modules in  this  series  use programs which |
| experiment  with the format and contents of discs. These |
| experiments may  have disasterous effects if you use any |
| of the programs  on  discs  which store programs or data |
| which you cannot afford to lose.  You  should  first try |
| out  the  programs  using  discs  that  have either been |
| duplicated or, better still, have not been used at all.  |
+-----------------------------------------------------------+
```

Osword &7F

Osword &7F executes the 8271 disc controller commands. If the disc interface uses a 1770 disc controller then Osword &7F emulates the 8271 command set using the 1770. The complete command set executed by Osword &7F is shown in figure 1. Osword &7F commands from &69 to &7D were covered in module 1. In this module I will explain how to use the Osword &7F command numbers &4A to &63.

Command number	Param- eters	1770	Action
&4A	2	Yes	Write data 128 bytes
&4B	3	Yes	Write data multi-sector
&4E	2	Yes	Write deleted data 128 bytes
&4F	3	Yes	Write deleted data multi-sector
&52	2	Yes	Read data 128 bytes
&53	3	Yes	Read data multi-sector
&56	2	Yes	Read data and deleted data 128 bytes
&57	3	Yes	Read data and deleted data multi-sector
&5B	3	Yes	Read sector ids
&5E	2	Yes	Verify data and deleted data 128 bytes
&5F	3	Yes	Verify data and deleted data multi-sector
&63	5	Yes	Format track
&69	1	Yes	Seek
&6C	0	Part	Read drive status
&75	4	Part	Initialise 8271
&75	4	Part	Load bad tracks
&7A	2	Part	Write special register
&7D	1	Part	Read special register

Figure 1. The Osword &7F commands

Osword &7F Read Sector IDs

Before using this command you should ensure that the appropriate track register contains the current physical track number or use the Osword &7F Seek command to seek track 0. Osword &7F Read Sector IDs uses the appropriate track register as a base from which to seek a specified physical track. It then reads the required number of sector IDs and stores them in the buffer area specified in the parameter block. Sector IDs are transfered into the buffer area in physical sector order. The command returns four bytes For each sector ID that it reads.

byte 0 = logical track number
byte 1 = head number
byte 2 = logical sector number
byte 3 = data size code (0=128, 1=256, ... 4=2048)

The following parameter block is used by Osword &7F read sector IDs:

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address for sector IDs
Parameter block &05       = &03 (3 command parameters)
Parameter block &06       = &5B (read sector IDs command)
Parameter block &07       = physical track number
Parameter block &08       = &00
Parameter block &09       = number of IDs to be read
Parameter block &0A       = result byte
```

One of many possible uses for this command is to check if a disc has been formatted. If Osword &7F Read Sector IDs fails to find at least one sector on track zero the most probable reason is that the disc has not been formatted and this will be reported as error number &18, sector not found (see figure 2). The program NOTFORM uses this idea to see if a disc in the current drive has been formatted.

Result	Interpretation
&02	Scan met equal **
&04	Scan met not equal **
&08	Clock error
&0A	Late DMA **
&0C	Sector ID CRC error
&0E	Data CRC error
&10	Drive not ready
&12	Disc write protected
&14	Physical track 0 not found
&16	Write fault
&18	Sector not found
Errors marked ** should not occur	

Figure 2. The error codes returned in the result byte

```
   10 REM: NOTFORM
   20 osword=&FFF1
   30 DIM mcode &100
   40 FOR pass = 0 TO 2 STEP 2
   50 P%=mcode
   60 [       OPT pass
   70         LDA #&7F
   80         LDX #block MOD 256
   90         LDY #block DIV 256
  100         JSR osword
  110         RTS
  120 .block
  130         EQUB &FF      \ current drive
  140         EQUD buffer   \ buffer address
  150         EQUB &03      \ 3 parameters
  160         EQUB &5B      \ read sector IDs
  170         EQUB &00      \ track 0
  180         EQUB &00
  190         EQUB &01      \ read 1 sector
  200 .result
  210         EQUB &00      \ result byte
  220 .buffer
  230         EQUD &00
  240 ]
  250 NEXT
  260 CALL mcode
  270 IF ?result=&18 PRINT"Disc not formatted" ELSE PRINT"Disc formatted"
```

The Osword &7F Read Sector IDs command was used in the program IDSDUMP used in module 0. After working through this module you might like to go back to module 0 and look again at the example programs.

Osword &7F Format Track

The Osword &7F Format Track command uses a sector table stored in the buffer specified in the parameter block. This table contains four bytes for every ID field on the track. For each sector to be on the disc the following bytes must be stored in the buffer:

byte 0 = logical track number (&00-&FF)
byte 1 = head number (use &00)
byte 2 = logical sector number (&00-&FF)
byte 3 = data size code (0=128, 1=256, ... 4=2048)

The following parameter block is used with the Osbyte &7F Format Track command:

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address for sector IDs
Parameter block &05       = &05 (5 command parameters)
Parameter block &06       = &63 (format track command)
Parameter block &07       = physical track number
Parameter block &08       = gap 3 size (see figure 2)
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = gap 5 size (always use &00)
Parameter block &0B       = gap 1 size (always use &10)
Parameter block &0C       = result byte
```

The gap 3 size can be taken from figure 3 (all numbers in figure 3 are in decimal).

The most significant 3 bits of the number stored in parameter block &09 contain the sector size code (column 2 of figure 3). The least significant 5 bits contain the number of sectors per track. To calculate the number to be stored in parameter block &09 multiply the size code by 32 and add the number of sectors. For example, if you want to format a track with five sectors of 512 bytes then parameter block &09 will contain 2*32+5 = 69 = &45.

No. Sectors	Size code	Length	Gap1	Gap2	Gap3	Gap4	Gap5
18	0	128	16	11	11	24	0
10	1	256	16	11	21	30	0
5	2	512	16	11	74	88	0
2	3	1024	16	11	255	740	0
1	4	2048	16	11	0	1028	0

Figure 3. The relationship between sector size code, length and gap size.

The Osword &7F Format Track command uses the appropriate track register as a base from which to seek the specified physical track. It then uses the sector IDs stored in the buffer to create the ID fields for each sector. It calculates and writes the ID field CRC bytes, creates the correct gap sizes, fills the data fields with bytes of &E5, and calculates and writes the data field CRC bytes.

The program FORM10 can be used to format physical track &27 of the disc in the current drive with 10 sectors of 256 bytes. This program will destroy all the data stored on track &27 - you have been warned!

The buffer used in the program FORM10 stores 4 bytes for each sector. Taking physical sector &00 in line 240 as an example, The bytes &27, &00, &00 and &01 are stored (the order of the bytes is reversed with the EQUD command). These four bytes represent the logical track number, the head number, the logical sector number and the data size code. You can alter the logical track and sector numbers and use the IDSDUMP program from module 0 to see the effect this has on the format of the disc. If you alter the logical track number you will be unable to *BACKUP the disc.

```
   10 REM: FORM10
   20 DIM mcode &100
```

```
   30 osword=&FFF1
   40 FORpass=0 TO 2 STEP 2
   50 P%=mcode
   60 [       OPT pass
   70         LDA #&7F
   80         LDX #block MOD 256
   90         LDY #block DIV 256
  100         JSR osword
  110         RTS
  120 .block
  130         EQUB &FF      \ current drive
  140         EQUD buffer   \ address of sector table
  150         EQUB &05      \ 5 parameters
  160         EQUB &63      \ format track command
  170         EQUB &27      \ physical track &27
  180         EQUB 21       \ gap 3 (from figure 2)
  190         EQUB &2A      \ 10 sectors of 256 bytes
  200         EQUB &00      \ gap 5 (always &00)
  210         EQUB &10      \ gap 1 (always &10)
  220         EQUB 0        \ result byte
  230 .buffer
  240         EQUD &01000027
  250         EQUD &01010027
  260         EQUD &01020027
  270         EQUD &01030027
  280         EQUD &01040027
  290         EQUD &01050027
  300         EQUD &01060027
  310         EQUD &01070027
  320         EQUD &01080027
  330         EQUD &01090027
  340 ]
  350 NEXT
  360 CALL mcode
```

There is more to formatting a disc than just formatting all the tracks. It is also necessary to create an empty catalogue on track 0 of the disc. Formatting discs will be covered in more detail in module 3.

Osword &7F Verify Data and Deleted Data multi-sector

Osword &7F Verify Data and Deleted Data multi-sector uses the appropriate track register as a base from which to seek the track and sector specified in the parameter block. It attempts to verify the sector and returns &00 in the result byte if it is successful. It returns &20 in the result byte if deleted data have been successfully verified. If more than one sector is specified this procedure repeats until either all the sectors have been verified or an error occurs.

The following parameter block is used with the Osbyte &7F Verify Data and Deleted Data multi-sector command:

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &03 (3 command parameters)
Parameter block &06       = &5F (verify multi-sector command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = result byte
```

The most significant 3 bits of the number stored in parameter block &09 contain the sector size code (column 2 of figure 3). The least significant 5 bits contain the number of sectors per track. To calculate the number to be stored in parameter block &09 multiply the size code by 32 and add the number of sectors. For example, if you want to verify a track with ten sectors of 256 bytes then parameter block &09 will contain 1*32+10 = 42 = &2A.

If you want to verify discs that use different physical and logical track numbers it is necessary to use the Osword &7F Seek command to find the appropriate track, Osword &7F Read Sector IDs to read the logical track and sector number, and Osword &7F Write Special Register to write the logical track number into the appropriate track register before using Osword &7F Verify Data and Deleted Data multi-sector. After verifying the sector(s) it is then necessary to use Osword &7F Write Special Register to write the physical track number back into the appropriate track register. This procedure was used in the program VERIFY in module 0.

The following code could be used to verify track &27 of the disc formatted with the program FORM10

```
        LDA #&7F
        LDX #block MOD 256
        LDY #block DIV 256
        JSR &FFF1
        RTS
.block
        EQUB &FF          \ current drive
        EQUD &00          \ buffer address (not used)
        EQUB &03          \ 3 parameters
        EQUB &5F          \ verify command
        EQUB &27          \ logical track &27
        EQUB &00          \ start with logical sector &00
        EQUB &2A          \ 10 sectors of 256 bytes
        EQUB &00          \ result byte
```

Osword &7F Verify Data and Deleted Data 128 bytes

If you need to verify just one sector of 128 bytes you can use the Osword &7F Verify Data and Deleted Data 128 bytes command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &5E (verify 128 bytes command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = result byte
```

Although using this command saves you the trouble of calculating the sector size/number of sectors parameter, I cannot recommend using it when the multi-sector command is so much more versitile.

Osword &7F Write Data multi-sector

Osword &7F Write Data multi-sector uses the appropriate track register as a base from which to seek the track and sector specified in the parameter block. If it cannot find the track and sector it returns the sector not found error (&18) in the result byte. If it finds the required track and sector it writes a data mark at the start of the data field and copies the first sector of data from the buffer to the specified sector. If more than one sector is specified this procedure repeats until either all the sectors have been written or an error occurs.

The following parameter block is used with the Osword &7F Write Data multi-sector command.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address
Parameter block &05       = &03 (3 command parameters)
Parameter block &06       = &4B (write data multi-sector command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = result byte
```

The most significant 3 bits of the number stored in parameter block &09 contain the sector size code (column 2 of figure 3). The least significant

5 bits contain the number of sectors per track. To calculate the number to be stored in parameter block &09 multiply the size code by 32 and add the number of sectors. For example, if you want to verify a track with ten sectors of 256 bytes then parameter block &09 will contain 1*32+10 = 42 = &2A.

The program WRITE10 can be used to demonstrate this command. This program will destroy all the data stored on track &01 of the disc it uses - you have been warned!

WRITE10 uses an normally formatted DFS disc and stores 2.5k of data on track &01. The buffer starts at PAGE (line 240) and so the program could be used to store a small BASIC program on a disc. The data are stored without an entry in the DFS catalogue and cannot be read using any of the DFS star commands. If you use the DFS commands to store any other data on the disk, the data stored with WRITE10 could be overwritten. The data can be read from the disc using the Osword &7F Read Data and Deleted Data command which is explained later in this module.

```
   10 REM: WRITE10
   20 mcode = &0A00
   30 osword=&FFF1
   40 page = PAGE
   50 FORpass=0 TO 2 STEP 2
   60 P%=mcode
   70 [       OPT pass
   80         LDA #&7F
   90         LDX #block MOD 256
  100         LDY #block DIV 256
  110         JSR osword
  120         LDA result
  130         BEQ ok
  140         BRK
  150         BRK
  160         EQUS "Write error"
  170 .ok
  180         BRK
  190         EQUS "Write sucessful"
  200         BRK
  210         BRK
  220 .block
  230         EQUB &FF      \ current drive
  240         EQUD page     \ start at PAGE
  250         EQUB &03      \ 3 parameters
  260         EQUB &4B      \ write data multi-sector
  270         EQUB &01      \ logical track 1
  280         EQUB &00      \ start logical sector 0
  290         EQUB &2A      \ 10 sectors of 256 bytes
  300 .result
  310         EQUB &00      \ result byte
  320 ]
  330 NEXT
  340 PRINT''"Type: CALL &";~mcode;" to save 10 sectors"'
```

Osword &7F Write Data 128 bytes

If you need to write just one sector of 128 bytes you can use the Osword &7F Write Data 128 bytes command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &4A (write data 128 bytes command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = result byte
```

Although using this command saves you the trouble of calculating the sector size/number of sectors parameter, I cannot recommend using it when the multi-sector command is so much more versitile.

Osword &7F Write Deleted Data multi-sector

Osword &7F Write Deleted Data multi-sector uses the appropriate track register as a base from which to seek the track and sector specified in the parameter block. If it cannot find the track and sector it returns the sector not found error (&18) in the result byte. If it finds the required track and sector it writes a deleted data mark at the start of the data field and copies the first sector of data from the buffer to the specified sector. If more than one sector is specified this procedure repeats until either all the sectors have been written or an error occurs.

The following parameter block is used with the Osword &7F Write Deleted Data multi-sector command.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address
Parameter block &05       = &03 (3 command parameters)
Parameter block &06       = &4F (write deleted data multi-sector command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = result byte
```

The most significant 3 bits of the number stored in parameter block &09 contain the sector size code (column 2 of figure 3). The least significant 5 bits contain the number of sectors per track. To calculate the number to be stored in parameter block &09 multiply the size code by 32 and add the number of sectors. For example, if you want to verify a track with ten sectors of 256 bytes then parameter block &09 will contain 1*32+10 = 42 = &2A.

