master your zx microdrive

programs, machine code and networking

andrew pennell

master your zx microdrive

programs, machine code and networking

andrew pennell

First published 1983 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street,
London WC2R 3LD

Reprinted 1983

ISBN 0 946408 19 X

Cover design by Graphic Design Ltd.
Illustration by Stuart Hughes.
Typeset and printed in England by Commercial Colour Press, London E7.

CONTENTS

Contents in detail

CHAPTER 6
Using the RS232 Interface
Peripheral types and speeds, Text channels, Binary channels, driving a printer including TAB implementation, screen copies, and other uses.

CHAPTER 7
Using the Network
Sending programs over the net, sending data, broadcasting, Printer & Microdrive server programs.

CHAPTER 8
Machine-Language and the Interface
The shadow ROM, paging mechanism, how streams and channels work, channel variables, ROM routines — Microdrive, RS232, Network and Various. Hook code summary.

CHAPTER 9
Adding BASIC Commands
How to alter the syntax, a program to change the Microdrive commands, and how to add commands from machine-code.

Appendix A
The Interface system variables, their locations, mnemonics and their uses.

Appendix B
Z80 listings of Stream14–Z$, On EOF goto, OPEN #anything, Status, On ERROR goto, and RS232 TAB, with annotations.

Appendix C
Interface bugs — A comprehensive list of the major faults affecting use of the Interface from BASIC and machine-code.

Introduction

My aim in writing this book is to reveal the best ways of using the Sinclair ZX Interface 1, with particular reference to the ZX Micro drives. The Interface greatly increases the usefulness of the Spectrum, and the Microdrives offer fast-access mass storage at very low cost.

The standard features are all clearly explained, but are supplemented by a lot of extra 'hidden' features that make the Interface much more useful. There are several machine-code routines included which greatly increase its potential, and they have been presented so that anyone can easily enter and use them without any knowledge of machine-code. For those readers who do have such knowledge I have included a section on the Interface ROM routines and how to use them, as well as annotated listings of my own routines.

The machine-code programs are given in the form of a series of numbers in DATA statements, and a FOR loop to POKE them into place. Wherever possible the machine-code has been position-independent. This means that it can be put anywhere in memory and will still execute correctly. The best place for machine-code is above RAMTOP (see page 179 in the Spectrum manual), which can be moved down with the CLEAR statement. Other places to store machine-code are the user-defined graphics area (168 bytes) and the printer buffer (256 bytes). REM statements are to be avoided for storing machine-code using the Interface, for technical reasons. For ease of use, each routine has two recommended locations (for 16K and 48K machines) that will allow all the routines to be present simultaneously. If they are used, RAMTOP must first be moved down sufficiently — for all the routines, CLEAR 32009 for 16K or CLEAR 64779 for 48K will reserve enough room. As some of the longer routines contain many numbers, which are liable to error, each routine forms a 'checksum' as it POKEs. If the total is incorrect, the routine will STOP with an appropriate message. The data should then be re-checked, and corrected where necessary.

There are also sections on program security and the adding of add- itional commands to BASIC. The appendices contain more technical information, as well as documenting the known problems (or 'bugs') when using the Interface.

The very production of this book begins to show some of the Spectrum's potential as a 'serious' machine — I used a 48K Spectrum with a real keyboard and printer interface, in conjunction with a word-processor and dot-matrix printer.

Andrew Pennell,
Cliftonville, Kent.
September 1983.

CHAPTER 1
Streams and Channels

Before discussing the many extra features that the ZX Interface 1 gives the Spectrum, the concepts of streams and channels should be understood, as they are fundamental to the efficient use of the Microdrives, as well as RS232 and Networking.

A *channel* is a part of a computer system that data can be sent to and received from, such as the keyboard, for data input, and the screen, for data output. The routes along which the data passes to the channels are called *streams.*

On the Spectrum, there are sixteen streams available from BASIC, four of which are initially assigned to certain channels, as shown in the table below:

STREAM NO.	CHANNEL SPECIFIER	OUTPUT USE	INPUT USE
#0	"K"	lower screen	keyboard
#1	"K"	lower screen	keyboard
#2	"S"	upper screen	none
#3	"P"	ZX printer	none

Streams 0 and 1 are identical, and are attached to the "K" channel. This outputs characters to the lower screen (where error reports and INPUTs occur), and inputs them from the keyboard. Stream 2 is the "S" channel, which prints characters on the upper screen, and stream 3 is the "P" channel, which sends characters to the ZX printer. The Spectrum system prevents input from streams 2 and 3. For example, attempting the statement INPUT #3;a$ will produce the error message **Invalid I/O device.**

The symbol for stream is the hash sign (#, symbol shift 3), which can be added to PRINT, INPUT, and some other commands. For example, to print on the lower lines on the screen, which are normally unavailable, you can use streams 0 or 1 — eg.

PRINT #0; AT 0,0;"Line 23"; AT 1,0;"Line 24";: PAUSE 0

The pause is necessary to prevent the 'OK' report from destroying the message. When printing to channel "K", the lower screen may unexpectedly scroll up into the upper screen. This can be prevented by the use of AT commands, and by ending any text with a semi-colon.

The statement PRINT #2; is the same as the usual PRINT statement, which sends characters to channel "S" ie the upper screen. PRINT #3; is the same as LPRINT, which prints characters on the ZX printer (channel "P"). An advantage of using streams 2 and 3 in programs is that a variable can easily be used to determine whether output goes to the screen or the printer. The following program demonstrates the use of this technique:

```
1000 INPUT "Printer (Y/N)";a$
1010 LET c=2: IF a$="y" OR a$="Y" THEN LET c=3
1020 PRINT #c;"This is a ZX Spectrum"
```

It is also possible to mix streams in the same PRINT statement, eg

```
2000 PRINT #2;"The screen"; #3;"The ZX Printer"
```

The statements INPUT and LIST can also be used with streams. With a bare Spectrum, (ie with no Interface connected) only INPUT #0; and INPUT #1; statements are allowed, and are equivalent to the normal INPUT statement. However, the INPUT # statement is extremely useful when used with the Interface, as will be seen in later chapters.

