

RAMSOFT proudly presents:

DISCiPLE/+D TECHNICAL GUIDE

Revision 8 (09 Sept 1999)

SUMMARY:
=====

- 1 DISCiPLE/+D general features
- 2 Memory layout
- 3 I/O ports
- 3.1 ... DISCiPLE port 7Bh (123 dec.)
- 4 UFIA layout
- 5 System calls (hook codes)
- 5.1 ... Internal system calls
- 5.2 ... Programming example: loading a file
- 6 Disk layout
- 7 Filesystem details
- 8 File types table
- 9 GDOS/G+DOS extended BASIC commands
- 9.1 ... The network
- 9.2 ... The snapshot button
- 9.3 ... GDOS/G+DOS and UNIDOS error messages
- 10 GDOS/G+DOS and UNIDOS system variables
- 11 UNIDOS extended BASIC commands
- 12 Connectors pinouts
- 12.1 ... Connecting a PC 5.25" drive
- 13 VL1772 FDC programming info
- 14 Credits and contact info

DISCLAIMER
=====

Although we have tried to be very accurate, this document may contain some errors. The authors do not assume any responsibility for any loss and/or damage directly or indirectly caused to your system by use of any information reported here.

See the credits at the end of the document on how to contact the authors.

FOREWORDS
=====

We (Ramsoft) have been using the DISCiPLE interface for a long time and we have appreciated all the power of this disk system. We have spent many hours trying to understand the smallest details to make full use of its capabilities, and a lot still has to be known.

Now we have decided to release all the info we collected mostly ourselves, in the hope that it may be useful to anybody and that it will encourage the development of new programs and products for this wonderful system.

COMMON TERMS
=====

DRAM - The sector buffer.
 RPT - A system variable which points to a byte in the DRAM.
 UFIA - User File Information Area, a 24 byte structure which describes a file for system calls.
 DFCA - Disk File Channel Area.

1. DISCiPLE / +D features

8 KB EPROM (for disk BIOS)
 8 KB RAM
 NMI magic button (snapshot)
 Parallel port (not bi-directional)
 Floppy disk port (controlled by VL1772 FDC)
 High speed disk operations: load 128K in less than 7 seconds.

DISCiPLE only features:

Two ATARI joystick ports (Sinclair 1, Sinclair 2 / Kempston)
 Two network connectors (Interface 1 compatible, 3.5mm mono jack)
 Inhibit button (to lock out the interface)
 Throughout bus connector (to plug in other devices)

2. MEMORY LAYOUT

When the interface memory is paged in (see below), the first 16K of the Z80 address space have this mapping:

| Address | DISCiPLE GDOS | DISCiPLE UNIDOS | +D |
|---------|---------------|-----------------|--------|
| 0x0000 | 8K RAM | 8K ROM | 8K ROM |
| 0x2000 | 8K ROM | 8K RAM | 8K RAM |

So, UNI-DOS memory mapping is the same as +D, even on DISCiPLE. Read [3.1] to see how it is possible to swap ROM/RAM addresses on the DISCiPLE.

3. DISCiPLE and PlusD I/O PORTS

NOTE: Joystick 1 is both Kempston (port 1Fh) and Sinclair 2 (keys 6,7,8,9,0)
 Joystick 2 is Sinclair 1 (keys 1,2,3,4,5)
 Network is an Interface 1 compatible net.

DISCiPLE I/O ports:

| Port | In | Out | Notes |
|------|----------------|-----------------|---------------------|
| 1Bh | FDC status | FDC command | See also section 13 |
| 5Bh | track register | track | |
| 9Bh | sector regist. | sector register | |
| DBh | data register | data register | |

1Fh Joystick 1 control:
 b0 right drive select
 b1 left side select
 b2 down single/double density
 b3 up ROM bank select
 b4 fire Inhibit switch control
 b5 -- ext. select (?)
 b6 PRN BUSY printer STROBE
 b7 network network

3Bh -- wait when net=1 (*)
 7Bh set boot reset boot see [3.1]
 BBh mem. page in memory page out (**)
 FBh -- printer data
 FEh Joystick 2 scanned as Sinclair joy.

(*) Port 3Bh is used for network synchronization (same as bit 5 of Interface One's port EFh). Any OUT to port 3Bh will halt the Spectrum until the logic level on the network is 0. It is used to wait for the start bit of a transmission frame.
 The network bus carries TTL logic levels (0 = 0 Volts, 1 = 5 Volts).
 The bit rate is 87.5 Kbps and data are exchanged in packets of 256 bytes max using a simple data-link level protocol.

PlusD I/O ports:

| Port | In | Out | Notes |
|------|----------------|-----------------------|---------------------|
| E3h | FDC status | FDC command | See also section 13 |
| EBh | track register | track | |
| F3h | sector regist. | sector register | |
| FBh | data register | data register | |
| EFh | b0/b1 -- | drive select | |
| | b2 -- | single/double density | |
| | b3 -- | ROM bank select | |
| | b4 -- | -- | |
| | b5 -- | ext. select (?) | |
| | b6 -- | printer STROBE | |
| | b7 -- | side select | |
| 7h | mem. page in | memory page out | (**) |
| F7h | b0/b6 -- | printer data (8 bits) | |
| | b7 | PRN BUSY | |

(**) DISCiPLE and +D memories are also paged in whenever the Z80 fetches an instruction from the following addresses:

0x0000, 0x0008, 0x0066, 0x028E.

3.1 DISCiPLE PORT 7Bh AND MEMORY ADDRESSES

Port 7Bh (123 decimal) is available only on the DISCiPLE and has a flip flop attached to it. It can be used to swap the RAM/ROM addresses in this way:

Access ROM RAM Purpose

IN 0x0000 0x2000 reset ff
OUT 0x2000 0x0000 set ff

This feature is used by GDOS to know if it necessary to load the system file from disk on boot or after two consecutive resets without any DOS command between them; UNIDOS ignores this feature, so any swap attempt will result in a system crash.

In GDOS there is a variable located in RAM at offset 0x1DE4 that is set to 0x44 ('D') after a BASIC syntax check (i.e. after a RST 08h with a code lower than 1Bh) and after a bootstrap: this variable indicates that the DOS services have been called almost once. Whenever the user resets the computer, the flip flop attached to port 7Bh is reset, so the ROM will be placed at 0x0000. When the first interrupt occurs, the keyboard scanning routine is called at 0x028E and the DISCiPLE memory is automatically paged in. At offset 0x028E in the DISCiPLE's ROM there's a routine that checks if the variable we said above holds 0x44: if it's the case, then the same routine puts 00h in there to say that DOS services haven't been called since last reset; otherwise the routine sets the variable to 0x53 ('S') and copies the first 2335 (0x091F) bytes of ROM in the RAM: in this case the system file has to be loaded again. When all is finished, the memories will be swapped again (i.e. the flip flop will be set) by OUTing to port 7Bh, the DISCiPLE paged out by OUTing to port BBh and the keyscan routine is finally executed.