The program WRITE10 can be modified to demonstrate this command. Alter line 260 from EQUB &4B to EQUB &4F. Using deleted data will effectivly disable the *BACKUP command. Don't forget that this program will destroy all the data stored on track &01 of the disc it uses - you have been warned!

Osword &7F Write Deleted Data 128 bytes

If you need to write just one sector of 128 bytes of deleted data you can use the Osword &7F Write Deleted Data 128 bytes command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &4F (write deleted data 128 bytes command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = result byte
```

Although using this command saves you the trouble of calculating the sector size/number of sectors parameter, I cannot recommend using it when the multi-sector command is so much more versitile.

Osword &7F Read Data and Deleted Data multi-sector

Osword &7F Read Data and Deleted Data multi-sector uses the appropriate track register as a base from which to seek the track and sector specified in the parameter block. If it cannot find the track and sector it returns the sector not found error (&18) in the result byte. If it finds the required track and sector it copies the first sector of data into the buffer specified in the parameter block. If more than one sector is specified this procedure repeats until either all the sectors have been read or an error occurs.

The following parameter block is used with the Osword &7F Write Deleted Data multi-sector command.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address
Parameter block &05       = &03 (3 command parameters)
Parameter block &06       = &57 (read data and deleted data multi-sector)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = result byte
```

The most significant 3 bits of the number stored in parameter block &09 contain the sector size code (column 2 of figure 3). The least significant 5 bits contain the number of sectors per track. To calculate the number to be stored in parameter block &09 multiply the size code by 32 and add the number of sectors. For example, if you want to verify a track with ten sectors of 256 bytes then parameter block &09 will contain 1*32+10 = 42 = &2A.

The program READ10 can be used to demonstrate this command. This program reads the data written onto a disc by the program WRITE10, which was used to illustrate the Osword &7F Write Data multi-sector command.

READ10 uses an normally formatted DFS disc and reads 2.5k of data from track &01. The buffer starts at PAGE (line 240) and so the program could be used to read a small BASIC program from a disc. The programs WRITE10, with the Osword &7F write deleted data command, and READ10 could be used to create a copy-protected disc. I will return to copy-protection in later modules of this series.

```
   10 REM: READ10
   20 mcode = &0A00
   30 osword=&FFF1
   40 page = PAGE
   50 FORpass=0 TO 2 STEP 2
   60 P%=mcode
   70 [       OPT pass
   80         LDA #&7F
   90         LDX #block MOD 256
  100         LDY #block DIV 256
  110         JSR osword
  120         LDA result
  130         BEQ ok
  140         BRK
  150         BRK
  160         EQUS "Read error"
  170 .ok
  180         BRK
  190         EQUS "Read sucessful"
  200         BRK
  210         BRK
  220 .block
  230         EQUB &FF      \ current drive
  240         EQUD page     \ start at PAGE
  250         EQUB &03      \ 3 parameters
  260         EQUB &57      \ read data and deleted data
  270         EQUB &01      \ logical track 1
  280         EQUB &00      \ start logical sector 0
  290         EQUB &2A      \ 10 sectors of 256 bytes
  300 .result
  310         EQUB &00      \ result byte
  320 ]
  330 NEXT
  340 PRINT''"Type: CALL &";~mcode;" to read 10 sectors"'
```

Osword &7F Read Data and Deleted Data 128 bytes

If you need to read just one sector of 128 bytes you can use the Osword &7F Read Data and Deleted Data 128 bytes command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &56 (read data and deleted data 128 bytes)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = result byte
```

Although using this command saves you the trouble of calculating the sector size/number of sectors parameter, I can not recommend using it when the multi-sector command is so much more versitile.

Osword &7F Read Data multi-sector

If you need to read data which is not marked as deleted data then you can use the Osword &7F Read Data multi-sector command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &53 (read data multi sector command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = sector size/number of sectors
Parameter block &0A       = result byte
```

This command can be used in exactly the same way as the Osword &7F Read Data and Deleted Data command but it cannot read deleted data. There is no need to use this command when the Osword &7F Read Data and Deleted Data command is more versitile.

Osword &7F Read Data 128 bytes

If you need to read just one sector of 128 bytes of data which is not marked as deleted data then you can use the Osword &7F Read Data 128 bytes command. This command uses the following parameter block.

```
Parameter block &00       = drive number (&00-&03 or &FF)
Parameter block &01 - &04 = buffer address (not used)
Parameter block &05       = &02 (2 command parameters)
Parameter block &06       = &52 (read data 128 bytes command)
Parameter block &07       = logical track number
Parameter block &08       = logical sector number
Parameter block &09       = result byte
```

Although using this command saves you the trouble of calculating the sector size/number of sectors parameter, I can not recommend using it when the multi-sector command is so much more versitile.

The Acorn DFS Osword commands - by - Gordon Horsington

Module 3. Formatting single density discs

```
      +-----------------------------------------------------------+
      | All the DFS modules in  this  series  use programs which |
      | experiment  with the format and contents of discs. These |
      | experiments may  have disasterous effects if you use any |
      | of the programs  on  discs  which store programs or data |
      | which you cannot afford to lose.  You  should  first try |
      | out  the  programs  using  discs  that  have either been |
      | duplicated or, better still, have not been used at all.  |
      +-----------------------------------------------------------+
```

Writing your own disc formatting program can be quite a useful exercise because it will give you the opportunity to optimise the speed at which data can be written to and read from a disc. You might think that all disc formatting programs are the same but, if you do, then you are quite wrong.

The time taken to write to and read from a disc is affected by the settings on the keyboard DIL switches and the logical sector offsets created during formatting. If you have a DIL switch block on your keyboard you might like to experiment with the settings to increase the performance of your disc drives.

Figure 1 shows the effect of switching links 3 and 4 on the keyboard DIL switch block, but do remember that some disc drive manuals specify the settings to be used and these should not be altered. If you do not have a switch block fitted on you keyboard the effect of using these switches can be simulated using Osbyte &FF. Osbyte &FF takes effect after a soft Break and remains active until a hard Break. All but the slowest disc drives can be operated with link 3 open (off) and link 4 closed (on) and many modern disc drives can be used with both links 3 and 4 closed (on). If the settings for links 3 and 4 have not been specified for your disc drive you should experiment and use the fastest reliable speed.

3	4	Step time	Settle time	Load time	Osbyte &FF	Speed
on	on	4	16	0	*FX 255,0,207	Fastest
on	off	6	16	0	*FX 255,0,223	Faster
off	on	6	50	32	*FX 255,0,239	Fast
off	off	24	20	64	*FX 255,0,255	Slow

Figure 1. The effect of keyboard switches 3 and 4 on disc access times
---

The Acorn DFS uses discs with either 40 or 80 tracks and with 10 sectors of 256 bytes per track. The physical and logical track numbers must be the same so that, for example, the ten ID fields on track &01 must all use logical track number &01. The 10 sectors on each track must use the logical sector numbers &00 to &09 but the logical and physical sector numbers do not have to be the same.

Logical sectors &00 and &01 on physical track &00 store the disc catalogue. The catalogue uses the structure shown in figure 2 and an empty catalogue must be created by the disc formatting program.

Sector &00 Track &00

```
&00 - &07  First 8 bytes of the 12 byte disc title.
&08 - &0E  First file name.
&0F        Directory of first file name.
&10 - &16  Second file name.
&17        Directory of second file name.
&18 - &FF  and so on for the 31 files.
```

Sector &01 Track &00

```
&00 - &03  Last 4 bytes of the 12 byte disc title.
&04        Disc cycles (BCD number 0-99).
&05        8 * (Number of catalogue entries).
&06        (bits 0 and 1) Most significant two bits of the number of
           sectors on the disc.
           (bits 4 and 5) The boot up option set using *OPT4,n.
&07        The least significant 8 bits of the (10 bit) number of sectors
           on the disc. The most significant bits are in bits 0 and 1 of
           byte &06.
&08 - &09  Load address of first file, least significant 16 bits.
&0A - &0B  Execution address of first file, least significant 16 bits.
&0C - &0D  Length of first file, least significant 16 bits.
&0E        (bits 0 and 1) Startsector of first file, most sig. 2 bits.
           (bits 2 and 3) Load address of first file, most sig. 2 bits.
           (bits 4 and 5) Length of first file, most sig. 2 bits.
           (bits 6 and 7) Execution address of first file, most sig. bits.
&0F        Start sector of first file, least significant 8 bits.
&10 - &FF  Load address, execution address, file length, and sector number
           for every other file on the disc (8 bytes per file). This is
           the information given by the *INFO call.
```

Figure 2. The structure of the DFS catalogue

Although there is a lot of information stored in a disc catalogue nearly all this information is written by the DFS when the disc is being used. An Acorn single density formatting program must fill the catalogue with null bytes (&00) with the exception of bytes &06 and &07 of sector &01. These bytes must store the number of sectors made available on the disc by the formatting program.

When a single density formatter is used with an eighty track disc drive it will create &320 sectors on a disc. Bits 0 and 1 of byte &06 on logical sector &01 must store the number &03 (%11), and byte &07 of logical sector &01 must store the number &20. When the program is used with a fourty track disc drive it will create &190 sectors. Bits 0 and 1 of byte &06 on logical sector &01 must then store the number &01 (%01), and byte &07 of logical sector &01 must store the number &90. Figure 3 shows a part of a sector dump made with a newly formatted eighty track disc. The only information stored in the empty catalogue is the number of available sectors on the disc.

```
Track: 00   Logical Sector: 00

       0  1  2  3  4  5  6  7

00     0  0  0  0  0  0  0  0  ........
08     0  0  0  0  0  0  0  0  ........
10     0  0  0  0  0  0  0  0  ........
18     0  0  0  0  0  0  0  0  ........
20     ...

Track: 00   Logical Sector: 01

       0  1  2  3  4  5  6  7

00     0  0  0  0  0  0  3 20  .......
08     0  0  0  0  0  0  0  0  ........
10     0  0  0  0  0  0  0  0  ........
18     0  0  0  0  0  0  0  0  ........
20     ...
```

Figure 3. Part of the catalogue of a newly formatted 80 track disc

The most efficient operation of an Acorn single density disc occurs if physical sector &00 does not store logical sector &00 on every track. The optimum distribution of logical sectors with respect to physical sectors for most disc drives is shown in figure 4. Figure 4 shows that, on track &00, the physical and logical sector numbers are the same. On track &01, physical sector &00 stores logical sector &07, physical sector &01 stores logical sector &08, and so on. This is known as a logical sector offset.

```
                         Physical sector numbers

                    | 00 01 02 03 04 05 06 07 08 09
                 ---+------------------------------
              T  00 | 00 01 02 03 04 05 06 07 08 09
              r  01 | 07 08 09 00 01 02 03 04 05 06 Logical
              a  02 | 04 05 06 07 08 09 00 01 02 03 sector
              c  03 | 01 02 03 04 05 06 07 08 09 00 numbers
              k  04 | 08 09 00 01 02 03 04 05 06 07
              s  05 | 05 06 ...
```

Figure 4. The logical sectors numbers for optimum speed

The logical sector numbers are offset because the disc drive head takes a predictable amount of time to step from one track to the next when writing to or reading from a disc.

Consider what will happen if you use the *LOAD command to read a file which is stored in 10 sectors starting with sector &02 on track &00. This hypothetical file will be stored on logical sectors &02 to &09 on track &00, and on logical sectors &00 and &01 on track &01. The sectors on track &00 will be loaded from sector &02 to sector &09 and then the head will step in to track &01. This step will take a predictable amount of time which is long enough for most disc drives to miss logical sector &00 if it is stored on physical sector &00. The disc would then have to make a complete revolution before sector &00 reappears. Using the offset illustrated in figure 4 would ensure that, for most disc drives, sector &00 on track &01 would become immeadiatly available as the head steps in from track &00 to &01. All the tracks in figure 4 use the same sector offset with respect to each other to give an optimum distribution of logical sectors.

Not all disc drives take the same amount of time to step the head from one track to another and for this reason the amount of offset used to produce an optimally formated disc will vary from one disc drive to another. The distribution of logical sectors shown in figure 4 uses an offset of 3 sectors. That is, sector &00 is offset 3 sectors with respect to sector &09 on the preceding track. A modern fast disc drive might only require an offset of 2 sectors so that the physical logical sector &00 on track &01 will be logical sector &08. An old disc drive liberated from the local junk shop might require an offset of 4 or even 5 sectors.

The program OFFSET can be used to experiment with the amount of offset given to the logical sector numbers. It can format both 40 and 80 track discs with any offset from 0 to 9 sectors.