The LIST #n statement, where 'n' is a stream number, can also be used, and directs a listing of a BASIC program to the selected stream. LIST #3 is equivalent to the LLIST command, which produces a listing on the ZX printer, and LIST #0 produces and interesting, though not at all useful effect on the screen.

The INKEY$ #n statement can also be used with streams, but is of no use with the standard channels. INKEY$ #0 and INKEY$ #1 normally produce null strings as results, and #2 and #3 are not allowed. Later chapters show how this can be used with the extra Interface channels.

Open # and CLOSE

With a standard Spectrum, the OPEN # command is rather limited. Its purpose is to attach a channel to a stream, and has the general form

```
OPEN #n,f$
```

where n is the stream number, and f$ is the channel specifier. Note that the # sign is part of the OPEN # keyword, obtained by extended mode symbol shifted 4. The value of n must be from 0 to 15 (else an **Invalid stream** error will occur), and the channel specifier must be a single letter, and specify a valid channel (in either upper- or lower-case), ie "K","k","S","s","P"

or "p". If it is not valid, then an **Invalid filename** error will occur. With the Interface 1 connected, channel specifiers "M","N","B" and "T" and their lower-case versions are also accepted, and represent Microdrive, Networking, Binary RS232 and Text RS232 channels, respectively. These additional channel names also allow the use of a semi-colon as a separator in the OPEN # statement, as well as a comma, and some also require further data to follow them. Further details are given in the relevant chapters.

As an example,

```
OPEN #3;"s"
```

will make all stream 3 output, normally the ZX printer, go to channel "S", the upper screen, so LLIST will produce a listing on the screen. This can be useful for debugging programs that use the ZX Printer, while it is not connected, or to save paper. This opposite of this is

```
OPEN #2,"P"
```

which makes all stream 2 output, normally the screen, be sent to the printer. If both these commands are done simultaneously program output can be rather confusing, to say the least!

You can also use OPEN # to utilise additional streams. Normally streams 4 to 15 are 'channel-less', and any attempt to use them (eg PRINT #6;"6";) will produce the **Invalid stream** error message. However, these extra streams can be OPENed in the normal way — for example,

```
OPEN #4,"s"
PRINT #4;"Hi there!"
```

will produce the words "Hi there!" on the screen, channel "S". Streams 4 to 15 are used extensively with the extra Sinclair peripherals, using the Interface.

To 'un-do' an OPEN #n statement, its complement, CLOSE #n command must be used, with n referring to the stream number. Closing streams 0 to 3 will make them revert to their usual channels, while closing streams 4 to 15 will reset them to no channel.

IMPORTANT: beware of using the CLOSE # statement with streams 4 to 15 with no Interface connected — due to a fault in the BASIC, if the stream is already closed strange things may occur. Usually, a curious error report is generated, but occasionally the whole computer may reset! The problem does not exist with the Interface connected, as it corrects the fault.

Clear # and MOVE

There is another statement that can be used with channels — namely CLEAR # (note the hash). This bears no relevance to the normal CLEAR

statement — its purpose is to CLOSE streams 4 to 15, and to reset streams 0 to 3 to their initial channels. Caution should be exercised in its use with the extra Interface channel types. The final command for use with channels is MOVE. It can have different syntax forms, but it is basically

MOVE #a TO #b

Where **a** and **b** are stream numbers (and **TO** is a keyword). What it does is to read in a character from stream a and prints it on stream b. It repeats this process until a certain condition is satisfied — the condition depends on the type of channel that stream number a is connected to. It is a very powerful command, and its different applications are explained in subsequent chapters.

It is possible for machine-language routines to create special streams for particular uses. For example, the BASIC operating system internally uses channel "R" on stream −1 (which is not available to BASIC programs) to 'print' characters into the internal workspace area of memory. It is also possible to modify existing channels — all the independently produced Centronics — type printer interfaces modify channel "P", so that all stream 3 output (eg LPRINT) sends characters to an external printer via their interface, instead of the ZX printer.

When using several streams at once, it is often difficult to keep track of which channel is connected to which stream, and which streams are not used at all. The following program produces a table showing all the relevant details on each stream, including streams −3 to −1, which are available only to machine-code programs. When the program is RUN, it prints a summary of the use of each stream, and whether it can be used for input, or output, or both. When the summary has finished, you are prompted for a stream number. When you enter it, greater detail on that particular stream is given. The program, as it stands, gives information on the normal channels only. Further features will be added in the relevant chapters in the course of the book.

Stream Lister

```
1000 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
1005 CLS : PRINT TAB 10; INVERSE
 1;"STREAM USAGE"
1010 PRINT "Stream";TAB 7;"In/Ou
t";TAB 14;"Name";TAB 19;"Use"
1020 FOR s=-3 TO 15
1025 PRINT TAB 2;s;
1030 LET d=FN p(s*2+23566+8)
```

```
1040 IF d=0 THEN PRINT TAB 19; I
NVERSE 1;"Unused": GO TO 1200
1050 LET d=d+FN p(23631)-1
1060 IF FN p(d+2)<>5572 THEN PRI
NT TAB 7;"In";
1070 PRINT TAB 9;"/";
1080 IF FN p(d)<>5572 THEN PRINT
 TAB 10;"Out";
1090 LET f$=CHR$ PEEK (d+4)
1100 PRINT TAB 14;"""";f$;"""";T
AB 19;
1110 IF f$="K" THEN PRINT "lower
 screen": GO TO 1200
1120 IF f$="S" THEN PRINT "upper
 screen": GO TO 1200
1130 IF f$="P" THEN PRINT "print
er": GO TO 1200
1140 IF f$="M" THEN PRINT "Micro
drive": GO TO 1200
1150 IF f$="N" THEN PRINT "Netwo
rk": GO TO 1200
1170 IF f$="T" THEN PRINT "RS232
": GO TO 1200
1180 IF f$="R" THEN PRINT "Works
pace": GO TO 1200
1190 PRINT FLASH 1;"Unspecified"
1200 NEXT s
1210 PRINT AT 0,0;
1500 INPUT "Enter stream number
or ENTER to exit ?"; LINE S$
1505 IF S$="" THEN STOP
1510 CLS
1515 LET S=VAL S$
1520 PRINT TAB 10; INVERSE 1;"ST
REAM NUMBER ";S''
1530 LET D=FN P(S*2+23566+8)
1540 IF D=0 THEN PRINT "CLOSEd s
tream": GO TO 1500
1550 LET D=D+FN P(23631)-1
1560 LET F$=CHR$ PEEK (D+4)
1570 PRINT "Channel specifier:";
F$
1580 REM Check each type
1600 IF F$="K" THEN GO TO 2000
```