INning from port 7Bh has the same meaning of a system reset for the DOS, so after reading 2 times from port 7Bh without typing a DOS command between them the system file needs to be reloaded.

NOTE that since all this is based on the keyscan routine in the Spectrum's ROM, nothing will happen by INning from port 7Bh if the call is not performed (i.e. if interrupts are disabled in IM 1 or we're not in IM 1 or keyboard is scanned in a custom way); however the last operation with port 7Bh must be an OUT before the routine in the ROM is executed if you want to keep the system safe by resetting once.

4. USER FILE INFORMATION AREA (UFIA)

This is a DOS structure often used by the kernel routines and usually pointed to by the register IX.

Offset Len Meaning

0 1 Drive number (1, 2 or '*' (2Ah) for current)
1 1 Program number (in the directory)
2 1 Stream number
3 1 Device density type ('d'=DD, 'D'=SD)
4 1 Directory description (see below)
5 10 File name (padded with spaces)
15 1 File type (see below)
16 2 Length of file
18 2 Start address
20 2 Basic length
22 2 Autostart line

5. SYSTEM CALLS (both DISCiPLE and +D)

To invoke system services you must use the IF1 protocol:

```
RST 8
DB #service
```

All the functions return an error code into register A.

IMPORTANT NOTES:

You cannot perform RST 8 calls from within a routine located into the interface's RAM.

You must not call ROM address 0x028E (keyboard scanning routine) from within an interrupt routine, since this would crash the Spectrum when a DISCiPLE/+D is connected; some programs crash due to this fact (eg SoundTracker v1.1) - you may try to correct the problem replacing the CALL 0x028E with a RST 0x38.

A few programs (like some games converted to disk) do not use the RST 8 mechanism but make absolute CALLs to the DOS or the BIOS routines instead - a very bad practice!

This is the list of GDOS3 (G+DOS2) hook codes with input parameters:

Implemented Interface I hook codes:

```
CONSIN (1Bh) - Console input
CONSOUT (1Ch) - Console output
PRTOUT (1Fh) -
KBDTST (20h) -
SELDRV (21h) - Select drive
OPTMPM (22h) -
CLOSEM (23h) -
ERASE (24h) -
RDSEQ (25h) -
WRREC (26h) -
OPTMPM (2Bh) -
DELBUF (2Ch) -
UNPAGE (31h) - Unpage shadow ROM
CALL (32h) - Call shadow ROM routine
```

GDOS3 (G+DOS2) specific hook codes:

```
HXFER (33h) - transfer file description and header to the DFCA.
    IX = UFIA address
OFSM (34h) - open file sector map with the info in the DFCA.
    The RTP is set to the beginning of the DRAM.
HOFLE (35h) - open a file.
    IX = UFIA
    Combines the previous two functions.
    Sets the last 9 bytes of UFIA with the file header.
SBYT (36h) - Save a byte to DRAM location pointed by RTP.
    A = byte to save.
    If the sector buffer is full, it is automatically saved
    to disk.
HSVBK (37h) - Save a block of data.
    DE = start address of data.
    BC = number of bytes to save.
CFSM (38h) - Close file sector map.
    Flushes DRAM, closes file and updates the directory.
PNTP (39h) - Output a byte to the parallel port.
    A = byte to output.
```

COPS (3Ah) - Copy the screen to printer.

HGFLE (3Bh) - Get a file from disk.
 IX = UFIA
 The first sector is loaded to DRAM and RPT is set to the first byte.

LBYT (3Ch) - Load the byte pointed by RPT.
 Returns A = byte read.
 If needed, another sector is read from the disk.
 RPT is updated consequently.

HLDBK (3Dh) - Load a block of data.
 DE = start address (where the data will be put)
 BC = number of bytes to read

WSAD (3Eh) - Write the DRAM to a sector in the disk.
 D = track
 E = sector
 RPT is restored to the beginning of DRAM.

RSAD (3Fh) - Read a sector to DRAM.
 D = track
 E = sector
 Same as 3Eh.

REST (40h) - Reset drive and seek track 0.
 Drive number is specified into the UFIA.

HERAZ (41h) - Erase the file on disk identified by UFIA.
 IX = UFIA address.

(42h) - Large screen dump

PCAT (43h) - Disk catalogue
 It uses the information in the UFIA. Drive and stream must be set up.
 Offset +0Fh of UFIA may contain one of the following:
 02h for CAT !
 04h for CAT
 12h for CAT ! with a filename mask
 14h for CAT with a filename mask
 The filename must be stored starting at offset +05 of UFIA and may contain wildcards.

HRSAD (44h) - Load sector
 A = drive number
 D = track number
 E = sector number
 IX = Address to load to

HWSAD (45h) - Save sector
 A = drive number
 D = track number
 E = sector number
 IX = Address to save from

(46h) - Open and close streams (how?)

PATCH (47h) - Pages the shadow memory
 Returns HL=0 on DISCiPLE
 HL=1 on PlusD
 HL=2 on DiSCDOS

UNIDOS specific hook codes:

(48h) - Load file
 (49h) - Verify file
 (4Ah) - Merge
 (4Bh) - Save file
 (4Ch) - Open file
 (4Dh) - POINT (see UNIDOS)
 (4Eh) - Flush buffers to disk
 (4Fh) - Close file
 (50h) - Clear channels
 (51h) - Rename file

- (52h) - Move stream
- (53h) - Move file
- (54h) - Select disk and directory

5.1 INTERNAL SYSTEM CALLS (both DISCiPLE and +D)

Here is the purpose of the RST commands when the DISCiPLE or the +D memories are paged in.

- RST 00h - Reset
- RST 08h - Call system services; the required service code must follow (1)(2)
- RST 10h - Call in Spectrum ROM; the routine address must follow (3)
- RST 18h - GDOS, G+DOS : reserved (for syntax check)
 - Uni-DOS : low-level system services; the required service code must follow
- RST 20h - Print DOS error report : the error code must follow
- RST 28h - Performs a RST 20h in the Spectrum ROM
- RST 30h - Gets interpreter status : Z=0 if checking syntax, Z=1 if executing
- RST 38h - Enables interrupts

- (1) = ROM 1 must be paged in Spectrum 128K
- (2) = The interface is automatically paged in by the hardware
- (3) = Be sure that the right Spectrum ROM is paged in

5.2 EXAMPLE: LOADING A FILE.

Here is the simple loader that we have often used in our programs. It will load a CODE file called "blk0.DF" in memory at its original start address. This routine will work fine with DISCiPLE, +D and UNIDOS (all versions). Note: no error checking is done.

```

LOAD: LD  IX,UFIA      ; IX must point to the UFIA
      RST 08 DB 3B     ; HGFLE: open the file
      LD  DE,UFIA+0F   ; the file header will be put here
      LD  B,09         ; the first 9 bytes of the file
L_HDR: RST 08 DB 3C     ; LBYT: get a byte from the DRAM
      LD  (DE),A       ; store the byte
      INC DE
      DJNZ L_HDR       ; fetch all the 9 bytes
      LD  DE,(UFIA+10) ; get the file start address
      LD  BC,(UFIA+12) ; get the file length
      RST 08 DB 3D     ; HLDBK: load the whole block of data
      RET              ; finished!