To find the amount of offset needed with a particular disc drive you should create a set of 10 discs with the offset varying from 0 to 9 sectors. Each disc should be used to measure the time taken to store a very large file a large number of times within a program loop. It is a good idea to store the same large file 20 or 30 times and to use a stop watch rather than the computer to measure the time taken. If you start with an offset of 9 sectors and work down to zero offset you should find that the time decreases with each disc until, with one disc, there is an increase in the time taken to store the files. If, for example, the increase is with a disc using an offset of 2 sectors then an optimum offset of 3 sectors is needed for your disc drive. Most disc drives need an offset of 3 sectors.

```
   10 REM: OFFSET
   20 DIM mcode &400
   30 oswrch=&FFEE
   40 osnewl=&FFE7
   50 osword=&FFF1
   60 osbyte=&FFF4
   70 FORpass=0 TO 2 STEP 2
   80 P%=mcode
   90 [       OPT pass
  100         JSR osnewl
  110 .loop
  120         LDA &FF       \ poll escape flag
```

```
130         BPL noescape  \ bit 7 set if Escape pressed
140 .escape
150         LDA #&7E
160         JSR osbyte    \ acknowledge Escape
170         BRK
180         BRK
190         EQUS "Escape"
200         BRK
210 .noescape
220         LDA track     \ load physical track number
230         STA block+7   \ store physical track number
240         BEQ endoffset \ don't offset track zero
250         LDX #36       \ logical track index
260         LDY #38       \ logical sector index
270 .inloop
280         LDA track     \ load physical track number
290         STA table,X   \ store logical track number
300         LDA shear     \ load logical sector offset
310         BEQ zero      \ branch if no offset
320         STA temp      \ temporary store
330 .offsetloop
340         SEC
350         LDA table,Y   \ load logical sector number
360         SBC #1        \ subtract 1
370         BPL positive
380         LDA #9
390 .positive
400         STA table,Y   \ store logical sector number - 1
410         DEC temp      \ decrement sector offset
420         BNE offsetloop \ offset again
430 .zero
440         DEX
450         DEX
460         TXA
470         TAY           \ subtract 4 from Y register
480         DEX
490         DEX           \ subtract 4 from X register
500         BPL inloop    \ branch if less than 10 sectors
510 .endoffset
520         LDA #&7F
530         LDX #block MOD 256
540         LDY #block DIV 256
550         JSR osword    \ format track
560         LDA block+12  \ load result byte
570         BNE error     \ format OK if result = 0
580         JSR printtrack \ print track number
590         INC track     \ increment track number
600         LDA track     \ load track number
610         CMP finish    \ is that the last track?
620         BCC loop      \ branch if more tracks to format
630         LDA #&7F
640         LDX #catblock MOD 256
650         LDY #catblock DIV 256
660         JSR osword    \ store empty catalogue
670         LDA catblock+10 \ check result byte
680         BNE error     \ branch if not saved
690         RTS           \ return to BASIC
700 .error
710         BRK
720         BRK
730         EQUS "Format error"
740         BRK
750 .printtrack
760         LDA track     \ load track number
770         LSR A
780         LSR A
790         LSR A
800         LSR A         \ isolate MS nybble
810         JSR nybble    \ print MS nybble
820         LDA track     \ load track number
830         JSR nybble    \ print LS nybble
840         LDA #ASC(" ")
850         JSR oswrch    \ print space
860         JMP oswrch    \ print space
870 .nybble
880         AND #&0F
890         SED
```

```
 900         CLC
 910         ADC #&90
 920         ADC #&40
 930         CLD
 940         JMP oswrch    \ print nybble and return
 950 .block
 960         EQUB &00      \ drive number 0-3
 970         EQUD table    \ sector table
 980         EQUB &05      \ 5 parameters
 990         EQUB &63      \ format track
1000         EQUB &00      \ physical track number 0-79
1010         EQUB &15      \ gap 3
1020         EQUB &2A      \ 10 sectors of 256 bytes
1030         EQUB &00      \ gap 5
1040         EQUB &10      \ gap 1
1050         EQUB &00      \ result byte
1060 .table
1070         EQUD &01000000
1080         EQUD &01010000
1090         EQUD &01020000
1100         EQUD &01030000
1110         EQUD &01040000
1120         EQUD &01050000
1130         EQUD &01060000
1140         EQUD &01070000
1150         EQUD &01080000
1160         EQUD &01090000
1170 .catalogue
1180         OPT FNfill(262)
1190                       \ store 262 zeros
1200 .sectors
1210         EQUW &2003    \ &320 sectors (80 tracks)
1220         OPT FNfill(248)
1230                       \ store 248 zeros
1240 .catblock
1250         EQUB &00      \ drive number 0 - 3
1260         EQUD catalogue \ address of buffer
1270         EQUB &03      \ number of parameters
1280         EQUB &4B      \ write data multi-sector
1290         EQUB &00      \ logical track
1300         EQUB &00      \ start logical sector
1310         EQUB &22      \ 2 sectors of 256 bytes
1320         EQUB &00      \ result byte
1330 .track
1340         EQUB &00      \ physical track number
1350 .finish
1360         EQUB &00      \ number of tracks
1370 .shear
1380         EQUB &00      \ sector offset
1390 .temp
1400         EQUB &00      \ sector offset
1410 ]
1420 NEXT
1430 REPEAT
1440 INPUT"Drive number (0-3) "D%
1450 UNTIL D%>-1 AND D%<4
1460 ?block=D%
1470 ?catblock=D%
1480 REPEAT
1490 INPUT"Number of tracks (40/80) "T%
1500 UNTIL T%=40 OR T%=80
1510 ?finish=T%
1520 IF T%=40 THEN ?sectors=&01 : sectors?1=&90
1530 REPEAT
1540 INPUT"Logical sector offset (0-9) "L%
1550 UNTIL L%>-1 AND L%<10
1560 ?shear=L%
1570 INPUT"Ready to format? (Y/N) "yes$
1580 IF LEFT$(yes$,1)="Y" THEN CALL mcode
1590 INPUT'"Another disc? (Y/N) "yes$
1600 IF LEFT$(yes$,1)="Y" THEN RUN
1610 END
1620 DEF FNfill(size)
1630 FOR count = 1 TO size
1640 ?P%=0
1650 P%=P%+1
1660 NEXT
```

```
 1670 =pass
```

The logical sector offsets produced by the program OFFSET can be demonstrated by using the program IDSDUMP introduced in module 0. The formatted discs it produces can be verified using any DFS verification program including the program VERIFY, also intorduced in module 0.

The Acorn DFS Osword commands - by - Gordon Horsington

Module 4. Converting 40 track discs for 80 track drives

```
      +----------------------------------------------------------+
      | All the DFS modules in  this  series  use programs which |
      | experiment  with the format and contents of discs. These |
      | experiments may  have disasterous effects if you use any |
      | of the programs  on  discs  which store programs or data |
      | which you cannot afford to lose.  You  should  first try |
      | out  the  programs  using  discs  that  have either been |
      | duplicated or, better still, have not been used at all.  |
      +----------------------------------------------------------+
```

In this module I will examine the problem of modifying forty track discs so that they work properly on eighty track disc drives.

There are a number of possible solutions to the problem of using forty track discs on an eighty track disc drive. If you have a 40/80 track switchable disc drive it is possible to use *BACKUP 0 0 and switch the track density selector from forty to eighty before inserting an eighty track disc and then switch back to forty for the forty track disc. If you use dual switchable drives this can be a less error prone method because you would not have to remember to switch the drives between reading and writing. One problem with this method is that, although it will produce a disc which can be used on an eighty track disc drive, it will only use forty of the eighty tracks. This could be overcome by using the *COPY command and switching the track density selector appropriately. A philosophical problem with using either of these methods to make the conversion is that the conversion itself is pointless if you have switchable disc drives.

The real problem exists for disc users who have a forty track disc and only one eighty track disc drive. In these circumstances it is necessary to modify the disc itself so that it becomes an eighty track disc. In this module I will demonstrate how to use the Osword &7F commands to modify a forty track disc on an eighty track disc drive so that the forty track disc becomes an eighty track disc.

Forty track discs have a track density exactly one half of that used by eighty track discs. Track &00 of both forty and eighty track discs share the same physical position as the outer track on the disc. Track &01 on a forty track disc is in the same physical position as track &02 on an eighty track disc, track &02 is in the same position as track &04, and so on. The relative position of the physical tracks on forty and eighty track discs is shown in figure 1.

```
40T> 00    01 ... 09    0A    0B ... 13    14    15 ... 25    26    27
80T> 00 01 02 ... 12 13 14 15 16 ... 26 27 28 29 2A ... 4A 4B 4C 4D 4E 4F
```

Figure 1. The relative position of tracks on 40 and 80 track discs
----

All the numbers in figure 1 are in hexadecimal and I am using the term track density to refer to the physical proximity of the tracks (which are closer together on 80 track discs). Because the track density of an eighty track disc is twice that of a forty track disc, an eighty track disc is sometimes refered to as double density even if it uses the single density format. I will not refer to 80 track discs as double density because this can be confused with the double density ADFS, which can use 80 track discs in a double density format.

If you look at figure 1 you should be able to work out what needs to be done to make forty track discs work on an eighty track disc drive. Because track &00 is in the same physical position with both forty and eighty track discs, any data on track &00 can be accessed with either type of disc drive. All the other tracks on a forty track disc are in the wrong physical position to be read on an eighty track disc drive. Assuming that you have an Acorn DFS forty track disc in an eighty track disc drive, you will need to copy physical track &02 (logical track &01) onto physical track &01, copy physical track &04 (logical track &02) onto physical track &02 and so on until physical track &4E (logical track &27) is copied onto physical track &27. This copying is summarised in figure 2.

```
            Read physical track -> Format and write physical track

                            &00 -> &00
                            &02 -> &01
                            &04 -> &02
                            &06 -> &03
                                 .
                                 .
                                 .
                            &4B -> &25
                            &4C -> &26
                            &4E -> &27

           Figure 2. The required copying of physical tracks
           -------------------------------------------------
```

You should note that figure 2 refers to the physical tracks and not the logical tracks. This is important because, with Acorn DFS forty track discs in an eighty track disc drive, physical track &02 will contain logical track &01, physical track &04 will contain logical track &02, and so on. It will be necessary to re-format each physical track after reading and before writing the data back onto the disc. This is because the odd numbered tracks will be unformatted and the even numbered tracks will have the wrong logical track numbers in their ID fields.

When all forty tracks have been copied you can then go on to format the tracks from &28 to &4F to make the disc an eighty track disc. In order that the DFS can access all eighty tracks it is necessary to modify the catalogue to indicate that 800 (&320) sectors are available. The number of sectors is stored in bytes &06 (MSB) and &07 (LSB) of track &00, sector &00. Note that it is not LSB and MSB as you might expect.

The following algorithm can be used to implement this idea.

1) Start with current track number = &00

2) Seek current track * 2 (ie. for logical track &01 seek &02, and so on)

3) Write the current track number into the track register (number &12). The head is now positioned above the appropriate track for reading the data and the track register also contains the logical track number.

4) Read the entire track into a buffer.

5) Write the current track number * 2 into the track register.

6) Format the physical track indicated by the current track number.

7) Write the contents of the buffer onto the newly formatted track.

8) Increment the current track number. If the current track number is less than &28 go back to 2) to copy the next track.

9) All forty tracks have been converted. Now format tracks &28 to &4F to create an eighty track disc.

10) Read the contents of track &00, sector &01 into a buffer.

11) OR the contents of buffer+&06 with the number &03, and store the number &20 in buffer+&07 to indicate that &320 (800) sectors are available on the disc.

12) Write the contents of the buffer back onto track &00, sector &01.

This algorithm has been implemented in the program CONVERT. You can use the program CONVERT with an eighty track drive &00 to convert an Acorn formatted 40 track single density disc into an eighty track single density disc. The program will not work with copy-protected discs and you should only use it if you have a duplicate of the disc you intend to convert.

If you use CONVERT and press the Escape key before it has finished the conversion you will probably destroy some or all the data on the disc. Do not use the program with a forty track disc drive. You have been warned to be careful with all the programs used to illustrate this series. This program can easily destroy all the data on your disc if you fail to use it with care.

If you want to modify the program to make it more idiot proof you could delete line 130 to take out the routine which polls the Escape flag. This would make the program safer for ham-fisted or inexperienced users because pressing Escape would not halt the program before it had finished.