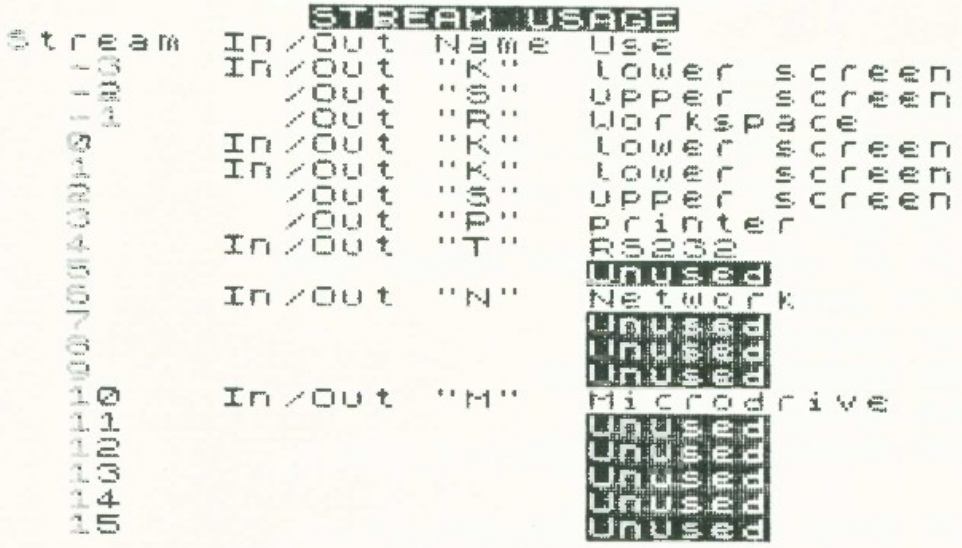
```
1610 IF F$="S" THEN GO TO 2020
1620 IF F$="P" THEN GO TO 2040
1660 IF F$="R" THEN GO TO 2050
1670 PRINT FLASH 1;"Unknown spec
ifier"
1700 GO TO 1500
1800 PRINT "Output routine   :";
FN P(D)
1810 PRINT "Input routine    :";
FN P(D+2)
1820 IF FN P(D)=5572 THEN PRINT
FLASH 1;"Input only"
1830 IF FN P(D+2)=5572 THEN PRIN
T FLASH 1;"Output only"
1870 RETURN
1999 REM Channel K
2000 PRINT INVERSE 1;"Lower scre
en/keyboard"
2010 GO TO 2060
2019 REM Channel S
2020 PRINT INVERSE 1;"Upper scre
en"
2030 GO TO 2060
2039 REM Channel P
2040 PRINT INVERSE 1;"ZX Printer
"
2045 GO TO 2060
2049 REM Channel R
2050 PRINT INVERSE 1;"Workspace"
2060 GO SUB 1800
2070 GO TO 1500
```

The program works by PEEKing various memory locations — read this explanation only if it interests you.

The **s** for-next loop (line 1020) steps through each stream in turn. The variable **d** becomes the displacement in the STRMS area (1030) for the selected stream, which will be zero if the stream is closed (1040). The displacement is then added to the system variable CHANS (1050), with d pointing to a number of bytes in the CHANS area of memory. The established channels (ie K,S,P, and R) each occupy the minimum of five bytes in the CHANS area (giving a total of 20 bytes). In each section of five bytes, the first two point to the 'output a character' routine, the second two point to the 'input a character' routine, and the fifth byte is the upper-case channel specifier. Location 5572 is the **Invalid I/O device** error jump (try RANDOMIZE USR 5572 to see), and this is checked for in line 1060 and

1080. The channel specifier is printed in line 1100, and then checked for the known channels in lines 1110–1180, and a relevant device name printed. After the summary, the user can enter a stream number (1500), and further details are then given. For the current channels, this is only the Input and Output locations, using a subroutine at 1800, but this will be extended later on. **Figure 1** shows a typical output from the program.

Figure 1. Stream Lister Output



Stream 14-z$ program

The following machine-code routine creates stream 14 such that it 'prints' characters into a BASIC string variable. When the routine is called with a GO SUB 8000 statement, the variable **st** should contain the place in memory where the routine is to go. It is 101 bytes in length, and its recommended locations are 65260 (48K) and 32490 (16K). After the routine has been called, all stream 14 output will be added to the end of the string variables z$ (which must not be dimensioned). If z$ does not exist, or is dimensioned, a **Variable not found** error will occur. It is possible to PRINT #14; and LIST #14, and, perhaps most usefully, it is used in the next chapter to CATalogue a Microdrive into the string. There is one restriction however — statements which print strings directly will not work correctly — for example,

PRINT #14;"string"

will incorrectly add "ssssss" to the end of z$. To overcome this, either print each character individually (eg **PRINT #14;"s";"t";"r";"i";"n";"g"**) or use another string variable (**LET a$ = "string": PRINT #14;a$**). For advanced programmers — the reason is that the routine shifts sections

of memory up to insert the characters, and also shifts the workspace up, where the temporary string is stored, so the first character is always added.