```

Now the UFIA follows. Only the first 15 bytes must be preset by the user before calling HGFLE.

The last 9 bytes are overwritten with the 9 bytes header of the file.

```

UFIA:  DB 01,         ; drive number (*' for default)
      00,
      00,
      'd',
      04,             ; file type 04 = CODE
      "blk0.DF ",    ; file name (padded with spaces)

```

| | | |
|---------|---------|---------------------------------|
| UFIA+0F | 00, | ; will contain the ROM-ID |
| UFIA+10 | 00, 00, | ; will contain the file address |
| UFIA+12 | 00, 00, | ; will contain the file length |
| | 00, | |
| | 00, | |
| | 00, | |
| | 00 | |

Note that loading address and length are read from the 9 bytes header of the file itself. To force the file to be loaded at a different address, simply change the LD DE,(UFIA+10) instruction (e.g. with a direct LD DE,nn).

6. DISK LAYOUT

The disk has 80 tracks of 10 sectors (512 bytes double density, 256 bytes low density) each, for a total capacity of 800KB (DS/DD).

The first four tracks of the disk (tracks 0-3 side 0) are reserved for the system and contain the disk directory, leaving 780KB available for user data.

The directory consists of 80 consecutive file descriptors, each one taking 256 bytes; thus, the descriptor of file #48 resides in the first 256 bytes of sector 4 track 2.

The directory has a fixed dimension and can only contain up to 80 files.

UNIDOS overcomes this limitation introducing subdirectories and allowing to specify the maximum number of file entries for each directory.

Disks formatted with DISCiPLE or +D can be read and written by common PC disk drives and viceversa.

7. FILE DESCRIPTOR FORMAT

Now we will see the details about a single directory entry.

NOTE: All numbers are in decimal.

General structure:

OFFSET MEANING

- 0 File type (see FILE-TYPES table); 0 = erased (free entry)
- 1-10 Filename (padded with spaces)
- 11-12 Number of sectors occupied by the file (in Motorola byte order)
- 13 Track number of the first sector of the file
- 14 Sector number of the first sector of the file
- 15-209 Sector allocation bitmap. Each bit corresponds to a disk sector.

A bit is set if the corresponding sector belong to the file.

Examples: byte 15, bit 0 corresponds to track 4, sector 1;

byte 16, bit 3 means track 5, sector 2...

IMPORTANT NOTE:

The s.a.b. is used only during saving operations: the s.a.b. of all the 80 files are merged together (OR) so that the system knows which sectors are free (not allocated to any file).

During loading a faster method is used: each sector contains only 510 bytes of data; the last two bytes contain the track number and the sector number of the next sector of the file, respectively. The last sector of the chain contains (0,0) as the

last two bytes.

210-255 Depend on the file type.

BASIC (type 1)

211 Always 0 (this is the id used in tape header)
212-213 Length
214-215 Memory start address (PROG when loading - usually 23755)
216-217 Length without variables
218-219 Autostart line
NOTE: These 9 bytes are also the first 9 bytes of the file.

NUMBER ARRAY (type 2)

211 Always 1 (this is the id used in taper header)
212-213 Length
214-315 Memory start address.
216-217 Array name, probably ignored.
218-219 Not used
NOTE: These 9 bytes are also the first 9 bytes of the file.

STRING ARRAY (type 3)

211 Always 2 (this is the id used in tape header)
212-219 Same as for type 2
NOTE: These 9 bytes are also the first 9 bytes of the file.

CODE FILE (type 4)

211 Always 3 (this is the id used in tape header)
212-213 Length
214-315 Start address
216-217 Not used
218-219 Autorun address (0 if there is no autorun address)

48K SNAPSHOT (type 5)

211-219 Not used
220-255 Z80 registers (in words) in the following order:

IY IX DE' BC' HL' AF' DE BC HL I SP (see below for R and AF)

Register I is in the MSB of the corresponding word (byte offset 239), so that it is loaded with:

```
POP AF
LD I,A
```

The Interrupt Mode is desumed by the value of the I register: if it contains 00h or 3Fh then IM1 is assumed, else IM2 is set. The IFF2 status (IFF1=IFF2) is retrieved from the P/V bit of the flag register F.

SP is actually SP-6, because the original stack is "corrupted" with the following 6 bytes (in ascending order):

```
R AF PC ( ----> decreasing stack )
| |
SP SP+6 (original SP)
```

(R is in the MSB of the corresponding word) so that the return code could be something like this (actually it is a bit more complex):

```
POP AF
LD R,A
POP AF
RET
```

MDRV (type 6)

This is a microdrive cartridge image. Details omitted.
NOTE: UNIDOS mdrv files are completely different from GDOS ones.

SCREEN\$ (type 7)

Same as type 4 with Start=16384 and Length=6912

SPECIAL (type 8)

211-255 Any meaning assigned by the programmer.

128K SNAPSHOT (type 9)

Same as 48K Snapshot. The first byte of the file is a copy of the page register (port 0x7FFD), usually held in the system variable BANKM (23388). The 8 RAM pages are saved in ascending order from 0 to 7.

OPENTYPE (type 10)

210 Number of 64K blocks in the file
211 Always 9 (not sure)
212-213 Length of the last block
214-255 Not used

NOTE: Opentype files can be more than 64K in length and are usually created and handled with the stream-related BASIC statements, such as OPEN #, CLOSE #, PRINT #, INPUT # and so on.
See chapter [9] for a brief description of these statements.

EXECUTE (type 11)

210-255 Same as CODE file (type 4), but Length=510 and Start=0x1BD6 implicitly (0x3DB6 for +D). The sector is loaded into the interface RAM and executed (it should contain relocatable code!).

SUBDIRECTORY (type 12) - UNIDOS

210-212 Same as Opentype (type 10). This file is always held on contiguous sectors. The last two bytes of a sector do not contain the address of the next sector. The structure is the same as the root directory, but the first entry contains the file header number of the parent directory. The last two bytes of the last sector contain 0xFFFF.
213 Capacity (number of file entries allowed).

CREATE (type 13) - UNIDOS

210-255 Same as CODE file but the start address is ignored.

8. FILE-TYPES TABLE

This table lists the various MGT file types with their correspondent ID's. Where possible, it is also reported the file type ID used by the standard Spectrum ROM in the tape header. The ROM-ID is part of the 9 bytes header of the file (same as bytes 211-219 of the directory entry).