Whatever modifications you make to this or any other of the programs used to ilustrate the DFS modules of the series, don't ignore the warning about the potentially disasterous effects these programs can have on your discs.

```
   10 REM: CONVERT
   20 osnewl=&FFE7
   30 oswrch=&FFEE
   40 osword=&FFF1
   50 osbyte=&FFF4
   60 DIM mcode &500
   70 DIM buffer &1000
   80 FOR pass=0 TO 2 STEP 2
   90 P%=mcode
  100 [       OPT pass
  110         JSR osnewl
  120 .mainloop
  130         JSR escape    \ check escape flag
  140         JSR seektwo   \ seek track * 2
  150         LDA track     \ load logical track number
  160         JSR register  \ write track register
  170         JSR read      \ read logical track
  180         LDA track     \ load logical track number
  190         ASL A         \ *2 = physical track
  200         JSR register  \ write track register
  210         JSR format    \ format physical track
  220         JSR write     \ write data onto disc
  230         JSR printbyte \ print track number
  240         INC track     \ get ready for next track
  250         LDA track     \ load logical track number
  260         CMP #40       \ is it track 40?
  270         BNE mainloop  \ if not read next track
  280 .formloop
  290         JSR escape    \ check escape flag
  300         JSR format    \ format tracks 40 - 79
  310         JSR printbyte \ print track number
  320         INC track     \ increment track number
  330         LDA track     \ load track number
  340         CMP #80       \ is it 80?
  350         BNE formloop  \ if not format next track
  360         LDA #0        \ go back to track 0
  370         STA track
  380         STA copyblock+7
  390         LDA #1        \ sector 1
  400         STA copyblock+8
  410         LDA #&21      \ 1 sector of 256 bytes
  420         STA copyblock+9
  430         JSR read      \ read track 0 sector 1
  440         LDA #&03      \ 800 sectors DIV 256
  450         ORA buffer+6  \ keep old *OPT4,n option
  460         STA buffer+6  \ MSB number of sectors
  470         LDA #&20      \ 800 sectors MOD 256
  480         STA buffer+7  \ LSB number of sectors
  490         JSR write     \ store track 0 sector 1
  500         JSR osnewl
  510         RTS           \ return to BASIC
  520 .escape
  530         LDA &FF       \ escape flag
  540         BMI pressed   \ bit 7 set if pressed
  550         RTS
  560 .pressed
  570         LDA #&7E
  580         JSR osbyte    \ acknowledge Escape
  590         BRK
  600         BRK
  610         EQUS "Escape"
```

```
 620         BRK
 630 .seektwo
 640         LDA track     \ source track number
 650         ASL A         \ *2
 660         STA seekblock+7 \ physical track number
 670         LDA #&7F
 680         LDX #seekblock MOD 256
 690         LDY #seekblock DIV 256
 700         JSR osword
 710         LDA seekblock+8 \ result
 720         BNE seekerror \ = 0 if OK
 730         RTS
 740 .seekerror
 750         BRK
 760         BRK
 770         EQUS "Seek error"
 780         BRK
 790 .format
 800         LDA track     \ source track number
 810         STA formblock+7 \ store physical track
 820         LDX #36
 830 .tableloop
 840         STA table,X   \ store logical track number in ID table
 850         DEX
 860         DEX
 870         DEX
 880         DEX
 890         BPL tableloop
 900         LDA #&7F
 910         LDX #formblock MOD 256
 920         LDY #formblock DIV 256
 930         JSR osword
 940         LDA formblock+12 \ result
 950         BNE formerror \ = 0 if OK
 960         RTS
 970 .formerror
 980         BRK
 990         BRK
1000         EQUS "Format error"
1010         BRK
1020 .register
1030         STA regblock+8 \ value to put in register
1040         LDA #&7F
1050         LDX #regblock MOD 256
1060         LDY #regblock DIV 256
1070         JSR osword
1080         LDA regblock+9 \ result
1090         BNE regerror  \ = 0 if OK
1100         RTS
1110 .regerror
1120         BRK
1130         BRK
1140         EQUS "Special register error"
1150         BRK
1160 .read
1170         LDA track     \ source track number
1180         STA copyblock+7 \ logical track number
1190         LDA #&53      \ read data multi-sector
1200         STA copyblock+6
1210         LDA #&7F
1220         LDX #copyblock MOD 256
1230         LDY #copyblock DIV 256
1240         JSR osword
1250         LDA copyblock+10
1260         BNE readerror
1270         RTS
1280 .readerror
1290         BRK
1300         BRK
1310         EQUS "Read error"
1320         BRK
1330 .write
1340         LDA #&4B      \ write data multi-sector
1350         STA copyblock+6
1360         LDA #&7F
1370         LDX #copyblock MOD 256
1380         LDY #copyblock DIV 256
```

```
1390         JSR osword
1400         LDA copyblock+10 \ result
1410         BNE writeerror \ = 0 if OK
1420         RTS
1430 .writeerror
1440         BRK
1450         BRK
1460         EQUS "Write error"
1470         BRK
1480 .printbyte
1490         LDA track     \ print source track number
1500         PHA
1510         LSR A
1520         LSR A
1530         LSR A
1540         LSR A
1550         JSR nybble    \ print MS nybble
1560         PLA
1570         JSR nybble    \ print LS nybble
1580         LDA #ASC(" ")
1590         JSR oswrch    \ print space
1600         JMP oswrch    \ print space
1610 .nybble
1620         AND #&0F
1630         SED
1640         CLC
1650         ADC #&90
1660         ADC #&40
1670         CLD
1680         JMP oswrch    \ print nybble and return
1690 .seekblock
1700         EQUB &00      \ drive 0
1710         EQUD &00      \ does not matter
1720         EQUB &01      \ 1 parameter
1730         EQUB &69      \ seek command
1740         EQUB &00      \ physical track
1750         EQUB &00      \ result byte
1760 .regblock
1770         EQUB &00      \ drive 0
1780         EQUD &00      \ does not matter
1790         EQUD &00127A02 \ write special register
1800         EQUB &00      \ result byte
1810 .copyblock
1820         EQUB &00      \ drive 0
1830         EQUD buffer   \ address of buffer
1840         EQUB &03      \ 3 parameters
1850         EQUB &57      \ read data multi-sector
1860         EQUB &00      \ logical track number
1870         EQUB &00      \ logical sector number
1880         EQUB &2A      \ 10 sectors of 256 bytes
1890         EQUB &00      \ result byte
1900 .formblock
1910         EQUB &00      \ drive 0
1920         EQUD table    \ address of sector table
1930         EQUB &05      \ 5 parameters
1940         EQUB &63      \ format command
1950         EQUB &00      \ physical track number
1960         EQUB &15      \ gap 3 size
1970         EQUB &2A      \ 10 sectors of 256 bytes
1980         EQUB &00      \ gap 5 size
1990         EQUB &10      \ gap 1 size
2000         EQUB &00      \ result byte
2010 .table
2020         EQUD &01000000
2030         EQUD &01010000
2040         EQUD &01020000
2050         EQUD &01030000
2060         EQUD &01040000
2070         EQUD &01050000
2080         EQUD &01060000
2090         EQUD &01070000
2100         EQUD &01080000
2110         EQUD &01090000
2120 .track
2130         EQUB &00      \ logical track number
2140 ]
2150 NEXT
```

```
2160 PRINT'"Place 40 track disc in 80 track drive 0"
2170 PRINT"Press Spacebar to convert to 80 tracks"
2180 REPEAT
2190 UNTIL GET = 32
2200 CALL mcode
```

The Acorn DFS Osword commands - by - Gordon Horsington

Module 5. Creating discs compatible with both 40 and 80 track drives
----

```
      +-----------------------------------------------------------+
      | All the DFS modules in  this  series  use programs which |
      | experiment  with the format and contents of discs. These |
      | experiments may  have disasterous effects if you use any |
      | of the programs  on  discs  which store programs or data |
      | which you cannot afford to lose.  You  should  first try |
      | out  the  programs  using  discs  that  have either been |
      | duplicated or, better still, have not been used at all.  |
      +-----------------------------------------------------------+
```

In this module I will examine the problem of modifying eighty track discs so that they work properly on both forty and eighty track disc drives. I will demonstrate how this can be done on an unswitched eighty track disc drive using an Osword &7F based program and a disc formatting program. I will then describe how the same dual format disc can be created with only a disc formatting program if a 40/80 track switchable disc drive is available.

Forty track discs have a track density exactly one half of that used by eighty track discs. Track &00 of both forty and eighty track discs share the same physical position as the outer track on the disc. Track &01 on a forty track disc is in the same physical position as track &02 on an eighty track disc, track &02 is in the same position as track &04, and so on. The relative position of the physical tracks on forty and eighty track discs is shown in figure 1.

```
40T> 00    01 ... 09    0A    0B ... 13    14    15 ... 25    26    27
80T> 00 01 02 ... 12 13 14 15 16 ... 26 27 28 29 2A ... 4A 4B 4C 4D 4E 4F
```

   Figure 1. The relative position of tracks on 40 and 80 track discs
   ----

All the numbers in figures 1 and 2 are in hexadecimal and I have used the term track density refers to the physical proximity of the tracks (which are closer together on 80 track discs). Because the track density of an eighty track disc is twice that of a forty track disc, an eighty track disc is sometimes refered to as double density even if it uses a single density format. I will not refer to 80 track discs as double density because this can be confused with the double density ADFS, which can use 80 track discs in a double density format.

Figure 2 illustrates one method of formatting a single density disc so that the same disc can be used with either a forty or an eighty track disc drive.

```
00 01 02 ... 12 13 14 15 16 ... 26 27 14    15    16 ... 25    26    27
|  \-------------/ \----------------/ \-------------------------------/
|  &01-&13 unused        &14-&27                   &14-&27
| 80 track density  80 track density          40 track density
|
Track &00, common to both track densities
```

            Figure 2. The format for 40/80 track discs

The disc can be divided into four regions.

1) Track &00, which is common to both forty and eighty track densities.
2) Tracks &01 to &13 (1 to 19 decimal) in eighty track density are unused.
3) Tracks &14 to &27 (20 to 39 decimal) used in eighty track density.
4) Tracks &14 to &27 (20 to 39 decimal) used in forty track density.

The data stored on this type of dual format disc are stored on track &00 and tracks &14 to &27. The data on tracks &14 to &27 are stored twice, once in forty track density and once in eighty track density. Sectors &00

and &01 of track &00 are used to store the catalogue for the disc. The remaining sectors on track &00 give a total of 2k available for data. Tracks &14 to &27 have 50k available for data. This dual format disc makes 52k available for programs and data.

This type of disc can be created on an eighty track disc drive by using a forty track disc formatting program to format the first forty tracks on a disc in an eighty track disc drive. You can use the program OFFSET introduced in module 3 if you want to optimise the logical sector offset. Store a 2k (&800 bytes) dummy !BOOT file to fill track &00 and a large (47.5k), locked dummy file to fill tracks &01 to &13. These files can be created with the following commands:

```
*SAVE :0.$.!BOOT 1900+400
*SAVE :0.D.DUMMY 1900+BE00
*ACCESS :0.D.DUMMY L
```

Up to 50k of data can then be stored on tracks &14 to &27 using the DFS star commands. Do not use the filenames $.!BOOT or D.DUMMY. When all the data, except the real !BOOT file, are stored on the disc then the dummy !BOOT file can be deleted and the real !BOOT file (which must not be longer than 2k) can be stored. Do not delete the dummy file D.DUMMY because this file is making sure that tracks &01 to &13 inclusive remain unused (see figure 2).

When all the programs and data have been copied onto the disc then logical tracks &14 to &27 should be copied from the eighty track density physical tracks to the forty track density physical tracks. Physical track &14 (eighty track density) will be copied onto physical track &28 (eighty track density), physical track &15 will be copied onto physical track &2A and so on as shown in figure 3.

```
            Read physical track -> Format and write physical track

                            &14 -> &28
                            &15 -> &2A
                            &16 -> &2C
                            &17 -> &2E
                                .
                                .
                                .
                            &25 -> &4B
                            &26 -> &4C
                            &27 -> &4E
```

Figure 3. The required copying of physical tracks

This copying will produce the format shown in figure 2. The following algorithm can be used to implement this idea using a disc which has had the first forty tracks formatted in an eighty track disc drive. It is important to format only the first forty tracks so that the DFS recognises the disc as a forty track disc even though it is used in an eighty track disc drive. Use *FORMAT 40 0 (or whatever is appropriate with your system). Do not use *FORMAT 80 0 and press Escape after forty tracks have been formatted because, if you do, the DFS will still recognise the disc as an eighty track disc even though only the first forty tracks have been formatted.

1) Start with physical track number &14 (decimal 20).

2) Read the sector IDs on the current physical track. These will be used to create the sector data for formatting the forty track copy.

3) Read all the data on the current track into a buffer.

4) Seek the current physical track * 2. This will be where the forty track copy will be written.

5) Write the physical track number (&14-&27) into the track register (number &12). This will allow the eighty track disc drive to write the data onto a track in the position it would use on a forty track disc

drive.

6) Format the track found in step 4) using the sector data from step 2).

7) Write the contents of the buffer onto the newly formatted track.

8) Write the physical track number * 2 (&28-&4E) into the track register (number &12). This takes the disc controller back to the eighty track status.

9) Increment the track number. If it is less than &28 (decimal 40) then go back to 2) to duplicate the next track.

This algorithm has been implemented in the program DUALDFS. You must use the program DUALDFS with an eighty track disc drive (drive &00) to convert a disc formatted as described above into a 40/80 track disc. The program will not work with copy-protected discs and you should only use it after you have made a backup copy of all the files on the disc you intend to convert.

If you use DUALDFS and press the Escape key before it has finished the conversion you will only be able to use the disc on an eighty track disc drive. Do not attempt to use the program to make the conversion using a forty track disc drive. You have been warned to be careful with all the programs used to illustrate this series. Whatever modifications you make to this or any other of the programs used to ilustrate the DFS modules of the series, don't ignore the warning about the potentially disasterous effects these programs can have on your discs.