Stream14-z$ listing

```
7995 REM ************************
7996 REM * stream14-z$ routine *
7997 REM ************************
7998 REM st=start location
7999 REM recommended:65260 / 324
90
8000 RESTORE 8070
8005 LET c=0
8010 FOR i=st TO st+100
8020 READ a: POKE i,a: LET c=c+a
8030 NEXT i
8035 IF c<>10557 THEN PRINT "Che
cksum error": STOP
8040 CLOSE #14
8050 RANDOMIZE USR st
8060 RETURN
8070 DATA 42,83,92,43,197,229,1,
11
8075 DATA 0,205,85,22,209,33,59,
0
8080 DATA 193,9,213,235,115,35,1
14,35
8085 DATA 235,1,247,255,9,1,9,0
8090 DATA 237,176,225,35,237,75,
79,92
8095 DATA 167,237,66,34,50,92,1,
0
8100 DATA 0,201,196,21,90,40,0,4
0
8105 DATA 0,11,0,245,42,75,92,12
6
8110 DATA 254,90,40,11,254,128,2
02,112
8115 DATA 6,205,184,25,235,24,24
0,35
8120 DATA 78,35,70,3,197,229,9,2
05
8125 DATA 82,22,35,235,225,193,1
12,43
8130 DATA 113,241,18,167,201
```

CHAPTER 2
Starting with Microdrives

How they work

A Microdrive cartridge contains a continuous loop of about 16 feet of thin magnetic tape, of a higher quality than that used in audio cassettes. The Microdrives themselves consist basically of a motor, to pull the tape round, and a record/playback head like that in a cassette player. The tape is pulled past the head at high speed, and can have data stored on it by recording, or read from it during playback. It is really a very fast cassette system, without the hassle of rewinding tapes etc.

The continuous loop system does have disadvantages though — the main one is access time. If the section of tape with your program on has just passed the head, and you wish to load it, the whole tape has to pass the head before it finds it again, as it cannot 'rewind' the tape. This means it can take up to seven seconds to find a program or some data on the tape.

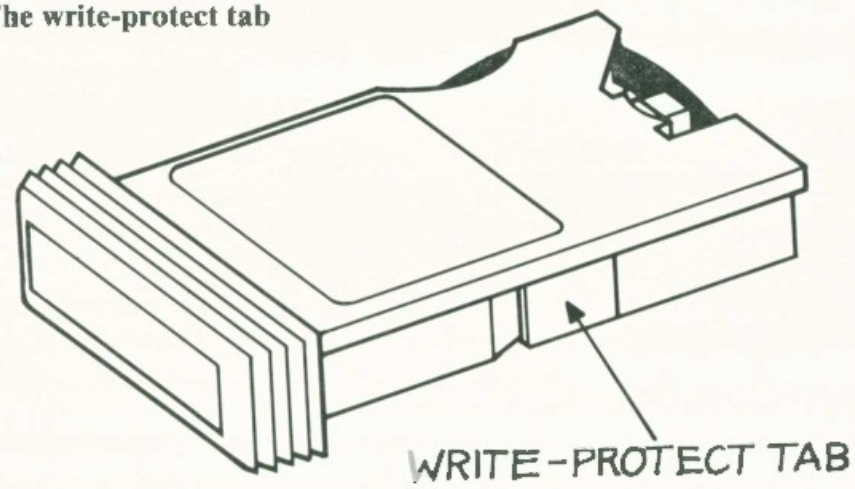
When writing data to a cartridge, it too can be quite slow. If the cartridge is almost new, with few other programs on it, the computer will soon be able to find a spare section on which to store it. However, if it is a well used cartridge the computer may have to search through the tape, and store little bits of your program in different places on the tape, which may take some time. This will also make it slower to load the next time.

Write-protection

Audio cassettes have two small tabs on the back, which can be pushed out to prevent you from recording something new on a cassette with something else already on it. They are called write-protect tabs, and are also used on the Sinclair Microdrive cartridges, for a similar purpose.

With reference to **Figure 2**, each cartridge has a small plastic tab on one side. If you lever it out, with a screwdriver or similar object, it will stop you, or anyone else from writing data on to it. This is very important for commercial programs, and can be very useful for your own that must not be erased. If, at a later stage after you have removed the write protect tab, you wish to re-use the cartridge, you can put some sticky tape back over the cutout. The tape should not be wide enough to overlap onto the top or bottom surfaces on the cartridge, else it may not fit correctly into the Microdrive.

Figure 2. The write-protect tab



Handling cartridges

The tape that cartridges contain is very delicate. You should never touch it with your fingers or anything else, and when one is not in use, always put it in its box to protect it.

When inserting cartridges into Microdrives, ensure that they are the right way up (with the large label uppermost) and push firmly. Once inserted, make sure they are pushed as far in as they will go, else errors may occur when they are in use. There are two very important rules to obey when using cartridges:

1. *Never remove a cartridge while the red LED is on.*
2. *Never have a cartridge in a Microdrive while the power is turned on or off.*

Failure to obey these rules may result in permanent damage to the cartridge, or the Microdrive itself.

Cartridges do not last as long or as well as cassette tapes, because of their stringent operating conditions. Always keep backup copies of important programs or data on another cartridge, or better still, on cassette.

Setting up a cartridge using FORMAT

When a new, blank cartridge is obtained, the tape inside will contain random magnetic information. Any attempt to use it will produce a **Microdrive not present** error. To enable it to be used, it must be configured using the FORMAT command. For the Microdrives, this has the general form

FORMAT "m";d;"*name*"

where d is the drive number (from 1 to 8) and "*name*" is the title that will be permanently assigned to the cartridge, of up to ten letters in length. This will take around 30 seconds, during which time the computer will configure the cartridge, store its title on it, and test each section of tape several times.

If it finds a section that it cannot use because it has been damaged, for example, it will mark the space to be avoided in future operations. The border will flash during the process. Before formatting, make sure the cartridge is properly engaged in the Microdrive. The first cartridge that I ever formatted was not pushed in fully, with the result that the computer thought only one-third of it was usable! When I repeated the command after pushing it in further, it formatted correctly. With brand new cartridges, it is best to do two FORMAT commands in quick succession.

The FORMAT command will destroy any programs that were previously on the cartridge, so always double check before you do the command. An error will occur if you try to format a cartridge with the write-protect tab removed.