NOTE: under UNIDOS, add 128 for hidden files and 64 for protected files.

| Code | Type | CAT string | ROM-ID |
|-------------|---------------------|------------|--------|
| 0 | ERASED (free entry) | (NA) | NA |
| 1 | BASIC | BAS | 0 |
| 2 | NUMBER ARRAY | D.ARRAY | 1 |
| 3 | STRING ARRAY | \$.ARRAY | 2 |
| 4 | CODE | CDE | 3 |
| 5 | 48K SNAPSHOT | SNP 48k | NA |
| 6 | MICRODRIVE | MD.FILE | NA |
| 7 | SCREEN\$ | SCREEN\$ | NA |
| 8 | SPECIAL | SPECIAL | NA |
| 9 | 128K SNAPSHOT | SNP 128k | NA |
| 10 | OPENTYPE | OPENTYPE | NA |
| 11 | EXECUTE | EXECUTE | NA |
| 12 (UNIDOS) | SUBDIRECTORY | DIR | NA |
| 13 (UNIDOS) | CREATE | CREATE | NA |

9. GDOS EXTENDED BASIC SYNTAX

GDOS extends the BASIC to provide support for disk operations.

When you switch the Spectrum on, you must initialize the system; to do so, insert a disk containing the operating system file ("SYS*" on DISCiPLE and "+SYS*" for +D) into the first drive and then enter "RUN".

The DOS also looks for the first file called "auto*" and runs it if found.

The "auto*" file is not searched for if the command "RUN boot" is entered.

NOTE: File names are case insensitive and may contain wildcards (* and ?).

To show the disk contents, enter:

```
CAT 1 (for drive 1) or
CAT * (current drive)
```

You may also enter commands like these:

```
CAT 1;"a*"
CAT #3;1;"sys*"
CAT 1!
CAT #3;1!
```

so you can redirect the CAT output to any channel and you may specify a file name which may contain wildcards to show only matching files.

If the command ends with a '!' then an abbreviated catalogue is shown, containing just a list of (matching) file names.

To LOAD/SAVE a file:

```
LOAD d<dn>;"filename" [CODE|SCREEN$|DATA|etc]
LOAD p<fn>
SAVE d<dn>;"program" [CODE|LINE|SCREEN$|etc]
```

where <dn> is a drive number (1-2)

<fn> is a file number (1-80)

You must specify 'S' to load a 48K snapshot and 'K' for a 128K snapshot.
Case of letters 'S' and 'K' is important.

Examples:

```
LOAD d1;"screen" SCREEN$
LOAD d*;"pippo"      loads program pippo from the current drive
LOAD d1;"snap128"K   loads the 128K snapshot 'snap128'
SAVE d2;"rom" CODE 0,16384 this saves the DISCiPLE memory!
```

Note: "d1" with lowercase 'd' refers to DS DD disks (80 tracks double sided);
in ROM version 3 the only difference seems to be that if you use 'D' a CAT
command is also performed after the operation.

Early versions of the OS used "D1" (capital 'D') for single sided disks.

This applies always when you have to specify the 'd<dn>' field.

Please use only DS/DD disks.

When saving CODE files, an autostart address can be specified as a third
parameter:

```
SAVE d*;"runme" CODE 32768, 8192, 33000
```

so when you load it back with LOAD CODE, it will be automatically launched
with an implicit RANDOMIZE USR 33000.

Each file can be referred both through its name and its directory number, so
if file "screen" is listed as number 7 you may also enter:

```
LOAD p7
```

Note that if you use the abbreviated notation, each file will be loaded
accordingly to its type (i.e. you can a CODE file will be loaded into memory
at its start address).

Of course MERGE and VERIFY are also available with a similar syntax.

To erase a file from the disk, enter:

```
ERASE d1;"file2del"
ERASE d1;"*"      dangerous!
```

The ERASE command can also be used to rename a file:

```
ERASE d1;"oldname" TO "newname"
```

To format a disk, use:

```
FORMAT d1      double density (250 Kbit/sec)
```

or

```
FORMAT sd1     single density
```

GDOS also extends streams, so that you can redirect a stream to a file and
vice versa. If you open a channel for writing to a disk file, then an OPENTYPE
file is created. Opentypes can be more than 64K in length.

Channels are accessed with the usual PRINT, INPUT, INKEY\$, etc. commands.

Examples:

```
OPEN #4;d1;"archive"
OPEN #5;d1;"temp" OUT  open for writing only
```

```
OPEN #5;d1;"temp" IN      open for reading
MOVE #3 TO #4
MOVE #4 TO d1;"temp1"
MOVE d1;"temp2" TO #4
CLOSE #4
```

Note that disk-mapped channels are buffered, so data is read/written to disk only when the 512 bytes buffer is empty/full.
When accessing BASIC, CODE, DATA and SCREEN\$ files through streams, remember that these files start with a 9 bytes header and the actual data starts at byte 10.

To copy a file into another, use:

```
SAVE d1;"file1" TO d2;"file2"
```

Since the SAVE TO command uses all the RAM available, when it has finished a system reset occurs.

You can LOAD/SAVE single disk sectors with this syntax:

```
LOAD @d,t,s,address
SAVE @d,t,s,address
```

where 'd' is a drive number (1 or 2)
't' is the track number (0..79 + 128 if side 1)
's' is the sector number (1..10)
'address' is the address of the 512 bytes buffer

For example, you may read the first sector of the disk (which holds file descriptors number 0 and 1) with `LOAD @1,0,1,40000`.

DISCiPLE GDOS recognizes the Microdrive syntax, so you can enter commands like this:

```
LOAD *"m";1;"pippo"
```

which will load the BASIC program pippo from drive 1. All your microdrive programs should run over GDOS without modifications. Remember that PlusD does not support the IF1 syntax.

9.1 THE NETWORK

GDOS (DISCiPLE) implements an IF1 compatible network, with some enhancements. Up to 63 Spectrums can be connected together and share their resources (files and printers) simply through a 3.5 mm mono jack cable. Each station is given an unique station number ranging from 1 to 63. The station number is assigned with the command:

```
FORMAT Ns
```

where 's' is the station number (1-63). This command can be entered even without having the system file loaded (i.e. system not booted).

Station number 0 is reserved for broadcasting. If you enter `FORMAT N0`, the network is switched off; type `FORMAT Ns` to switch it on again.

Station numbers are divided into four classes:

Number Purpose

0 Broadcasting
1 Master station
2-9 Assistants (they load the system file)
10-63 Pupils or assistants (pupils do not load the system file)

The following network configurations are possible:

1. Shared Access Network

In this model, station number 1 owns the resources (disk drives and printer) and acts as a master. The other stations are called pupils and can access files which are onto the master's disks and print with the master's printer. Only the master must load the system file. Since they don't have to load the system file, pupils are identified with station numbers greater or equal than 10. So, if you try to enter `FORMAT N8` without the system loaded, you get an 'Invalid station number' error message.