After using DUALDFS to create a dual format disc you must not use the command *COMPACT with the disc. It is a good idea stick a write-protect tab on all dual format discs.

```
   10 REM: DUALDFS
   20 osnewl=&FFE7
   30 oswrch=&FFEE
   40 osword=&FFF1
   50 osbyte=&FFF4
   60 DIM table &50
   70 DIM mcode &500
   80 DIM buffer &1000
   90 FOR pass=0 TO 2 STEP 2
  100 P%=mcode
  110 [       OPT pass
  120         JSR osnewl
  130 .mainloop
  140         JSR escape    \ check escape flag
  150         JSR sectorids \ read all sector ids
  160         JSR read      \ read all sectors
  170         JSR seektwo   \ seek source track * 2
  180         LDA track     \ source track number
  190         JSR register  \ write track register
  200         JSR format    \ format 2 * source track
  210         JSR write     \ write all sectors
  220         LDA track     \ load source track
  230         ASL A         \ *2 = physical track number
  240         JSR register  \ write track register
  250         JSR printbyte \ print track number
  260         INC track     \ increment source track number
  270         LDA track     \ load source track number
  280         CMP #40       \ all done?
  290         BNE mainloop  \ if not read next track
  300         JSR osnewl
  310         RTS           \ return to BASIC
  320 .escape
  330         LDA &FF       \ escape flag
  340         BMI pressed   \ bit 7 set if pressed
  350         RTS
  360 .pressed
  370         LDA #&7E
  380         JSR osbyte    \ acknowledge Escape
  390         BRK
  400         BRK
  410         EQUS "Escape"
  420         BRK
```

```
 430 .seektwo
 440         LDA track     \ source track number
 450         ASL A         \ *2
 460         STA seekblock+7 \ physical track number
 470         LDA #&7F
 480         LDX #seekblock MOD 256
 490         LDY #seekblock DIV 256
 500         JSR osword
 510         LDA seekblock+8 \ result
 520         BNE seekerror \ = 0 if OK
 530         RTS
 540 .seekerror
 550         BRK
 560         BRK
 570         EQUS "Seek error"
 580         BRK
 590 .format
 600         LDA track     \ source track number
 610         STA formblock+7 \ store physical track
 620         JSR register  \ write track register
 630         LDA #&7F
 640         LDX #formblock MOD 256
 650         LDY #formblock DIV 256
 660         JSR osword
 670         LDA formblock+12 \ result
 680         BNE formerror \ = 0 if OK
 690         RTS
 700 .formerror
 710         BRK
 720         BRK
 730         EQUS "Format error"
 740         BRK
 750 .register
 760         STA regblock+8 \ value to put in register
 770         LDA #&7F
 780         LDX #regblock MOD 256
 790         LDY #regblock DIV 256
 800         JSR osword
 810         LDA regblock+9 \ result
 820         BNE regerror  \ = 0 if OK
 830         RTS
 840 .regerror
 850         BRK
 860         BRK
 870         EQUS "Special register error"
 880         BRK
 890 .sectorids
 900         LDA track     \ source track number
 910         STA idsblock+7 \ store physical track
 920         LDA #&7F
 930         LDX #idsblock MOD 256
 940         LDY #idsblock DIV 256
 950         JSR osword
 960         LDA idsblock+10 \ result
 970         BNE idserror  \ = 0 if OK
 980         RTS
 990 .idserror
1000         BRK
1010         BRK
1020         EQUS "Sector ID Error"
1030         BRK
1040 .read
1050         LDA track     \ source track number
1060         STA copyblock+7 \ logical track number
1070         LDA #&53      \ read data multi-sector
1080         STA copyblock+6
1090         LDA #&7F
1100         LDX #copyblock MOD 256
1110         LDY #copyblock DIV 256
1120         JSR osword
1130         LDA copyblock+10
1140         BNE readerror
1150         RTS
1160 .readerror
1170         BRK
1180         BRK
1190         EQUS "Read error"
```

```
1200          BRK
1210 .write
1220          LDA #&4B       \ write data multi-sector
1230          STA copyblock+6
1240          LDA #&7F
1250          LDX #copyblock MOD 256
1260          LDY #copyblock DIV 256
1270          JSR osword
1280          LDA copyblock+10 \ result
1290          BNE writeerror \ = 0 if OK
1300          RTS
1310 .writeerror
1320          BRK
1330          BRK
1340          EQUS "Write error"
1350          BRK
1360 .printbyte
1370          LDA track      \ print source track number
1380          PHA
1390          LSR A
1400          LSR A
1410          LSR A
1420          LSR A
1430          JSR nybble     \ print MS nybble
1440          PLA
1450          JSR nybble     \ print LS nybble
1460          LDA #ASC(" ")
1470          JSR oswrch     \ print space
1480          JMP oswrch     \ print space
1490 .nybble
1500          AND #&0F
1510          SED
1520          CLC
1530          ADC #&90
1540          ADC #&40
1550          CLD
1560          JMP oswrch     \ print nybble and return
1570 .seekblock
1580          EQUB &00       \ drive 0
1590          EQUD &00       \ does not matter
1600          EQUB &01       \ 1 parameter
1601          EQUB &69       \ seek command
1602          EQUB &00       \ physical track number
1603          EQUB &00       \ result
1610 .regblock
1620          EQUB &00       \ drive 0
1630          EQUD &00       \ does not matter
1640          EQUB &02       \ 2 parameters
1641          EQUB &7A       \ write special register
1642          EQUB &12       \ track register, drive 0/2
1643          EQUB &00       \ value to be put in register
1650          EQUB &00       \ result
1660 .idsblock
1670          EQUB &00       \ drive 0
1680          EQUD table     \ address of buffer
1690          EQUB &03       \ 3 parameters
1691          EQUB &5B       \ read sector IDs command
1692          EQUB &00       \ physical track number
1693          EQUB &00
1700          EQUB &0A       \ number of IDs
1701          EQUB &00       \ result
1710 .copyblock
1720          EQUB &00       \ drive 0
1730          EQUD buffer    \ address of buffer
1740          EQUB &03       \ 3 parameters
1741          EQUB &57       \ read data multi-sector
1742          EQUB &00       \ logical track number
1743          EQUB &00       \ start logical sector number
1750          EQUB &2A       \ 10 sectors of 256 bytes
1751          EQUB &00       \ result
1760 .formblock
1770          EQUB &00       \ drive 0
1780          EQUD table     \ address of sector table
1790          EQUB &05       \ 5 parameters
1791          EQUB &63       \ format command
1792          EQUB &00       \ physical track number
1793          EQUB &15       \ gap 3 size
```

```
 1800          EQUB &2A      \ 10 sectors of 256 bytes
 1801          EQUB &00      \ gap 5 size
 1802          EQUB &10      \ gap 1 size
 1803          EQUB &00      \ result
 1810 .track
 1820          EQUB 20       \ use tracks 20-39
 1830 ]
 1840 NEXT
 1850 PRINT'"Place 40 track formatted 80 track disc"
 1860 PRINT"in drive 0, and press Spacebar"
 1870 REPEAT
 1880 UNTIL GET = 32
 1890 CALL mcode
```

If you have a switched 40/80 track disc drive it is quite easy to produce dual format discs without using a conversion program such as DUALDFS. To produce a dual format disc you need to use a forty track formatter and to be very careful about the order in which files are saved on the dual format disc. The following algorith will produce dual formatted discs.

1) Switch the disc drive to 40 track mode.

2) Format a disc using a forty track formatter.

3) Switch the disc drive to 80 track mode.

4) Format the same disc again using the same forty track formatter.

5) Fill track &00 with a dummy !BOOT file using

   *SAVE !BOOT 1900+800

6) Fill tracks &01 to &13 with a locked dummy file. use the commands:

   *SAVE D.DUMMY 1900+BE00
   *ACESS D.DUMMY L

7) Copy up to 50k of programs onto the disc. Don't use the filenames $.!BOOT or D.DUMMY.

8) Switch the disc drive to 40 track mode.

9) Copy the same files copied in step 7) onto the disc in exactly the same order. It is important that the order should be exactly the same.

10) Delete the dummy !BOOT file and store the real !BOOT file on the disc. The !BOOT file must not be longer than 2k.

This method will produce exactly the same dual format disc as that produced by the program DUALDFS but it does require the use of a switched disc drive and a great deal of care in storing the files in the same order on both formats.

The Acorn DFS Osword commands - by - Gordon Horsington

Module 6. Creating copy-protected single density discs

```
+-----------------------------------------------------------+
| All the DFS modules in  this  series  use programs which |
| experiment  with the format and contents of discs. These |
| experiments may  have disasterous effects if you use any |
| of the programs  on  discs  which store programs or data |
| which you cannot afford to lose.  You  should  first try |
| out  the  programs  using  discs  that  have either been |
| duplicated or, better still, have not been used at all.  |
+-----------------------------------------------------------+
```

In this module I will describe in detail one method of producing copy-protected single density discs. This type of disc is designed to prevent other people seeing how your programs work and to stop them duplicating the disc. Meeting the first objective is relatively easy. Meeting the second objective is virtually impossible. It is easy to prevent amateur piracy using duplicating programs but you will never be able to stop a determined hacker undoing all your efforts to prevent duplication. I have not yet found a commercially produced copy-protected disc for the BBC range of computers which can not have all the protection removed so that the disc can be duplicated with the *BACKUP command.

If you choose to copy-protect your discs you will at least indicate to the users of your software that you are unwilling to have the disc duplicated. To stand the best chance of defeating a hacker you should use the method described in this module as a starting point for developing your own ideas about protection. Remember that everyone reading this module will know how to undo the protection by just reversing the steps in the procedure. You should at least aim to produce a disc which cannot be duplicated by either of the two disc duplicators described in the next module. This will not be easy but all the information you need to do it has been or will be presented in this series. You might like to consider using Bad Tracks, Unusual logical sector numbers and logical track numbers in unexpected combinations, encrypted machine code programs and multiple files which use unusual methods of writing to and reading from the disc. What you do is up to you but you must try to think like a hacker if you want to prevent your software being hacked.

The method used to introduce copy-protection will use a single density disc formatted with 5 sectors per track instead of the usual 10 sectors per track. The data will be stored on the disc using the Write Deleted Data command as an extra protection.

A disc cannot be formatted with 5 sectors per track using the DFS formatter and so the first step in this procedure will be to write a special disc formatting program. The program SECTOR5 can be used to produce an object code file which, in turn, can be used to produce a single density disc with a standard track &00 and the rest of the disc formatted with 5 sectors on every track. Each of the non-standard tracks is capable of storing 2.5k of data in 5 sectors, ie. 512 bytes per sector. The object code file produced by SECTOR5 is exactly 1k long and can be stored in two of the 512 byte sectors. I will use this object code file as an example for producing the required copy-protected disc.

Run the program SECTOR5 and, when prompted, give a suitable filename for the object code file. In the rest of this module I will refer to the object code file produced by SECTOR5 as FORM5.

```
   10 REM: SECTOR5
   20 DIM mcode &500
   30 zeropage=&70
   40 oswrch=&FFEE
   50 osword=&FFF1
   60 osbyte=&FFF4
   70 osnewl=&FFE7
   80 oswrch=&FFEE
   90 osrdch=&FFE0
  100 osasci=&FFE3
  110 FORpass=4 TO 6 STEP 2
  120 O%=mcode
```

The Acorn DFS Osword commands - by - Gordon Horsington

Module 7. Duplicating copy-protected single density discs

```
+-----------------------------------------------------------+
| All the DFS modules in  this  series  use programs which |
| experiment  with the format and contents of discs. These |
| experiments may  have disasterous effects if you use any |
| of the programs  on  discs  which store programs or data |
| which you cannot afford to lose.  You  should  first try |
| out  the  programs  using  discs  that  have either been |
| duplicated or, better still, have not been used at all.  |
+-----------------------------------------------------------+
```

This module deals with duplicating copy-protected single density discs. As with the previous module, which dealt with producing this type of disc, you should use the information in this module as a starting point for your own program designs. The two disc duplication programs used to illustrate this module will copy most, but not all, single density discs. They have been designed simply to illustrate the techniques used to achieve this objective and, for that reason, they are not the fastest disc duplication programs available. The program COPYDFS is quite slow because it reads and writes a track at a time, but the program COPYALL is even slower because it reads and writes a sector at a time. Both programs require either 40 or 80 track dual disc drives because they copy from drive 0 to drive 1.

One of the many techniques used to copy-protect single density discs is to include unformatted tracks on the disc. The programs which use this type of disc then look at a particular physical track and, if they find that the track has been formatted, they reject the disc as an illegal copy. If formatted discs are to be used for duplicating protected software it may be necessary to be able to de-format some tracks on the disc. This means that when an attempt is made to read from or write to the disc a 'Sector not found' error should be produced.

This error will be generated if an attempt is made to read from or write to a track formatted with one sector of 2048 bytes. The program DEFORM can be used to demonstrate this idea. DEFORM must be used with great care because it removes track &00 from the disc in the current drive. Again, you have been warned!

```
   10 REM: DEFORM
   20 osword=&FFF1
   30 DIM mcode &100
   40 FORpass=0TO3STEP3
   50 P%=mcode
   60 [       OPT pass
   70         LDA #&7F
   80         LDX #block MOD 256
   90         LDY #block DIV 256
  100         JSR osword
  110         LDA result
  120         BNE error
  130         RTS
  140 .error
  150         BRK
  160         BRK
  170         EQUS "De-format error"
  180         BRK
  190 .block
  200         EQUB &FF      \ current drive
  210         EQUD buffer   \ sector table
  220         EQUB &05      \ 5 parameters
  230         EQUB &63      \ format command
  240         EQUB &00      \ physical track
  250         EQUB &00      \ gap 3
  260         EQUB &C1      \ sectors/size
  270         EQUB &00      \ gap 5
  280         EQUB &10      \ gap 1
  290 .result
  300         EQUB &00      \ result byte
  310 .buffer
  320         EQUD &04000000
```

```
330 ]
340 NEXT
350 CALL mcode
```

The de-formatting technique illustrated in the program DEFORM is used in both COPYDFS and COPYALL to ensure that any unformatted tracks on the source disc are unformatted on the destination disc even if the destination disc has been previously formatted.

Both the copy programs use a similar algorithm to duplicate copy-protected discs. These programs have been written to make them as easy to understand as possible. They are both well structured and well commented and you should make every effort to understand how they work. It is only when you fully understand how disc duplicators work that you can design a disc format which will defeat disc duplication programs.

The program COPYDFS uses the following algorithm to duplicate each track:

1) Seek the physical track on the source disc.

2) Read one sector ID from the physical track. If a 'Sector not Found' error is generated the track has not been formatted and the destination track should be de-formatted.

3) If the source track has been formatted, extract the number of sectors on the track from the data read in step 2) and read all the sector IDs on the track.

4) Format the destination disc using the sector ID data from step 3).

5) Read every sector on the source track using the Read Data and Deleted Data command and the sector ID data from step 3). Check for deleted data on the track.

6) If deleted data is used on the source track then use the Write Deleted Data command to write the data onto the destination track, otherwise use the Write Data command to write the data onto the destination track.