After a FORMAT you can check the process by entering **CAT d** (where **d** is the drive number) which, after about seven seconds should print the title of the cartridge, then a number. This number is the number of kilobytes (K) that are usable on the cartridge. After a FORMAT, this should always be at least 85K. The most I have ever had on a cartridge is 92K.

Programs and the Microdrives

One of the main advantages of the Microdrives is their great speed of saving and loading of cassettes. To use them, commands similar to those for cassette are required, with a few modifications. For example, to save the stream list program in the previous chapter to cassette, you could use

SAVE "streams"

To use these commands with the Microdrives, various things have to be inserted in the line after the command. Firstly, an asterisk (*) should be added, then **"m"**; to show that it is a Microdrive operation, then **d;** to show which drive the operation is to use. To save the stream program onto a cartridge in drive 2, you could use

SAVE *"m";2;"streams"

(You can use "m" or "M" — there is no distinction between them.) The additional commands will not be accepted by the syntax checker of a Spectrum without the Interface connected.

When any Microdrive operation occurs, the motor will turn and the red LED come on. If a drive is operated without a cartridge in place, it will emit a high-pitched squeal, and produce the error **Microdrive not present**. You can normally tell if a drive/cartridge combination is working correctly by the sound they make. In normal operation, a smooth buzz comes from the drive. If anything is wrong, such as a cartridge is not inserted properly, the drive usually makes a strange 'clonking' sound, sometimes with a rattle. If this occurs, push the cartridge firmly into the drive. If it persists, BREAK (by holding CAPS SPACE) and examine the cartridge, replace it and try

again. If the drive squeals when a cartridge is in place, and an error message occurs, it probably means the cartridge is faulty and the tape can no longer turn. It should be replaced. If all is well, during a SAVE the drive will operate for at least seven seconds, the cartridge is searched for the chosen filename. If it finds one, the error **Writing to a 'read' file** will occur. The border flashes at any time when data is actually being written to a cartridge.

It is possible, as with cassettes, to use the LINE function in a SAVE statement so that a BASIC program automatically begins when it has loaded. To make the above program start from line 1000, you could use

SAVE *"m";2;"stream"LINE 1000

After a SAVE, you can check the correct operation of the equipment by using the verify function eg

VERIFY *"m";2;"stream"

If there is any difference, the error **Verification failed** will be produced. Microdrive programs should always verify perfectly — if they do not, check both cartridge and Microdrive thoroughly. If the fault persists, replace one or both items.

To load a program from cartridge, the LOAD command is used with the syntax

LOAD *"m";d;"*name*"

where d is the drive number and "*name*" is the required program name. Unlike cassette, it is not possible to load the first program on the tape by using an empty string as a filename. For example, attempting

LOAD *"m";1;""

will produce the error **Invalid filename**. If a program name cannot be found on a cartridge, a **File not found** error will occur.

The merge function also works on the Microdrives, and has a similar syntax to the other commands described ie

MERGE *"m";d;"*name*"

This will load the program "*name*" from drive **d** and merge it with the existing program and variables, replacing any existing ones with those on the cartridge. Unlike the similar cassette statement, it is not possible to MERGE an auto-executing program, saved with LINE xxx. An attempt to do so will produce the message **Merge error**.

An additional command, not required for cassette files, is ERASE. Its purpose is to delete from the cartridge any file, and has the syntax

ERASE "m";d;"*name*"

The erase statement does not require an asterisk, unlike the other Microdrive commands. It is not permitted to follow the filename with anything else eg LINE 2000.

If you BREAK during a SAVE operation, the file will become incomplete, and will not be loadable. You must ERASE any file that has been partially created with an interrupted SAVE. If you try to SAVE a program that is already on the cartridge, the error **Writing to a 'read' file** will occur. The old program should be ERASEd before a new one is SAVEd.

Chaining programs

Using the SAVE *"m"...LINE facility, one program can auto start and load another, perhaps some machine-code. The problem is that the chained programs will only work in a particular drive — to get round this, the first program should have a line such as

```
LET d = PEEK 23766
```

then the variable d will contain the drive that the program loaded from. If the second program was called "prog2", to load it from the first you could use

```
100 LOAD *"m";d;"prog2"
```

The two programs will then run regardless of which drive they are in. By the way, something like

```
100 LOAD *"m";PEEK 23766;"prog2"
```

will *not* work, for technical reasons. Always use a variable.

Arrays and machine-code on cartridge

As well as storing programs on cartridge, it is also possible to store arrays, machine-code, and screens, by adding certain keywords after the filename in the relevant commands. To store arrays, follow the filename with DATA, then the array name, numeric or string. For example, to save the array b() on drive 2 under the name "wages", use

```
SAVE *"m";2;"wages"DATA b()
```

With cassettes, it was possible to save ordinary, undimensioned string variables to tape, but when loaded back they become corrupted. This has been prevented on the Microdrive — to show it, type

```
LET a$ = "testing": SAVE *"m";1;"string"DATA a$()
```

which will produce **Nonsense in BASIC**.

Machine-code and bytes can be stored by adding the keyword CODE followed by two numbers, separated by a comma. The first number is the start address, and the second is the number of bytes. Both numbers are optional in LOAD and VERIFY statements. Pictures from the screen can

also be stored, by using SCREEN$. Unlike cassette files, it is possible to VERIFY saved pictures.

A **Wrong file type** error can be caused by trying to load a file with the wrong command eg

LOAD *"m";1;"test"CODE

if "test" was a program.

CATalogue

To find out what files you have on a cartridge, the CAT (short for catalogue) command must be used. It has the form

CAT d

where **d** is the drive number. This produces on the screen the title of the cartridge, then a blank line, then an alphabetic list of all the files on the cartridge, and finally a number. The number indicates how much of a cartridge is unused in kilobytes (K), and should always be at least 85K on a blank cartridge. A catalogue can be sent to other channels than the screen using

CAT #n,d

where n is the stream number. For example,

CAT #3,2

will produce a catalogue of drive 2 on stream 3, normally the ZX Printer. It will only list the first 50 files that it finds, so if you have more than 50 a CAT will not display all of them.