The master can send a file "pippo" to pupil number 16 with:

```
LOAD d1;"pippo"    first load the file, a Basic program in this case
SAVE N16           send it to station 16
```

Of course you can send `CODE` files, `SCREENS`, etc. too!
You can send the file to all the stations using broadcast:

```
SAVE N0           send to all
```

Station 6 can receive the program from the master entering:

```
LOAD N1           receive from station 1 (the master)
```

Each station can receive a broadcast message with:

```
LOAD N0
```

Pupils can also send broadcast messages with `SAVE N0` and communicate with other pupils in the same way.

The most interesting thing is that a pupil can enter disk and printing commands as if the disk drives and the printer would be connected to it. So, station number 6 can get the master's disk directory simply with:

```
CAT d1
```

and load file "pippo" with:

```
LOAD d1;"pippo"
```

Of course pupils can also save files onto the master's disk, and use its printer with the common

```
LLIST, LPRINT or COPY SCREENS
```

commands.

The master has the ability to force data transfer from and to a pupil station. On the master, if you type:

```
LOAD F4 SCREENS
```

then the Spectrum number 4 will stop and transfer its current SCREEN\$ to the master, which can therefore see what's going on at station 4. Similarly, you can also enter:

```
SAVE F8
```

which will force station number 8 to load the Basic program currently loaded onto the master.

Again, any variation of the LOAD/SAVE commands will work (CODE, SCREEN\$...).

Of course the master can use the system normally.

2. Independent Station Network

It is very similar to the previous model, but now each station has its own disk drives and printer. Of course all the stations must load the system file in this case, and the FORMAT command must be issued after that.

For example, station 4 can send a CODE block to station 3 with:

```
SAVE N3 CODE 32768, 16384
```

and station 3 will receive it with:

```
LOAD N4 CODE
```

You can use station number 0 for broadcasting.

Note that the master station in the previous model is an independent station.

Important note: only stations number 1 - 10 can be independent stations.

Stations 2 - 10 are called assistants (act like masters but cannot enter SAVE F- and LOAD F- commands).

3. Mixed Network

It is possible to have a network with both pupil and independent stations - the pupil stations being those without their own disk drives or printers.

In this way it is possible to have more than a master.

9.2 THE SNAPSHOT BUTTON

When you press the magic button, an NMI is generated and control passes to address 0x0066 (102 dec.) of the DISCiPLE/+D memory. The consequent behaviour of the system depends on the particular System version loaded. Under the standard systems (eg. systems 3a/3b/3d), you have to hold down CAPS SHIFT while pressing the button and then colored stripes appear in the border and five keys are active:

- 1 = Dump screen to printer
- 2 = Big screen dump (A4)
- 3 = Save current screen to disk
- 4 = Save a 48K snapshot
- 5 = Save a 128K snapshot.

Snapshots are saved to disk with names like "Snap A" and subsequent indexes depending on their position in the directory. When saving a 128K snap, the

system stops after creating the file and waits for the user to specify if the screen has changed since the beginning of the operation; the user must respond pressing either 'y' or 'n'. This happens because unlike Multiface the DISCiPLE and the +D haven't got any flip flop to store the status of port 0x7FFD bit 3 which tells which videoram is displayed. Before waiting for the user intervention, the system pages the first videoram (0x4000 page 5), so if the image displayed changes then it means that the second videoram was previously paged. Well, that's it!

After that, control is passed back to the interrupted program.

NOTE: The snapshot routine corrupts the stack with six bytes (PC,AF and R+F). This may cause some programs which use the stack in a particular way to crash if the magic button is pressed at certain times (eg. Batman the Movie). See the snapshot section (file type 5) in chapter [7] for more details.

NOTE: The snapshot feature cannot be used while the network is in use.

9.3 GDOS and UNIDOS ERROR MESSAGES

Here's a list of the error codes for both systems available for the RST 20h service.

| Code | GDOS | G+DOS |
|------|--------------------------|--------------------------|
| 00 | Nonsense in GDOS | Nonsense in G+DOS |
| 01 | Nonsense in GNOS | Nonsense in GNOS |
| 02 | Statement end error | Statement END error |
| 03 | BREAK requested | BREAK requested |
| 04 | SECTOR error | SECTOR error |
| 05 | FORMAT data lost | FORMAT data lost |
| 06 | NO DISC in drive | CHECK DISC in drive |
| 07 | No "SYSTEM" file | NO "+SYS " file |
| 08 | Invalid FILE NAME | Invalid FILE NAME |
| 09 | Invalid STATION | Invalid STATION |
| 10 | Invalid DEVICE | Invalid DEVICE |
| 11 | VARIABLE not found | VARIABLE not found |
| 12 | VERIFY failed | VERIFY failed |
| 13 | Wrong FILE type | Wrong FILE type |
| 14 | MERGE error | MERGE error |
| 15 | CODE error | CODE error |
| 16 | PUPIL set | PUPIL set |
| 17 | Invalid CODE | Invalid CODE |
| 18 | Reading a WRITE file | Reading a WRITE file |
| 19 | Writing a READ file | Writing a READ file |
| 20 | O.K. GDOS 3 | O.K. G+DOS |
| 21 | Network OFF | Network OFF |
| 22 | Wrong DRIVE | Wrong DRIVE |
| 23 | Disc write PROTECTED | Disc write PROTECTED |
| 24 | Not enough SPACE on disc | Not enough SPACE on disc |
| 25 | Directory FULL | Directory FULL |
| 26 | File NOT FOUND | File NOT FOUND |
| 27 | END of file | END of file |
| 28 | File NAME used | File NAME used |
| 29 | Not a MASTER station | NO G+DOS loaded |
| 30 | STREAM used | STREAM used |
| 31 | CHANNEL used | CHANNEL used |

| Code | UNIDOS |
|------|------------------------|
| 128 | Nonsense in Uni-Dos |
| 129 | O.K Uni-Dos |
| 130 | Break requested |
| 131 | Corrupt sector |
| 132 | Sector missing |
| 133 | Check disc in drive |
| 134 | DOS file not found |
| 135 | Invalid filename |
| 136 | Invalid sector number |
| 137 | Invalid device/channel |
| 138 | Wrong stream type |
| 139 | Verification failed |
| 140 | Wrong file type |
| 141 | CODE parameter error |
| 142 | Directory not found |
| 143 | File has zero length |
| 144 | Reading a write file |
| 145 | Writing a read file |
| 146 | POINT outside file |
| 147 | Channel out of order |
| 148 | Illegal drive number |
| 149 | Disc write protected |
| 150 | Not enough disc space |
| 151 | Directory full |
| 152 | File not found |
| 153 | End of file |
| 154 | Filename already used |
| 155 | File still open |
| 156 | File in use |
| 157 | Channel already open |
| 158 | Protected file |
| 159 | Unavailable RST 8 |

10. GDOS and UNIDOS SYSTEM VARIABLES

GDOS and UNIDOS system variables are stored into the interface's RAM and can be modified with the POKE command in the following form:

POKE @var, value

where:

var is a variable number
 value is the new variable value.