This simple algorithm will, somewhat surprisingly, duplicate many commercially protected discs but it is reletively easy to design a disc format which cannot be copied using this method. You might like to consider what would happen if, for example, you use a mixture of deleted and normal sectors on one track and use a !BOOT program which uses the appropriate command to read individual sectors rather than simply use the Read Data and Deleted Data command for all sectors. If you design a !BOOT program which does this on a copy-protected disc, then that disc could not be copied successfully using this algorithm. If you intend to take a serious interest in copy-protection then your first task should be to design a disc format which can not be copied by COPYDFS.

```
 10 REM: COPYDFS
 20 osnewl=&FFE7
 30 oswrch=&FFEE
 40 osword=&FFF1
 50 osbyte=&FFF4
 60 DIM table &50
 70 DIM mcode &500
 80 DIM buffer &1000
 90 FOR pass=0 TO 2 STEP 2
100 P%=mcode
110 [       OPT pass
120         JSR osnewl
130 .mainloop
140         JSR escape    \ check escape flag
150         JSR seek      \ seek physical tracks 0 - 40
160         JSR firstsector \ read sector id first sector
170         BNE notformatted \ if error then track not formatted
180         JSR sectorids \ read all sector ids
190         JSR format    \ format sector on drive 1
200         JSR copytrack \ read and write sector
210         JMP output
220 .notformatted
```

```
230         JSR deform    \ de-format this track
240 .output
250         JSR printbyte \ print track number
260         INC physical  \ increment physical track number
270         LDA physical  \ load physical track number
280         CMP last      \ all done?
290         BNE mainloop  \ if not copy next track
300         JSR osnewl
310         RTS           \ return to BASIC
320 .escape
330         LDA &FF       \ escape flag
340         BMI pressed   \ bit 7 set if pressed
350         RTS
360 .pressed
370         LDA #&7E
380         JSR osbyte    \ acknowledge Escape
390         BRK
400         BRK
410         EQUS "Escape"
420         BRK
430 .seek
440         LDA physical  \ physical track number
450         STA seekblock+7
460         LDA #&00      \ drive 0
470         STA seekblock \ store drive number
480         LDA #&7F
490         LDX #seekblock MOD 256
500         LDY #seekblock DIV 256
510         JSR osword
520         LDA seekblock+8 \ result
530         BNE seekerror \ = 0 if OK
540         LDA #&01      \ drive 1
550         STA seekblock \ store drive number
560         LDA #&7F
570         LDX #seekblock MOD 256
580         LDY #seekblock DIV 256
590         JSR osword
600         LDA seekblock+8 \ result
610         BNE seekerror \ = 0 if OK
620         RTS
630 .seekerror
640         BRK
650         BRK
660         EQUS "Seek error"
670         BRK
680 .format
690         LDA physical  \ physical track number
700         STA formblock+7 \ store physical track
710         LDX table+3   \ data size code
720         LDA gap,X     \ load gap 3 for these sectors
730         STA formblock+8 \ store for formatting
740         LDA #&7F
750         LDX #formblock MOD 256
760         LDY #formblock DIV 256
770         JSR osword
780         LDA formblock+12 \ result
790         BNE formerror \ = 0 if OK
800         RTS
810 .formerror
820         BRK
830         BRK
840         EQUS "Format error"
850         BRK
860 .deform
870         LDA physical  \ load physical track number
880         STA deblock+7 \ store physical track
890         LDA #&7F
900         LDX #deblock MOD 256
910         LDY #deblock DIV 256
920         JSR osword    \ de-format track
930         LDA deblock+12 \ result
940         BNE deerror   \ = 0 if OK
950         RTS
960 .deerror
970         BRK
980         BRK
990         EQUS "De-format error"
```

```
1000         BRK
1010 .register
1020         STA regblock+8 \ value to put in register
1030         LDA #&00      \ drive 0
1040         STA regblock
1050         LDA #&12      \ write track register 0/2
1060         STA regblock+7 \ register number
1070         LDA #&7F
1080         LDX #regblock MOD 256
1090         LDY #regblock DIV 256
1100         JSR osword
1110         LDA regblock+9 \ result
1120         BNE regerror  \ = 0 if OK
1130         LDA #&01      \ drive 1
1140         STA regblock
1150         LDA #&1A      \ write track register 1/3
1160         STA regblock+7 \ register number
1170         LDA #&7F
1180         LDX #regblock MOD 256
1190         LDY #regblock DIV 256
1200         JSR osword
1210         LDA regblock+9 \ result
1220         BNE regerror  \ = 0 if OK
1230         RTS
1240 .regerror
1250         BRK
1260         BRK
1270         EQUS "Special register error"
1280         BRK
1290 .firstsector
1300         LDA physical  \ physical track number
1310         STA idsblock+7 \ store physical track
1320         LDA #&01      \ one sector
1330         STA idsblock+9 \ number of ids
1340         LDA #&7F
1350         LDX #idsblock MOD 256
1360         LDY #idsblock DIV 256
1370         JSR osword
1380         LDA idsblock+10 \ result
1390         AND #&1E      \ = 0 if formatted
1400         RTS
1410 .sectorids
1420         LDX table+3   \ load data size code
1430         LDA sizes,X   \ load number of sectors
1440         STA idsblock+9 \ store number of sectors
1450         ASL A         \ *2
1460         ASL A         \ *4
1470         SEC
1480         SBC #&04      \ sectors*4-4
1490         STA sectornumber \ store index on sectors
1500         TXA           \ transfer data size code
1510         ASL A         \ *2
1520         ASL A         \ *4
1530         ASL A         \ *8
1540         ASL A         \ *16
1550         ASL A         \ *32
1560         ORA idsblock+9 \ add number of sectors
1570         STA copyblock+9 \ sector size/number
1580         STA formblock+9 \ sector size/number
1590         LDA #&7F
1600         LDX #idsblock MOD 256
1610         LDY #idsblock DIV 256
1620         JSR osword
1630         LDA idsblock+10 \ result
1640         AND #&1E
1650         BNE idserror  \ = 0 if OK
1660         RTS
1670 .idserror
1680         BRK
1690         BRK
1700         EQUS "Sector ID Error"
1710         BRK
1720 .copytrack
1730         LDX sectornumber \ load index on table
1740         LDA table+2,X \ load logical sector number
1750         STA copyblock+8 \ store for read sector
1760 .lowest
```

```
1770         DEX
1780         DEX
1790         DEX
1800         DEX
1810         BMI finished
1820         LDA table+2,X \ load logical sector number
1830         CMP copyblock+8 \ is it lower than the last one?
1840         BCS lowest    \ branch if not lowest sector
1850         STA copyblock+8 \ store if it is lower
1860         BCC lowest    \ look for lower sector number
1870 .finished
1880         LDA table     \ load logical track number
1890         STA copyblock+7 \ and store for read
1900         JSR register  \ write track register
1910         LDA #&00      \ drive 0
1920         STA copyblock
1930         LDA #&57      \ read sector command
1940         STA copyblock+6
1950         LDA #&7F
1960         LDX #copyblock MOD 256
1970         LDY #copyblock DIV 256
1980         JSR osword
1990         LDA copyblock+10 \ result
2000         BEQ notdel    \ not deleted data
2010         CMP #&20      \ deleted data result
2020         BNE readerror \ error if not &20
2030         LDA #&4F      \ write deleted data command
2040         BNE savecom
2050 .notdel
2060         LDA #&4B      \ write data command
2070 .savecom
2080         STA copyblock+6
2090         LDA #&01      \ drive 1
2100         STA copyblock
2110         LDA #&7F
2120         LDX #copyblock MOD 256
2130         LDY #copyblock DIV 256
2140         JSR osword
2150         LDA copyblock+10 \ result
2160         BNE writeerror \ = 0 if OK
2170         LDA physical
2180         JSR register  \ write track register
2190         RTS
2200 .readerror
2210         LDA physical
2220         JSR register
2230         BRK
2240         BRK
2250         EQUS "Read error"
2260         BRK
2270 .writeerror
2280         LDA physical
2290         JSR register
2300         BRK
2310         BRK
2320         EQUS "Write error"
2330         BRK
2340 .printbyte
2350         LDA physical  \ print physical track number
2360         PHA
2370         LSR A
2380         LSR A
2390         LSR A
2400         LSR A
2410         JSR nybble    \ print MS nybble
2420         PLA
2430         JSR nybble    \ print LS nybble
2440         LDA #ASC(" ")
2450         JSR oswrch    \ print space
2460         JMP oswrch    \ print space and return
2470 .nybble
2480         AND #&0F
2490         SED
2500         CLC
2510         ADC #&90
2520         ADC #&40
2530         CLD
```

```
2540          JMP oswrch     \ print nybble and return
2550 .seekblock
2560          EQUB &00       \ drive 0/1
2570          EQUD &00       \ does not matter
2580          EQUB &01       \ 1 parameter
2590          EQUB &69       \ seek command
2600          EQUB &00       \ physical track number
2610          EQUB &00       \ result byte
2620 .regblock
2630          EQUB &00       \ drive 0/1
2640          EQUD &00       \ does not matter
2650          EQUB &02       \ 2 parameters
2660          EQUB &7A       \ write special register
2670          EQUB &00       \ register number
2680          EQUB &00       \ value to put in register
2690          EQUB &00       \ result byte
2700 .idsblock
2710          EQUB &00       \ drive 0
2720          EQUD table     \ address of buffer
2730          EQUB &03       \ 3 parameters
2740          EQUB &5B       \ read sector IDs command
2750          EQUB &00       \ physical track number
2760          EQUB &00       \ always &00
2770          EQUB &00       \ number of IDs to be read
2780          EQUB &00       \ result byte
2790 .copyblock
2800          EQUB &00       \ drive 0/1
2810          EQUD buffer    \ address of buffer
2820          EQUB &03       \ 3 parameters
2830          EQUB &57       \ read data and deleted data
2840          EQUB &00       \ logical track number
2850          EQUB &00       \ logical sector number
2860          EQUB &00       \ sector size/number
2870          EQUB &00       \ result byte
2880 .formblock
2890          EQUB &01       \ drive 1
2900          EQUD table     \ sector table
2910          EQUB &05       \ 5 parameters
2920          EQUB &63       \ format track command
2930          EQUB &00       \ physical track number
2940          EQUB &00       \ gap 3 size
2950          EQUB &00       \ sector size/number
2960          EQUB &00       \ gap 5 size
2970          EQUB &10       \ gap 1 size
2980          EQUB &00       \ result byte
2990 .deblock
3000          EQUB &01       \ drive 1
3010          EQUD detable   \ sector table
3020          EQUB &05       \ 5 parameters
3030          EQUB &63       \ format track command
3040          EQUB &00       \ physical track number
3050          EQUB &00       \ gap 3 size
3060          EQUB &C1       \ sector size/number
3070          EQUB &00       \ gap 5 size
3080          EQUB &10       \ gap 1 size
3090          EQUB &00       \ result byte
3100 .detable
3110          EQUD &04000000
3120 .gap
3130          EQUB 11        \ Gap 3, 18 sectors
3140          EQUB 21        \ Gap 3, 10 sectors
3150          EQUB 74        \ Gap 3, 5 sectors
3160          EQUB 255       \ Gap 3, 2 sectors
3170          EQUB 0         \ Gap 3, 1 sector
3180 .sizes
3190          EQUB 18
3200          EQUB 10
3210          EQUB 5
3220          EQUB 2
3230          EQUB 1
3240 .physical
3250          EQUB &00
3260 .sectornumber
3270          EQUB &00
3280 .last
3290          EQUB &00
3300 ]
```

```
3310 NEXT
3320 INPUT'"Number of tracks (40/80) "tracks$
3330 IF tracks$="40" ?last=40 ELSE ?last=80
3340 PRINT'"Insert ";?last;" track source disc in :0"
3350 PRINT"Insert ";?last;" track destination disc in :1"
3360 PRINT'"Press Spacebar to copy from :0 to :1"
3370 REPEAT
3380 UNTIL GET=32
3390 CALL mcode
```

When you have designed a disc format and a data storage and retrievel system which cannot be copied with COPYDFS you should attempt to copy the disc using COPYALL. This program uses a similar algorithm to that used by COPYDFS but, instead of copying a track at a time, it attempts to copy the disc a sector at a time. This method of copying takes much longer but it does allow the program to check if deleted data is being used on an individual sector rather than on a track as a whole. This will give a better copy of the original disc but it is not a fool-proof method of duplicating 'difficult' discs.