What happens if you want to know what files are on the cartridge from within a program? One rather crude and slow way would be to clear the screen, do a CAT, then read each character from the screen using the SCREEN$ function. This method is rather cumbersome, and useless if there are more than 20 files on the cartridges as the screen would scroll up, losing the first filenames. The best way is to use the stream14–z$ routine in the previous chapter to set up stream 14 to add characters to the variable z$, then use

LET z$="": CAT #14,d

and z$ would contain the whole catalogue, to examine at will. To be able to use the information in z$, its format must be known, and is as follows:

Firstly, there are 10 characters which are the title of the cartridge, followed by two newline codes (CHR$ 13s). Then each filename is alphabetically stored, each consisting of 10 characters, then a newline. Finally, there is another newline, then one or two characters giving the number of kilobytes left, and lastly another newline. This format is made clearer by the following subroutine that produces a neat catalogue on the screen. The

stream14−z$ machine-code should be entered and set up before the GO SUB 2000.

Neat catalogue listing

```
1999 REM Neat Catalogue
2000 LET z$="": CAT #14,d
2010 CLS
2020 PRINT "Cartridge name:"; IN
VERSE 1;z$( TO 10)
2025 PRINT
2030 LET z$=z$(13 TO )
2040 LET f=0
2050 IF LEN z$<10 THEN GO TO 210
0
2060 LET f=f+1
2070 PRINT f;". ";z$( TO 10),
2080 LET z$=z$(12 TO )
2090 GO TO 2050
2100 PRINT ''f;" files leaving "
;z$(2 TO LEN z$-1);"K"
2110 RETURN
```

Storing variables

If you have a large program which requires quite a few arrays and crucial variables, it can be a very time-consuming process to either save the whole program and variables (with a normal SAVE) or by saving each array with a different filename (using SAVE. . . . DATA). The following subroutines allow you to save all the variables from a program onto a cartridge, and then load them back at a later stage, possibly into another different program. The variables are saved onto drive 1 under the names "var1" and "var2", but this can be changed to suit by altering lines 4040, 4050, 4110, and 4130. It is not possible to make the filenames and drive numbers into variables, because of the technique used to save them. By the way, it is not a misprint — lines 4010 and 4015 are the same, and both are necessary. The SAVE routine can be called by GO SUB 4000, but the LOAD routine should be started with GO TO 4100 — GO SUB cannot be used as it contains a CLEAR statement, which clears all previous GO SUBs, so line 4140 should be amended to jump back to the correct line in the calling program.

Variable save and load listing

```
3999 REM SAVE the variables
4000 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
```

```
4010 LET l=FN p(23641)-FN p(2362
7)
4015 LET l=FN p(23641)-FN p(2362
7)
4020 POKE 23565,l-256*INT (l/256
)
4030 POKE 23566,INT (l/256)
4040 SAVE *"m";1;"var1"CODE 2356
5,2
4050 SAVE *"m";1;"var2"CODE FN p
(23627),l
4060 RETURN
4098
4099 REM LOAD the variables back
4100 CLEAR
4110 LOAD *"m";1;"var1"CODE 2356
5
4120 DIM a$(FN p(23565)-7)
4130 LOAD *"m";1;"var2"CODE FN p
(23627)
4140 GO TO xxxx: REM rest of pro
gram
```

Multiple copies

As has been mentioned, it is always recommended to keep backup copies of important programs and data on alternate cartridges, or cassette. However, built into the Spectrum is the undocumented facility to SAVE a file any number of times on the same cartridge, with the same filename. If you try to save a file twice with the same name from BASIC though, an error will occur. To make a multiple copy of a program, or any other file type, just before you do the SAVE, you should POKE 23791,x where x is the number of copies, from 1 to 255, though a value of 2 is usually best. Then, when the SAVE is done, the file is saved x times. The number of copies is reset to 1 after the SAVE. This is invisible to you, as a CAT will only print the filename once. The advantage of this method of multiple copies is that if you, or someone else accidentally ERASE the file, only one copy will be deleted. The remaining copies will stay on the cartridge, and can be loaded in the normal way. If you make x copies of a file, to completely delete it you must do the ERASE command x times. Of course, multiple copies take up x times more space on the cartridge. Note that such multiple copies are NOT safe from a FORMAT "m"; command, which will delete all copies at once.

CHAPTER 3
Microdrive File Handling

Serial files

A file is a block of data that can have items added to it, and read from it. With *serial* files, you must read and write data in a particular order. For example, if you wanted the 10th item, you must initially read the first nine items of data before you can read the tenth. On the Spectrum, files are normally stored on a cartridge, and are of the serial type. (The other type of file *random access* is not available from BASIC.)

Chapter 1 explained how streams work, and described the channels "K", "S" and "P". To store data on a Microdrive, other than by saving it as programs or arrays, another channel specifier is used — "m" (or "M") for Microdrive. In a similar way to that already described, the OPEN # statement must be used to attach the channel "m" to a stream, though its syntax is slightly different, as more information is required by the Spectrum. It has the form

OPEN #n;"m";d;"*name*"

which will create a new channel with the specifier "m" on drive **d** with the name "*name*", and then attach the new channel to the stream number **n**. As the file "*name*" has just been created, there can be no data in it, so it is designated a *write* file.