Actually, the '@' operator is interpreted as an offset into the interface's RAM. The base addresses are the following:

0x2000 UNIDOS and G+DOS (+D)
 0x0298 GDOS (DISCiPLE)

List of the system variables:

VAR# Description

- 0 Flash border during disk operations. Set to 0 to leave the border unaltered. This byte is ANDed with the sector number currently accessed and then sent to port 0xFE.
- 1 Drive 1 capacity = number of tracks + 128 if double sided.
- 2 Drive 2 capacity, same as above.
- 3 Drive stepping rate. Set to 1 for the minimum (1ms).
WARNING: Poking 0 here may lock your disk drive. It can be unlocked reinstalling the OS from tape.
- 4 GDOS: disable Centronics printer port (0=enabled)
UNIDOS: Enable BREAK key if set.
- 5 Printer line length in number of characters (default 80)
- 6 Printer control flag. If set the codes sent to the printer are not filtered (binary output). Necessary to send control codes to the printer.
- 7 Printer line spacing expressed in n/72 of an inch. It is sent to the printer before every CR (default GDOS=12, UNIDOS=8).
- 8 Number of line feeds after CR (default: GDOS=1 UNIDOS=0).
- 9 Left margin for printing. This is the number of spaces inserted before the first character of a line (default=0).
- 10 Printer flag.
GDOS: if set then the printer driver generates the graphic representation of 'œ' and ',' (default=1).
UNIDOS: printer flag (default=0x80).
- 11 GDOS: network station number (default=1).
UNIDOS: Centronics enable, same as GDOS variable #4 (default=1).
- 12 UNIDOS: printer column number (default=1).
- 13 UNIDOS: CLS# screen color.
- 14 Extended syntax address (2 bytes). This address is called on error which are not related to hook codes and DOS syntax. Can be used to add extra commands. Ignored if 0.
- 16 UNIDOS: Interrupt address (2 bytes).
- 18 UNIDOS: Printer initialization codes (8 bytes). They are sent to the printer after a NEW or before pressing the 'P' key during a snapshot.
- 26 UNIDOS: Set character pitch (8 bytes).
- 34 UNIDOS: Set n/72 line space (8 bytes).
- 42 UNIDOS: Set UDG bit graphics density (8 bytes).
- 50 UNIDOS: Second initialise codes (8 bytes).
- 58 UNIDOS: Codes for 'œ' (8 bytes).
- 66 UNIDOS: Codes for '#' (8 bytes).
- 74 UNIDOS: Codes for (C) (8 bytes).
- 82 UNIDOS: Save SCREEN\$ 2 parameters (7 bytes).
- 89 UNIDOS: set dump graphics (8 bytes).
- 97 UNIDOS: address of extra error messages (default=0x1C68).
- 99 UNIDOS: error code
- 100 UNIDOS: address of LPRINT routine (default=0x34AA).
- 102 UNIDOS: DOS error return address (default=0x0000).
- 104 UNIDOS: snapshot workspace (20 bytes).
- 124 UNIDOS: called on reset (default=0x0000).
- 126 UNIDOS: called on boot (default=0x21A4).
- 7667 UNIDOS: set this to 0 to reset DOS.

11. UNIDOS EXTENDED BASIC SYNTAX

UNIDOS is an advanced operating system which runs both on DISCiPLE and PlusD and is available as an EPROM upgrade separately for the two interfaces.

To install it, get the EPROM chip specific for your interface and replace the GDOS ROM with it. Of course you also need the system disk which contains the RAM resident portion of the DOS, a file called "Uni-Dos" which is launched with the usual RUN command when the computer is switched on.

The system file is exactly the same both for the DISCiPLE and the +D (hence the name UNI-DOS, I suppose).

It is 6654 bytes in length; the "missing" two bytes contain a checksum which is constantly monitored by the DOS to detect system corruptions.

This mechanism works quite well and sometimes you'll see that the system file is loaded without an explicit statement: just don't worry! :)

Also remember that in the DISCiPLE the RAM and ROM addresses are swapped under UNIDOS. See chapters [2] and [3.1] for all the details.

The main UNIDOS features are powerful disk management with subdirectories, excellent printing facilities and a lot more of professional touches that you discover with use, making it extremely powerful.

UNIDOS provides a superset of the standard GDOS and G+DOS functions (some of them have been changed, however), so read the related chapter first if you don't know that already. Some parts are missing, such as the network routines which have been suppressed for space reasons. Also the FORMAT statement is no more part of the command set, but it is provided as a separate program in the system disk.

First of all, files now have two attributes: hidden and protected. Hidden files are not showed in the catalogue, while protected files are read-only. Also, the disk is given a name (a string containing up to 10 characters) during the FORMAT process.

As already said, UNIDOS introduces SUBDIRECTORIES. They work in the same way as you'd expect, so whenever a filename has to be specified you may use a complete path:

```
"/dir1/dir2/file"
```

Note that the slash '/' is used to separate the directory names, just like in Unix. Pathnames can be relative to the current directory or absolute (i.e. relative to the root, starting with a '/'). There is no limit to the depth of the directory tree. The root directory has a fixed dimension of 80 file entries, while subdirectories may have any capacity. If the string ends with the slash character, then it is a directory name. The special directory names "." and ".." obviously refer to the current and the parent directory, respectively.

The equivalent of the 'cd' (change directory) command is:

```
IN d1;"pathname/"
```

with the ending slash in the pathname (because it must be a directory name); this sets the current working directory and the current drive. You may also use IN to set the current drive only, just omit the pathname and the semicolon.

There are no standard commands to create and remove directories. See the CREATE files paragraph later.

The LOAD and SAVE statements are unchanged; an extra abbreviated form has been introduced for LOAD (and MERGE and VERIFY too, of course):

```
LOAD p"filename"
```

which loads "filename" from the current disk.
EXECUTE files can be launched with:

```
LOAD d1;"exe"X, <address>
```

In the SAVE statement you may specify the OVER keyword to avoid the overwrite check:

```
SAVE OVER d1;"program"
```

This will overwrite a previous file without asking for confirmation.

The ERASE command is unchanged. If NOT is added after the ERASE statement, then no error is reported. You cannot remove a file if it is currently open or it is protected.

The CAT statement has a new form:

```
CAT d1  
CAT d1;"pathname"
```

You must now specify the 'd1' instead of just '1' as in GDOS. All the GDOS variants are accepted.

If you add NOT after CAT, then it will list the hidden files too.

If you specify a directory name, CAT lists the directory contents.

Now it possible to MERGE CODE files too, but the only effect is that the autostart is removed.

The MOVE statement works as usual (OVER allowed) and it can be now used to copy one file to another, replacing the old SAVE TO in this way:

```
MOVE [OVER] d1;"file1" TO d2;"file2"  
MOVE [OVER] d1;"file1" TO "file2"
```

MOVE can copy snapshots, MDRV and opentype files too, even longer than 64K. It can copy entire subdirectories if a directory name (ending with '/') is specified.