When you start to think about designing a disc format which can not be copied by COPYALL I suggest that you should think carefully about using unusual logical track and sector combinations in unusual or unexpected ways. I cannot tell you what to do because everyone else who reads this module will then know what you have done. You can not make a disc unhackable (if there is such a word) but it is possible to make a disc uncopyable by either of these programs or any of the commercial disc duplication programs available at the time of writing (November 1987). All the information you need to do it has been presented in these DFS modules. You will have to look carefully at the information provided and think hard about what a program would need to do to duplicate your disc.

```
  10 REM: COPYALL
  20 osnewl=&FFE7
  30 oswrch=&FFEE
  40 osword=&FFF1
  50 osbyte=&FFF4
  60 DIM table &50
  70 DIM mcode &500
  80 DIM buffer &1000
  90 FOR pass=0 TO 2 STEP 2
 100 P%=mcode
 110 [       OPT pass
 120         JSR osnewl
 130 .mainloop
 140         JSR escape    \ check escape flag
 150         JSR seek      \ seek physical tracks 0 - 40
 160         JSR firstsector \ read sector id first sector
 170         BNE notformatted \ if error then track not formatted
 180         JSR sectorids \ read all sector ids
 190         JSR format    \ format sector on drive 1
 200 .loopsector
 210         JSR escape    \ check escape flag
 220         JSR copysector \ read and write sector
 230         BPL loopsector \ copy next sector
 240         LDA physical  \ physical track number
 250         JSR register  \ write track register
 260         JMP output
 270 .notformatted
 280         JSR deform    \ de-format this track
 290 .output
 300         JSR printbyte \ print track number
 310         INC physical  \ increment physical track number
 320         LDA physical  \ load physical track number
 330         CMP last      \ all done?
 340         BNE mainloop  \ if not copy next track
 350         JSR osnewl
 360         RTS           \ return to BASIC
 370 .escape
 380         LDA &FF       \ escape flag
 390         BMI pressed   \ bit 7 set if pressed
 400         RTS
 410 .pressed
 420         LDA #&7E
 430         JSR osbyte    \ acknowledge Escape
```

```
 440         BRK
 450         BRK
 460         EQUS "Escape"
 470         BRK
 480 .seek
 490         LDA physical  \ physical track number
 500         STA seekblock+7
 510         LDA #&00      \ drive 0
 520         STA seekblock \ store drive number
 530         LDA #&7F
 540         LDX #seekblock MOD 256
 550         LDY #seekblock DIV 256
 560         JSR osword
 570         LDA seekblock+8 \ result
 580         BNE seekerror \ = 0 if OK
 590         LDA #&01      \ drive 1
 600         STA seekblock \ store drive number
 610         LDA #&7F
 620         LDX #seekblock MOD 256
 630         LDY #seekblock DIV 256
 640         JSR osword
 650         LDA seekblock+8 \ result
 660         BNE seekerror \ = 0 if OK
 670         RTS
 680 .seekerror
 690         BRK
 700         BRK
 710         EQUS "Seek error"
 720         BRK
 730 .format
 740         LDA physical  \ physical track number
 750         STA formblock+7 \ store physical track
 760         LDA table+3   \ data size code
 770         TAX           \ used as index later
 780         ASL A         \ *2
 790         ASL A         \ *4
 800         ASL A         \ *8
 810         ASL A         \ *16
 820         ASL A         \ *32
 830         STA formblock+9 \ store datacode*32
 840         ORA #&01      \ add 1
 850         STA copyblock+9 \ store datacode*32+1
 860         LDA sizes,X   \ load number of sectors
 870         ORA formblock+9 \ add datacode*32
 880         STA formblock+9 \ store datacode*32+numbersectors
 890         LDA gap,X     \ load gap 3 for these sectors
 900         STA formblock+8 \ store for formatting
 910         LDA #&7F
 920         LDX #formblock MOD 256
 930         LDY #formblock DIV 256
 940         JSR osword
 950         LDA formblock+12 \ result
 960         BNE formerror \ = 0 if OK
 970         LDX table+3   \ load data size code
 980         LDA sizes,X   \ load number of sectors
 990         ASL A         \ *2
1000         ASL A         \ *4
1010         SEC
1020         SBC #&04      \ sectors*4-4
1030         STA sectornumber \ store index on sectors
1040         RTS
1050 .formerror
1060         BRK
1070         BRK
1080         EQUS "Format error"
1090         BRK
1100 .deform
1110         LDA physical  \ load physical track number
1120         STA deblock+7 \ store physical track
1130         LDA #&7F
1140         LDX #deblock MOD 256
1150         LDY #deblock DIV 256
1160         JSR osword    \ de-format track
1170         LDA deblock+12 \ result
1180         BNE deerror   \ = 0 if OK
1190         RTS
1200 .deerror
```

```
1210         BRK
1220         BRK
1230         EQUS "De-format error"
1240         BRK
1250 .register
1260         STA regblock+8 \ value to put in register
1270         LDA #&00      \ drive 0
1280         STA regblock
1290         LDA #&12      \ write track register 0/2
1300         STA regblock+7 \ register number
1310         LDA #&7F
1320         LDX #regblock MOD 256
1330         LDY #regblock DIV 256
1340         JSR osword
1350         LDA regblock+9 \ result
1360         BNE regerror  \ = 0 if OK
1370         LDA #&01      \ drive 1
1380         STA regblock
1390         LDA #&1A      \ write track register 1/3
1400         STA regblock+7 \ register number
1410         LDA #&7F
1420         LDX #regblock MOD 256
1430         LDY #regblock DIV 256
1440         JSR osword
1450         LDA regblock+9 \ result
1460         BNE regerror  \ = 0 if OK
1470         RTS
1480 .regerror
1490         BRK
1500         BRK
1510         EQUS "Special register error"
1520         BRK
1530 .firstsector
1540         LDA physical  \ physical track number
1550         STA idsblock+7 \ store physical track
1560         LDA #&01      \ one sector
1570         STA idsblock+9 \ number of ids
1580         LDA #&7F
1590         LDX #idsblock MOD 256
1600         LDY #idsblock DIV 256
1610         JSR osword
1620         LDA idsblock+10 \ result
1630         AND #&1E      \ = 0 if formatted
1640         RTS
1650 .sectorids
1660         LDX table+3   \ load data size code
1670         LDA sizes,X   \ load number of sectors
1680         STA idsblock+9 \ store number of sectors
1690         LDA #&7F
1700         LDX #idsblock MOD 256
1710         LDY #idsblock DIV 256
1720         JSR osword
1730         LDA idsblock+10 \ result
1740         AND #&1E
1750         BNE idserror  \ = 0 if OK
1760         RTS
1770 .idserror
1780         BRK
1790         BRK
1800         EQUS "Sector ID Error"
1810         BRK
1820 .copysector
1830         LDX sectornumber \ load index on table
1840         LDA table+2,X \ load logical sector number
1850         STA copyblock+8 \ store for read sector
1860         LDA table,X   \ load logical track number
1870         STA copyblock+7 \ and store for read
1880         JSR register  \ write track register
1890         LDA #&00      \ drive 0
1900         STA copyblock
1910         LDA #&57      \ read sector command
1920         STA copyblock+6
1930         LDA #&7F
1940         LDX #copyblock MOD 256
1950         LDY #copyblock DIV 256
1960         JSR osword
1970         LDA copyblock+10
```

```
1980         BEQ notdel    \ not deleted data
1990         CMP #&20      \ deleted data result
2000         BNE readerror \ error if not &20
2010         LDA #&4F      \ write deleted data command
2020         BNE savecom
2030 .notdel
2040         LDA #&4B      \ write data command
2050 .savecom
2060         STA copyblock+6
2070         LDA #&01      \ drive 1
2080         STA copyblock
2090         LDA #&7F
2100         LDX #copyblock MOD 256
2110         LDY #copyblock DIV 256
2120         JSR osword
2130         LDA copyblock+10 \ result
2140         BNE writeerror \ = 0 if OK
2150         SEC
2160         LDA sectornumber \ sector index on table
2170         SBC #&04
2180         STA sectornumber \ index=index-4
2190         RTS
2200 .readerror
2210         LDA physical  \ physical track number
2220         JSR register  \ write track register
2230         BRK
2240         BRK
2250         EQUS "Read error"
2260         BRK
2270 .writeerror
2280         LDA physical  \ physical track number
2290         JSR register  \ write track register
2300         BRK
2310         BRK
2320         EQUS "Write error"
2330         BRK
2340 .printbyte
2350         LDA physical  \ print physical track number
2360         PHA
2370         LSR A
2380         LSR A
2390         LSR A
2400         LSR A
2410         JSR nybble    \ print MS nybble
2420         PLA
2430         JSR nybble    \ print LS nybble
2440         LDA #ASC(" ")
2450         JSR oswrch    \ print space
2460         JMP oswrch    \ print space and return
2470 .nybble
2480         AND #&0F
2490         SED
2500         CLC
2510         ADC #&90
2520         ADC #&40
2530         CLD
2540         JMP oswrch    \ print nybble and return
2550 .seekblock
2560         EQUB &00      \ drive 0/1
2570         EQUD &00      \ does not matter
2580         EQUB &01      \ 1 parameter
2582         EQUB &69      \ seek command
2584         EQUB &00      \ physical track number
2586         EQUB &00      \ result byte
2590 .regblock
2600         EQUB &00      \ drive 0/1
2610         EQUD &00      \ does not matter
2620         EQUB &02      \ 2 parameters
2622         EQUB &7A      \ write special register
2624         EQUB &00      \ register number
2626         EQUB &00      \ value to put in register
2630         EQUB &00      \ result byte
2640 .idsblock
2650         EQUB &00      \ drive 0
2660         EQUD table    \ address of buffer
2670         EQUB &03      \ 3 parameters
2672         EQUB &5B      \ read sector IDs
```

```
2674          EQUB &00      \ physical track number
2676          EQUB &00      \ always &00
2678          EQUB &00      \ number of IDs to be read
2680          EQUB &00      \ result byte
2690 .copyblock
2700          EQUB &00      \ drive 0/1
2710          EQUD buffer   \ address of buffer
2720          EQUB &03      \ 3 parameters
2722          EQUB &57      \ read data and deleted data
2724          EQUB &00      \ logical track number
2726          EQUB &00      \ logical sector number
2728          EQUB &00      \ sector size/number
2730          EQUB &00      \ result byte
2740 .formblock
2750          EQUB &01      \ drive 1
2760          EQUD table    \ sector table
2770          EQUB &05      \ 5 parameters
2772          EQUB &63      \ format track command
2774          EQUB &00      \ physical track number
2776          EQUB &00      \ gap 3 size
2778          EQUB &00      \ sector size/number
2780          EQUB &00      \ gap 5 size
2782          EQUB &10      \ gap 1 size
2784          EQUB &00      \ result byte
2790 .deblock
2800          EQUB &01      \ drive 1
2810          EQUD detable  \ sector table
2820          EQUB &05      \ 5 parameters
2822          EQUB &63      \ format track command
2824          EQUB &00      \ physical track number
2826          EQUB &00      \ gap 3 size
2828          EQUB &C1      \ sector size/number
2830          EQUB &00      \ gap 5 size
2832          EQUB &10      \ gap 1 size
2834          EQUB &00      \ result byte
2840 .detable
2850          EQUD &04000000
2860 .gap
2870          EQUB 11       \ Gap 3, 18 sectors
2880          EQUB 21       \ Gap 3, 10 sectors
2890          EQUB 74       \ Gap 3, 5 sectors
2900          EQUB 255      \ Gap 3, 2 sectors
2910          EQUB 0        \ Gap 3, 1 sector
2920 .sizes
2930          EQUB 18
2940          EQUB 10
2950          EQUB 5
2960          EQUB 2
2970          EQUB 1
2980 .physical
2990          EQUB &00
3000 .sectornumber
3010          EQUB &00
3020 .last
3030          EQUB &00
3040 ]
3050 NEXT
3060 INPUT'"Number of tracks (40/80) "tracks$
3070 IF tracks$="40" ?last=40 ELSE ?last=80
3080 PRINT'"Insert ";?last;" track source disc in :0"
3090 PRINT"Insert ";?last;" track destination disc in :1"
3100 PRINT'"Press Spacebar to copy from :0 to :1"
3110 REPEAT
3120 UNTIL GET=32
3130 CALL mcode
```

```
130 P%=PAGE+&100
140 [       OPT pass
150 .firstbyte
160         EQUW &FF0D    \ NEW the BASIC program
170 .start
180         LDA #&0F
190         JSR osasci    \ scroll mode
200         LDX #title MOD 256
210         LDY #title DIV 256
220         JSR print     \ print title
230 .getdata
240         LDX #drivenum MOD 256
250         LDY #drivenum DIV 256
260         JSR print     \ which drive?
270 .whichdrive
280         JSR osrdch
290         BCS escape
300         SEC
310         SBC #ASC("0")
320         BMI whichdrive \ drive 0-3
330         CMP #&04
340         BCS whichdrive \ drive 0-3
350         STA block     \ format parameter block
360         STA catblock  \ catalogue parameter block
370         ADC #ASC("0")
380         JSR osasci    \ print 0, 1, 2 or 3
390         LDX #tracknum MOD 256
400         LDY #tracknum DIV 256
410         JSR print     \ 40 or 80 tracks?
420 .whichtrack
430         JSR osrdch
440         BCS escape
450         LDX #&27      \ 40 tracks
460         CMP #ASC("4")
470         BEQ continue
480         CMP #ASC("8")
490         BNE whichtrack
500         LDX #&4F      \ 80 tracks
510 .continue
520         STX finish    \ store number of tracks
530         JSR osasci    \ print "8" or "4"
540         LDA #ASC("0")
550         JSR osasci    \ print "0" to make "40" or "80"
560         LDX #ready MOD 256
570         LDY #ready DIV 256
580         JSR print     \ ready to format?
590         JSR osrdch
600         BCS escape
610         PHA           \ temp store for answer
620         JSR osasci    \ print answer
630         JSR osnewl
640         PLA           \ pull answer
650         AND #&DF      \ upper case
660         CMP #ASC("Y")
670         BNE getdata
680         JSR osnewl
690         LDA #0
700         STA track
710         LDA #&01      \ data size
720         LDX #&10      \ gap 3
730         LDY #&2A      \ number of sectors
740         JSR setup
750         LDA #&7F
760         LDX #block MOD 256
770         LDY #block DIV 256
780         JSR osword    \ format track 0 ten sectors
790         LDA result    \ load result byte
800         BEQ trackzero \ format OK if result = 0
810 .error
820         BRK
830         BRK
840         EQUS "Format error"
850         BRK
860 .escape
870         LDA #&7E
880         JSR osbyte    \ acknowledge Escape
890         BRK
```

```
 900         BRK
 910         EQUS "Escape"
 920         BRK
 930 .trackzero
 940         LDA #&7F
 950         LDX #catblock MOD 256
 960         LDY #catblock DIV 256
 970         JSR osword    \ store empty catalogue
 980         LDA catresult \ check result byte
 990         BNE error     \ quit if error
1000         JSR printbyte \ print track 00
1010 .loop
1020         LDA &FF       \ poll escape flag
1030         BMI escape    \ bit 7 set if Escape pressed
1040         INC track     \ increment track number
1050         LDA #&02      \ data size
1060         LDX #&4A      \ gap 3
1070         LDY #&45      \ number of sectors
1080         JSR setup
1090         LDA #&7F
1100         LDX #block MOD 256
1110         LDY #block DIV 256
1120         JSR osword    \ format track with 5 sectors
1130         LDA result    \ load result byte
1140         BNE error     \ quit if error
1150         LDA track     \ load track number
1160         PHA
1170         JSR printbyte \ print track number
1180         PLA
1190         CMP finish    \ is that the last track?
1200         BCC loop      \ branch if more tracks to format
1210         LDX #another MOD 256
1220         LDY #another DIV 256
1230         JSR print     \ another?
1240         JSR osrdch
1250         BCS escape
1260         PHA           \ temp store for answer
1270         JSR osasci    \ print answer
1280         JSR osnewl
1290         PLA           \ pull answer
1300         AND #&DF      \ upper case
1310         CMP #ASC("Y")
1320         BNE return
1330         JMP getdata
1340 .setup
1350         STX gap3
1360         STY numsectors
1370         LDX #39
1380         LDY track
1390         STY physical
1400 .setloop
1410         STA table,X
1420         DEX
1430         DEX
1440         DEX
1450         PHA
1460         TYA
1470         STA table,X
1480         PLA
1490         DEX
1500         BPL setloop
1510 .return
1520         RTS
1530 .print
1540         STX zeropage
1550         STY zeropage+1
1560         LDY #0
1570 .printloop
1580         LDA (zeropage),Y
1590         BEQ endprint
1600         JSR osasci
1610         INY
1620         BNE printloop
1630 .endprint
1640         RTS
1650 .printbyte
1660         PHA
```

```
1670         LSR A
1680         LSR A
1690         LSR A
1700         LSR A
1710         JSR nybble    \ print MS nybble
1720         PLA
1730         JSR nybble    \ print LS nybble
1740         LDA #ASC(" ")
1750         JSR oswrch    \ print space
1760         JMP oswrch    \ print space
1770 .nybble
1780         AND #&0F
1790         SED
1800         CLC
1810         ADC #&90
1820         ADC #&40
1830         CLD
1840         JMP oswrch    \ print nybble and return
1850 .block
1860         EQUB &00      \ drive number 0-3
1870         EQUD table    \ sector table
1880         EQUB &05      \ 5 parameters
1890         EQUB &63      \ format track
1900 .physical
1910         EQUB &00      \ physical track number 0
1920 .gap3
1930         EQUB &15      \ gap 3
1940 .numsectors
1950         EQUB &2A      \ 10 sectors of 256 bytes
1960         EQUB &00      \ gap 5
1970         EQUB &10      \ gap 1
1980 .result
1990         EQUB &00      \ result byte
2000 .table
2010         EQUD &00000000
2020         EQUD &00010000
2030         EQUD &00020000
2040         EQUD &00030000
2050         EQUD &00040000
2060         EQUD &00050000
2070         EQUD &00060000
2080         EQUD &00070000
2090         EQUD &00080000
2100         EQUD &00090000
2110 .catalogue
2120         EQUB &15      \ disc title (disable VDU)
2130         OPT FNfill(7) \ 7 zero bytes
2140         EQUS "!BOOT  $" \ next 8 bytes
2150         OPT FNfill(240) \ end of first sector
2160         OPT FNfill(5) \ start of second sector
2170         EQUB &08      \ number of files * 8
2180         EQUW &0A20    \ 10 sectors and *OPT 4,2
2190         EQUD &00      \ load and exec = &0000
2200         EQUW &0800    \ length = &800 bytes
2210         EQUB &00      \ MS 2 bits of sector number
2220         EQUB &02      \ starting at sector 2
2230         OPT FNfill(240)
2240 .catblock
2250         EQUB &00      \ drive number
2260         EQUD catalogue \ address of buffer
2270         EQUB &03      \ number of parameters
2280         EQUB &4B      \ save data multi sector
2290         EQUB &00      \ logical track
2300         EQUB &00      \ start logical sector
2310         EQUB &22      \ 2 sectors of 256 bytes
2320 .catresult
2330         EQUB &00      \ result byte
2340 .title
2350         EQUB &0D
2360         EQUS "5 Sector DFS Format"
2370         EQUB &0D
2380         BRK
2390 .drivenum
2400         EQUB &0D
2410         EQUS "Drive number? (0-3) "
2420         BRK
2430 .tracknum
```

```
 2440         EQUB &0D
 2450         EQUS "40 or 80 tracks? (4/8) "
 2460         BRK
 2470 .ready
 2480         EQUB &0D
 2490         EQUS "Ready to format? (Y/N) "
 2500         BRK
 2510 .another
 2520         EQUB &0D
 2530         EQUS "Another? (Y/N) "
 2540         BRK
 2550 .track
 2560         EQUB &00      \ physical track number
 2570 .finish
 2580         EQUB &00      \ last track number
 2590 .lastbyte
 2600 ]
 2610 NEXT
 2620 INPUT'"Save filename = "filename$
 2630 IF filename$ = "" THEN END
 2640 *OPT1,2
 2650 OSCLI("SAVE "+filename$+" "+STR$~(mcode)+"+"+STR$~(las
      tbyte-firstbyte)+" "+STR$~(start)+" "+STR$~(firstbyte))
 2660 *OPT1,0
 2670 END
 2680 DEF FNfill(size)
 2690 FOR count = 1 TO size
 2700 ?O%=0
 2710 O%=O%+1
 2720 P%=P%+1
 2730 NEXT
 2740 =pass
```