Write files

These are files that did not exist before an OPEN # command was executed, and are created by it. The main command for sending data to a Microdrive file is the PRINT #n; statement, which works in a similar way to the normal PRINT statement, but 'prints' the data onto a cartridge. As a simple demonstration, RUN the following program and notice when the Microdrive actually operates:

```
100 OPEN #4;"m";1;"testfile"
110 FOR i=1 to 500
120 PRINT i;" ";
130 PRINT #4;i
140 NEXT i
150 CLOSE #4
```

You may have expected the drive to operate most of the time, as it prints each number, but it doesn't! After the OPEN # command, the drive operates for a while as it searches for a file with the same name, which it should not find for it to be a write file. What OPEN # also does is to create an area of memory called a *buffer*, which it uses as a temporary store for characters. When it comes across a PRINT # statement it will store the characters in the buffer, but won't write them onto the cartridge unless the buffer is full. When it does fill, the drive will operate for about a second while it transfers the 512 characters from the buffer onto the cartridge, and then clears the buffer, ready to wait for some more. This is the reason why the drive only operated a couple of quick times, as the buffer fills about four times over. A CLOSE # statement is absolutely essential when using write files because, if the buffer is not full, and you don't do a CLOSE, the last data you thought you sent with PRINT # will never get onto the cartridge, and the file will be incomplete. A CLOSE # will send what is left in the buffer to the cartridge, then remove the buffer from memory. If you do a CLEAR # while a write stream is open,the file will become incomplete and unusable. Such a partial file should be erased. If you try and OPEN a write file that is already open on another stream, the message **Reading from a 'write' file** will occur.

Read files

Once a write file that has been created by OPEN # and has had data sent to it, and been CLOSEd, it becomes a 'read' file, and is visible on the cartridge CATalogue. To read data from it, it must be opened, again using OPEN # with the same syntax as before. The actual reading of data is usually done with the INPUT # command. To read in the file created in the previous program, enter the following:

```
100 OPEN #4;"m";1;"testfile"
110 INPUT #4;i
120 PRINT i;" ";
130 GO TO 110
```

This initially creates a read file channel on stream 4. The variable **i** is then read from the file (line 110) and printed on the screen. The program then loops back, but will eventually stop with an **End of File** error. (We shall see how to detect an end of file before it occurs later on in the chapter). As you may have been able to tell as the program ran, a buffer is also used with read files, and is again 512 characters long. When an INPUT # is encountered, a block of 512 characters (or less if it is the last block) is read from the cartridge, and stored in the buffer. When a character is needed by the INPUT routine (or INKEY$ #) it is extracted from the buffer. If it is the last character in the buffer then another block of 512 will be read in from

the cartridge until there are no more in the file, when an **End of file** error will occur.

There is another way of reading characters from a file, apart from using INPUT # — the INKEY$ # function, which was briefly mentioned in Chapter 2. Unlike INPUT, which assembles a sequence of characters until it reaches the end of a line and then copies it to a variable, INKEY$ #n just reads a single character. Unlike the normal INKEY$ function, it does 'wait' for a character — INKEY$ # with a Microdrive stream will never return an empty string ("") as a result. When you have finished reading a file, it should be ended with a CLOSE # command. Although it is not as important as with write files, it is advisable to always do it for neatness, and to prevent large amounts of memory from being used up by unnecessary buffers. It is alright to do a CLEAR # with open read files.

When using INPUT # statements, the counter used by the Spectrum to produce the 'scroll?' prompt is reset, so if you are reading a file and printing it on the screen, it will scroll up continuously. If this is undesirable in your program, use a line like

LET sc = PEEK 23692: INPUT # (whatever): POKE 23692,sc

The reason for it is that the counter is reset by any INPUT statement, which is never usually noticed. In fact, a simple way to disable the 'scroll' prompt in a program is to use the statement

INPUT ""

Notes on PRINT and INPUT

You should be very careful when using PRINT # and INPUT # statement with Microdrive files, because of the effect of the separators used between arguments. When printing to the screen or printer, a comma (,) will take you to the start of the next half-line, and an apostrophe (') will do a new line. However, there are no 'lines' in Microdrive files, and printing a comma to a file (eg PRINT #4;a,b) will actually send the control code for comma ie a CHR$ 6. Similarly, an apostrophe will send a CHR$ 13 (a newline code). Character code 6's can confuse the Spectrum when you try to use INPUT # (but not INKEY #). For example, try the following program:

```
10 OPEN #4;"m";1;"errortest"
20 LET a$ = "first"; LET b$ = "second"
30 PRINT #4;a$,b$
40 CLOSE #4
50 OPEN #4;"m";1;"errortest"
60 INPUT #4;a$
70 INPUT #4;b$
80 CLOSE #4
```

You may not have expected the error **End of file** at line 70, but the PRINT # at line 30 actually sent to the cartridge the text "first", then a comma character (CHR$ 6), then the text "second", and finally a newline (CHR$ 13). An INPUT # statement expects variables in a file to be separated by CHR$ 13's, so it read in a$ as "first"+CHR$ 6+"second", and emptied the file, so trying to read in b$ afterwards produced the error.

Separators in INPUT # statements also create unexpected problems — if you try

```
10 OPEN $4;"m";1;"testfile"
20 INPUT #4;a$,b$
```

(assuming the file "testfile" exists) the error **Writing to a 'read' file** will occur. This is because the comma in line 20 is trying to send a CHR$ 6 to a read file. If quote marks (") are included in data files, reading strings using INPUT # can cause problems, so INPUT #...LINE should always be used. the golden rules for using PRINT # and INPUT # are:

1. *Always separate variables with apostrophes when using* **PRINT #** *or use separate lines eg* **PRINT #4;a$'b$** *or* **PRINT #4;a$: PRINT;#4b$**

2. *Always use semi-colons as separators in* **input** *statements, and use* **LINE** *for string variables eg* **INPUT #4; LINE a$; LINE b$**

Using MOVE with Microdrive files

The MOVE command has many uses, but only the ones directly applicable to Microdrive files will be discussed here. If you have a data file, and you want to see what it contains without having to open it and reading it character by character, you can do

MOVE "m";d;"*name*" TO #2

(where TO is a keyword) which prints the contents of the specified file onto stream 2, the screen. (If you try and MOVE any other sort of file, such as a program, a "Wrong file type" error will occur). If you want hard copy of a file, you can use

MOVE "m";d;"*name*" TO #3

which will print it on stream 3, the ZX printer. You can also use MOVE to copy data files —

MOVE "m";d1;"*file 1*" TO "m";d2;"*file 2*"

which copies the data from "file1" on drive d1 to drive d2 with the name "file2".