Unlike SAVE TO, MOVE uses only the memory between the BASIC area and the machine stack, so it does not require a system reset when it has finished.

The CLS command can be used to reset the screen colors to those stored in the system variable 13:

```
CLS #
```

A great facility UNIDOS introduces are RANDOM ACCESS FILES, i.e. OPENTYPE files which can be accessed in random way, not only serially one byte after another.

To open a random file, the OPEN # statement is used in this form:

```
OPEN #4;d1;"file" RND
```

If you specify IN instead of RND, the file is read only but still has random access. Up to 16 channels can be attached to the same file at once.

This statement:

```
OPEN #4;d1;"file" RND <length>[,<byte>]
```

creates the file of the specified length and fills it with the specified byte (if specified!).

Remember to close or clear all channels before you remove the disk from the drive, or else an error will be given when you enter DOS commands after the disk swapping.

You can work with channels using the usual PRINT, INPUT and INKEY\$ commands with all the respective variants. However, some new commands are available:

```
POINT #4, <offset>
```

will set the file pointer at the specified offset of the random access channel #4. An error is given if the file boundaries are crossed.

The CLEAR statement has now two forms:

```
CLEAR #  
CLEAR #* [<channel>]
```

The former clears all channels without creating an openout file, the latter closes one or all the disk channels creating an openout file and flushing the buffers.

You may flush the buffers only, without closing the file, with:

```
OUT #<channel>
```

As we said above, a special care has been involved in the PRINTING section of UNIDOS. The LPRINT and LLIST commands work in the usual way, now redirected to the parallel port of the interface.

You can print a screen dump with:

```
SAVE SCREEN$ 1 or SAVE SCREEN$ 2
```

each using a different preset of parameters. The second preset can be user defined, just alter the relative system variables. These are also the routines invoked by the snapshot keys '1' and '2' (see later).

The most general and powerful form of SAVE SCREEN\$ is the following:

```
SAVE SCREEN$ #flag [,pass [,margin [,y [,x [,h [,w]]]]]]]
```

where flag = horiz. magnification (0-7) + 8 * vert. magnification (0-7) +
+ 64 if you want color processing + 128 if you want sideways
printing.

A magnification of 0 actually means 8.

pass = number of passes for a single printing line

margin = left margin in characters

y & x = top left corner of the window to print

w & h = width and height of the window to print

See the system variable list for more advanced settings which affect printing.

An useful statement implements the ON ERROR GOTO mechanism:

```
LINE 9000
```

enables error trapping; all errors but OK, STOP and BREAK are caught and control is passed to the specified line, where the program can identify the error type reading system variable 99 with (PEEK @99).

UNIDOS also implements new BASIC FUNCTIONS; they are all surrounded by brackets, i.e. are in the form of (<function>).

We have just seen the:

```
(PEEK @offset)
```

which reads a system variable (and in general the RAM location offset+8192).

The length of a random channel is obtained with:

```
(LEN #<channel>)
```

while:

```
(POINT #<channel>)
```

returns the current file pointer (offset) or 0 if we are at the end of the file (EOF).

A specified number of bytes can be read from any channel with:

```
(IN #<channel>, <length>)
```

Lastly, you can check if a file or directory exists with:

```
AT d1;"pathname"
```

which returns 0 if the file does not exist or its directory number if found.

Now let's examine the last new file type introduced, CREATE files.

These are machine code programs (generally new commands or syntax extensions) that are loaded between the channel area and the BASIC area, so they are lost after a NEW or a reset.

You may LOAD and SAVE CREATE files with:

```
LOAD d1;"pathname" USR  
SAVE d1;"pathname" USR <address>, <length>
```

You can load as many create files as you need (but the memory space is obviously affected).

The system disk provided with UNIDOS contains two create files with the following extensions:

"ext_code" contains:

```
FORMAT d1;"diskname"  
to format disks. The disk name is stored in the last bytes of  
the first root directory entry.  
(LINE)  
returns the current drive number + 128 if the disk is write  
protected and the value is negated if there is no disk in drive.  
(STR$ #<channel>)  
return a null string if the channel is not open, 'd' if it is an  
openout stream and 'D' if it is a random stream, or the letter  
that the channel uses.
```

"dir_code" contains:

```
SAVE [OVER] d1;"dirname" CAT <capacity>  
creates a directory with a capacity of the specified number of  
files.  
ERASE d1;"dirname" CAT  
removes the whole specified directory.  
(STEP [d<drive>])  
returns the pathname of the last directory accessed or of the  
current directory (if the drive number is specified).
```

Now a few words about the SNAPSHOT. The active keys are the same as in GDOS with the same meanings, plus the key 'x' to return to the interrupted program and 'p' which sends the initialization codes to the printer.

If some error occurs during the snapshot saving, then control is passed back to the snapshot menu and not to the interrupted program.

Note that on the +D the snapshot button is disabled during the processing of a DOS command.

12. CONNECTORS PINOUTS

The following pinouts are viewed from the back of the interface.

DISC CONNECTOR

```
----- 1-33 Ground (0V) 22 Write data
| 33          3 1 | 8 Index    24 Write gate
| o o o o o o o o o o o o o o o o | 10 Disk1 select 26 Track00
|          | 12 Disk2 select 28 Write protect
| o o o o o o o o o o o o o o o o | 16 Motor on   30 Read data
| 34          4 2 | 18 Step dir. 32 Side select
----- 20 Step pulse
```

PRINTER CONNECTOR

```
----- 1 Strobe    13 D5
| 25          3 1 | 3 D0      15 D6
| o o o o o o o o o o o o o o o o | 5 D1      17 D7
|          | 7 D2      21 Busy (input)
| o o o o o o o o o o o o o o o o | 9 D3      2-22 Grounds (0V)
| 26          4 2 | 11 D4
-----
```

The parallel pinout allows direct connection to a Centronics connector simply through a flat cable.

12.1 CONNECTING A PC 5.25" DRIVE

Petri Andras sent an interesting document to explain how to connect a common 5.25" (1.2 MB) PC disk drive to the DISCiPLE and +D. Although not all drives are suitable for the purpose, the chances to succeed are quite good; you can decide whether your drive will work or not with a simple test described below.

Note that the procedure requires a certain experience, so you'd better ask an expert friend if you are not familiar with such operations. Take care, we are not responsible for any damage caused to your system.

SUITABILITY TEST. The floppy drive must be able to operate in "low density" mode. This is the crucial point, as the DISCiPLE expects a specific RPM from the floppy drive; the PC floppy controller, however, handles different densities without changing the floppy drive's RPM, by re-programming its internal clock frequency. Therefore many PC floppy units, especially later ones, do not support RPM changing at all. There is a line of the interface for switching the floppy's RPM (Pin 2), but the floppy unit may ignore it.