There are a number of points worth noting about the program SECTOR5. The object code is used to create a 5 sector per track formatted disc and, for the sake of demonstrating the protection techniques, the object code will also be stored on the disc it formats. The program assembles the object code to run at PAGE+&100. This is so that the programs which will be used to store the object code on a protected disc and to read that code back from the disc can be located at PAGE. These saving and loading programs are very short and &100 bytes is more than enough room for them. When you design your own protection system you may need more space for your saving and loading programs.

The formatting program introduced in module 3 created an empty catalogue which only contained the number of sectors available on the disc. The catalogue created by the program above actually contains a disc title, some file information and the start up option. The disc title is the ASCII character &15 which disables the VDU and so prevents the disc being catalogued. The dummy file $.!BOOT is entered into the catalogue and it is specified as &800 bytes long starting in sector &02. This means that the dummy !BOOT file uses all the available space on track &00. The number of available sectors on the disc is specified as &0A and so the catalogue and the dummy !BOOT file use all the available space on the disc. The start up option is equivalent to *OPT 4,2 so that the !BOOT file will be *RUN on Shift+Break. This formatting program produces the sector map shown in figure 1. You can see that it has not been optimised for speed and you may like to modify using the logical sector offsets described in module 3.

```
                DFS format physical sector numbers

              | 00  01  02  03  04  05  06  07  08  09
           ---+---------------------------------------
        T  00 | 00  01  02  03  04  05  06  07  08  09
        r  01 |   01      02      03      04      05    Logical
        a  02 |   01      02      03      04      05    sector
        c  03 |   01      02      03      04      05    numbers
        k  04 |   01      02      03      04      05
        s  05 |   01 ...
```

Figure 1. The distribution of logical sectors

When the disc has been formatted you can use the program IDSDUMP (from module 0) to check the ID fields and the program VERIFY (also from module 0) to verify the format.

The object code program FORM5, which is used to format the disc, can be stored on the newly formatted disc using the object code generated by the program ENCODE. Run ENCODE and choose a suitable filename for the object code it produces when prompted. For the sake of this demonstration I will assume that you will call it SAVE5. The object code program SAVE5 will save the &400 byte program FORM5 onto the first two logical sectors of track &01, storing the data as deleted data.

*LOAD FORM5 and then type *RUN SAVE5. Swap the DFS disc for the newly formatted 5 sector per track disc and press the spacebar. The object code program FORM5 will be stored on the disc. You can use the program VERIFY to make sure that the data is sucessfully stored as deleted data.

```
   10 REM: ENCODE
   20 DIM mcode &100
   25 page=PAGE
   30 osrdch=&FFE0
   40 oswrch=&FFEE
   50 osword=&FFF1
   60 FORpass=4 TO 6 STEP 2
   70 O%=mcode
   80 P%=page
   90 [       OPT pass
  100 .firstbyte
  110         EQUW &FF0D    \ NEW the BASIC program
  120 .start
  130         LDX #&00
  140 .loop
  150         LDA message,X
  160         BEQ end
  170         JSR oswrch
  180         INX
  190         BNE loop
  200 .end
  210         JSR osrdch
  220         BCC save
  230         BRK
  240         BRK
  250         EQUS "Escape"
  260         BRK
  270 .save
  280         LDA #&7F
  290         LDX #block MOD 256
  300         LDY #block DIV 256
  310         JSR osword    \ write deleted data
  320         LDA result    \ load result byte
  330         AND #&1E      \ isolate error code
  340         BNE error
  350         BRK
  360         BRK
  370         EQUS "Write OK"
  380 .error
  390         BRK
  400         BRK
  410         EQUS "Write failed"
  420         BRK
  430 .block
  440         EQUB &FF      \ current drive
  450         EQUD page+&100 \ start at PAGE+&100
  460         EQUB &03      \ 3 parameters
  470         EQUB &4F      \ write deleted multi-sector
  480         EQUB &01      \ logical track &01
  490         EQUB &00      \ start logical sector &00
  500         EQUB &42      \ 2 sectors of 512 bytes
  510 .result
  520         EQUB &00      \ result byte
  530 .message
  540         EQUS "Press Space to save data on current disc"
  550         BRK
  560 .lastbyte
  570 ]
  580 NEXT
```

```
 590 INPUT'"Save filename = "filename$
 600 IF filename$ = "" THEN END
 610 *OPT1,2
 620 OSCLI("SAVE "+filename$+" "+STR$~(mcode)+"+"+STR$~(las
     tbyte-firstbyte)+" "+STR$~(start)+" "+STR$~(firstbyte))
 630 *OPT1,0
 640 END
```

The data stored on the disc using SAVE5 cannot be read back off the disc using any of the DFS star commands. The disc has been formatted so that a !BOOT file will be *RUN on Shift+Break and so the dummy !BOOT file entered into the catalogue by the formatting program needs to be replaced with a real !BOOT file which will load and run the program stored on track &01.

The program DECODE will produce an appropriate machine code !BOOT file which simply reverses the Write Deleted Data operation which put the data onto the disc. When the deleted data have been reloaded into memory the command CALL (PAGE+&102) <Return> is inserted into the keyboard buffer, and BASIC is entered using Osbyte &8E. The VDU is disabled before inserting the data into the keyboard buffer (lines 460-470) and re-enabled before the command is executed (line 610). This hides what you are doing from the disc user. Any command, or series of commands, can be executed in this way so that BASIC programs as well as machine code programs can be stored on, and run from, the copy-protected disc.

The optional code in lines 140 to 230 of DECODE can be used to further protect your software but you should start to think of your own ideas to incorporate with these techniques. Disc copy-protection is only useful if it is unique and very difficult to break. These programs are neither but they are a good introduction to the necessary techniques and the discs they produce cannot be duplicated with the *BACKUP command.

```
  10 REM: DECODE
  20 page=PAGE
  30 DIM mcode &100
  40 osasci=&FFE3
  50 osword=&FFF1
  60 osbyte=&FFF4
  70 FORpass=4 TO 6 STEP 2
  80 O%=mcode
  90 P%=page
 100 [       OPT pass
 110 .firstbyte
 120         EQUW &FF0D    \ NEW the BASIC program
 130 .start
 140                       \ LDA #&C8 \ write Break effect
 150                       \ LDX #&02 \ clear memory on Break
 160                       \ LDY #&00
 170                       \ JSR osbyte
 180                       \ LDA #&4C \ JMP opcode
 190                       \ LDX #&87 \ JMP &287 gives an
 200                       \ LDY #&02 \ endless loop on Break
 210                       \ STA &287
 220                       \ STX &288
 230                       \ STY &289
 240         LDA #&7F
 250         LDX #block MOD 256
 260         LDY #block DIV 256
 270         JSR osword    \ read deleted data
 280         LDA result    \ load result byte
 290         AND #&1E      \ isolate error code
 300         BNE error
 310         LDA #&FF      \ initialise offset
 320         PHA           \ save offset
 330 .pull
 340         PLA
 350         TAX
 360         INX           \ increment offset
 370         TXA
 380         PHA           \ store current offset
 390         LDY keyboard,X \ load byte for keyboard buffer
 400         BEQ continue  \ branch if finished
 410         LDX #0
 420         LDA #&8A
 430         JSR osbyte    \ put byte into keyboard buffer
```

```
440         JMP pull
450 .continue
460         LDA #&15
470         JSR osasci    \ disable VDU
480         LDA #&BB
490         LDX #0
500         LDY #&FF
510         JSR osbyte    \ find BASIC
520         LDA #&8E
530         JMP osbyte    \ select BASIC no return
540 .error
550         BRK
560         BRK
570         EQUS "Disc read error"
580         BRK
590 .keyboard
600         EQUS "CALL &"+STR$~(page+&102)
610         EQUB &06      \ enable VDU
620         EQUB &0D
630         BRK
640 .block
650         EQUB &FF      \ current drive
660         EQUD page+&100 \ start at PAGE+&100
670         EQUB &03      \ 3 parameters
680         EQUB &57      \ read deleted multi-sector
690         EQUB &01      \ logical track &01
700         EQUB &00      \ start logical sector 0
710         EQUB &42      \ 2 sectors of 512 bytes
720 .result
730         EQUB &00      \ result byte
740 .lastbyte
750 ]
760 NEXT
770 PRINT'"Press Space to save !BOOT file"
780 REPEAT
790 UNTIL GET=32
800 *OPT1,2
810 OSCLI("SAVE $.!BOOT "+STR$~(mcode)+"+"+STR$~(lastbyte-
    firstbyte)+" "+STR$~(start)+" "+STR$~(firstbyte))
820 *OPT1,0
```