This is the operative test: apply power to the floppy with the flat cable removed, insert a disk into it, move the jumper to DS0 as described above, short-circuit pins 11-12 and 15-16 (with two small alligator clamps on the edge connector). Now the floppy LED will light on and you can hear the drive motor running. Take a third alligator clamp or a piece of wire, and short pins 1 and 2. If the RPM of the disk changes (the difference is audible), then the floppy unit is OK.

If no change occurs, remove power, get a multimeter and search for a jumper that is connected to pin 2 of the floppy's edge connector. If you are lucky, you will find one; try to set it into a different position and test again.

THE PROCEDURE. The floppy drive must be connected to the edge connector on the PC floppy cable that is BEFORE the twist (where drive B: is connected in a real PC). If the floppy cable has an old-style 5.25" edge connector only after the twist (newer PC floppy cables), the edge connector must be disassembled, the twisted wires of the flat cable must be straightened, and the connector must be re-assembled to the cable. This operation requires a little dexterity, but a real Speccy hacker surely has it ;-)

The floppy drive has a set of four jumpers somewhere, described as DS0, DS1, DS2, DS3 (or something similar, numbered from 0 to 3). The jumper is ALWAYS in the DS1 position in the case of PC floppies. This must be moved to DS0 if you want to use it as Disk 1 on the DISCiPLE. (Disk 2 must have this jumper on DS1).

Technical note: This jumper actually determines the Drive Select line that activates the floppy unit. The original Tandon/Shugart floppy interface supported four Drive Select lines and a common Motor Enable line. The PC floppy interface uses DS1 and MotorEnable for disk selection and motor enable for drive A:, and DS0 and DS2 for drive B:. The twist in the floppy cable ensures that the two floppy units can be jumpered identically.

13. VL1772 PROGRAMMING INFO

Here is the low-level technical information about the VL1772 floppy disk controller. Some devices equipped with this component are MGT DISCiPLE, MGT PLUS D, MGT SAM COUPE and ATARI ST. It seems to be almost compatible with other FDC devices such as 1791 and 1793.

See section 3 for the port addresses of the FDC registers in the DISCiPLE and the +D.

----- -COMMAND REGISTER (W)- -----

Commands are divided into four classes. The lower 4 bits of the command byte have a different meaning depending on the command class; remember to OR them with the command codes given below.

Type 1 commands:

b0-b1 = Stepping rate
00 = 6 ms
01 = 12 ms
10 = 20 ms
11 = 30 ms
b2 = Verify track
b3 = Load/unload head at beginning

Command name Code Comments

RESTORE 0x00 Restore disk head to track 0
SEEK 0x10 Seek a track (send the track number to the DATA reg.)

STEP_NUPD 0x20 Step using current dir without updating track register
 STEP_UPD 0x30 Step drive using current direction
 STEP_IN_NUPD 0x40 Increase track without updating track register
 STEP_IN_UPD 0x50 Increase track
 STEP_OUT_NUPD 0x60 Decrease track without updating track register
 STEP_OUT_UPD 0x70 Decrease track

Type 2 commands:

b0 = f8 (deleted dam) / fb (dam) if set in READ commands
 b1 = Enable side compare
 b2 = 15 ms delay
 b3 = Compare for side 1/0

Command name Code Comments

READ_1SECTOR 0x80 Read one sector
 READ_MSECTOR 0x90 Read multiple sectors
 WRITE_1SECTOR 0xA0 Write one sector
 WRITE_MSECTOR 0xB0 Write multiple sectors

Type 3 commands:

b0-b1 = 0
 b2 = 15 ms delay
 b3 = 0

Command name Code Notes

READ_ADDRESS 0xC0 Read address
 The controller reads the header of the first sector encountered and produces 6 bytes which must be read: track number, side number, sector number, sector size and a two-byte checksum.
 READ_TRACK 0xE0 Read a whole track (including headers and control data)
 WRITE_TRACK 0xF0 Write a whole track (used to format the track)

Type 4 commands:

b0 = Not ready to read transition
 b1 = Ready to not read transition
 b2 = Index pulse
 b3 = Immediate interrupt, requires reset
 b0-b3 = 0000 -> Terminate with no interrupt

Command name Code Notes

FORCE_INTERPT 0xD0 Force interrupt (stops the current command)

-STATUS REGISTER (R)-

Some bits assume a different meaning depending on the last command issued.

After a Type 1 command:

| Bit | Meaning | Comments |
|-----|---------|----------|
|-----|---------|----------|

| | | |
|---|---------------|--------------------------------|
| 0 | BUSY | Wait BUSY=0 for a new command |
| 1 | INDEX PULSE | Index pulse |
| 2 | TRACK00 | Signals head on track 00 |
| 3 | CRC ERROR | Sector corrupted |
| 4 | SEEK ERROR | Seek error |
| 5 | HEAD LOADED | Head loaded |
| 6 | WRITE PROTECT | Disk is write protected |
| 7 | MOTOR ON | Motor is on or drive not ready |

After a Type 2/3 command:

| Bit | Meaning | Comments |
|-----|------------------|---|
| 0 | BUSY | Wait BUSY=0 for a new command |
| 1 | DRQ | Need to send or read data from DATA register |
| 2 | LOST DATA | Error (eg you did not respect I/O timings) |
| 3 | CRC ERROR | Sector corrupted |
| 4 | RECORD NOT FOUND | Non-existent track/sector or no more data to read |
| 5 | REC.TYP/WR.FAULT | Read: record type; Write: write fault |
| 6 | WRITE PROTECT | Disk is write protected |
| 7 | MOTOR ON | Motor is on or drive not ready |

-TRACK REGISTER (RW)-

Contains the current track number.

-SECTOR REGISTER (RW)-

Current sector number for read/write operations.

-DATA REGISTER (RW)-

Here you may read and write the data to the controller. Check the status register before reading or writing data.

14. CREDITS AND CONTACT INFO

This document was written by Luca Bisti of Ramsoft.
Stefano Donati wrote chapters 3.1, 5.1, 9.3 and helped with errors correction.

Thanks to Dominic Morris who provided info about hook codes 43h, 44h, 45h and 47h.

Petri Andras sent the document reported in chapter 12.1.

You may contact these people at the following addresses:

- * THE RAMSOFT STAFF:
- Ramsoft WHQ (WWR)... ramsoft@retroplay.com
- Luca Bisti luca.bisti@studenti.ing.unipi.it

Stefano Donati the.king@flashnet.it

* Dominic Morris djm@jb.man.ac.uk

* Petri Andras petri@mit.bme.hu

You can always get the latest version of this tech at the RAMSOFT homepage:

<http://www.retroplay.com/Mecenate/ramsoft> (World Wide Ramsoft)

If the address above doesn't work, try:

<http://www.geocities.com/SiliconValley/Bay/5673>

which will redirect you the the current URL for World Wide Ramsoft.

See you there!