Adding to files

Suppose you have a file containing a lot of data, such as a list of your software, and you wish to add some items to the end of it. As you cannot print to a read file, you must create a new file, copy the contents of the old one to it and, while it is still OPEN, add the new data. One way would be to read each item from the old file, then print it to the new, but it would be rather slow and inefficient, and would not work with some forms of data. The best way to do it is using the MOVE command, as shown in the following example, where it is required to add the strings "Space Invaders" and "25 December 1983" to the file "software":

```
10 OPEN #4;"m";1;"software2"
20 MOVE "m";1;"software" TO #4
30 PRINT #4;"Space Invaders" "25 December 1983"
40 CLOSE #4
```

If you wanted the new file to have the same name as the old one, you could add the lines

```
50 ERASE "m";1;"software"
60 MOVE "m";1;"software2" TO "m";1;"software"
70 ERASE "m";1;"software2"
```

The above program will execute very much faster if the two files are stored on different Microdrives.

The MOVE command is also very useful for adding two data files together — for example, to form the file "third" by adding the file "first" to the file "second"—

```
10 OPEN #4;"m";1;"third"
20 MOVE "m";1;"first" TO #4
30 MOVE "m";1;"second" TO #4
40 CLOSE #4
```

The speed of the above can also be greatly improved if two or even three Microdrives are used, with a file on each.

Using LIST

Although the main method of writing data to a cartridge is by using PRINT #, the LIST # statement can also be used. It will list a BASIC program into a stream, which can be a Microdrive channel. When you read the file back, the token (eg STOP) will only be one character (CHR$ 266 for STO), and

not individual characters with spaces between keywords. Converting a program into character form using LIST # can be very useful if you wish to process or search through your BASIC program, perhaps with a word-processor. The next program, Program Search, will search through a data file created by OPEN #, LIST # then CLOSE # statements for any particular sequence, and will print any line containing it on the screen. This is very useful for searching for variable names, or certain GO TO statements. It can also be used for searching through any type of data file, not just program listings. It will eventually stop with the report **End of file.**

```
999 REM Program Search
1000 INPUT "Drive no?";d,"Filename?";LINE f$
1005 CLOSE #4: OPEN #4;"m";d;f$
1010 INPUT "String to search for?";LINE b$
1020 INPUT #4; LINE a$
1030 FOR i=1 TO LEN a$-LEN b$
1035 IF a$(i TO i+LEN b$-1)=b$ THEN PRINT a$: PAUSE
50: GO TO 1050
1040 NEXT i
1050 GO TO 1020
```

If you wish to search for tokens (eg GO TO) you should type THEN, which puts the Spectrum into K mode, then press the relevant key (G for GO TO), then delete the word THEN.

End of file (EOF)

When reading a file, eventually you will reach the end of it. If you try to read any more, the message **End of file** will be produced. There are several ways however to detect an EOF before it occurs, which makes programs much neater and more efficient.

The simplest way to detect an EOF is to print a special character or sequence of characters that would normally not be used at the end of a file, just before CLOSEing it. My personal preference is to use the string CHR$ 0+CHR$ 0 which I would never use for anything else. This is the best method for most programs, but cannot be used if you have a program which is designed to read all sorts of data files, or which can read program or machine-code files (using a method explained later).

Other disc-based BASICs usually have a function ON EOF GOTO or more generally ON ERROR GOTO which makes control jump to a particular line if an EOF (or any other error) occurs. Unfortunately the Spectrum has no such function, but a couple of lines of BASIC can detect 99% of EOFs. If you add the following lines to a program (as near the

beginning as possible) then the function FN E(x) where x is the stream number will give 1 if the file is empty, else it will give 0.

```
10 DEF FN E(A)=((INT (PEEK (FN D(A)+67)/2)-2*INT
(PEEK (FN D(A)+67)/4) AND (FN P (FN D(A)+11)>=FN P (FN
D(A)+69))))
11 DEF FN P(P)=PEEK P+256*PEEK (P+1)
12 DEF FN D(D)=FN P(D*2+23574)+FN P(23631)-1
```

(If stream **x** is not open or not an "M" channel then the result will be meaningless). The 1% of the time when this fails is when the number of characters in the buffer is an exact mutliple of 512, which is thankfully rare. While creating a write file, the way to make sure this doesn't happen is to use the function

```
FN P(FN D(X)+11
```

just before closing stream x. If this is 0 then print an extra character to the stream beforehand.

Alternatively the following machine-code routine can be used to achieve an ON EOF GOTO function. It is 67 bytes long, and can be positioned anywhere in memory. Before doing so a GO SUB 8150, the variable **eof** should be set to the required location, and **line** to the required line number (which must exist). The routine POKEs in the code, then executes it. After that, when any EOF occurs, no report will be produced — the interpreter will jump to the specified line. The function is cancelled when any error occurs, including an EOF. If you wish to change the line number of the error jump, change **line** and GO SUB 8220. When an EOF occurs, to find out which line it occured in use

```
LET errline=PEEK 23753+256*PEEK 23754
```

Do this before any Microdrive operations, else it will return an invalid result.

On EOF GOTO listing

```
8145 REM **********************
8146 REM *    On EOF GOTO    *
8147 REM **********************
8148 REM eof=start location, lin
e=error jump
8149 REM recommended:65190/32490
8150 RESTORE 8250
8160 LET c=0
8170 FOR i=eof TO eof+66
```

```
8180 READ a: LET c=c+a
8190 IF a<>260 THEN POKE i,a
8200 NEXT i
8210 IF c<>6456 THEN PRINT "Chec
ksum error": STOP
8220 POKE eof+49,line-256*INT (l
ine/256)
8225 POKE eof+50,INT (line/256)
8230 RANDOMIZE USR eof
8240 RETURN
8250 DATA 33,17,0,9,235,42,61,92
8260 DATA 115,35,114,207,49,1,0,
0
8270 DATA 201,42,61,92,58,58,92,
254
8280 DATA 7,194,3,19,94,35,86,21
3
8290 DATA 205,176,22,253,203,55,
174,205
8300 DATA 110,13,42,69,92,34,201
,92
8310 DATA 17,260,260,33,66,92,11
5,35
8320 DATA 114,35,54,1,253,54,0,2
55
8330 DATA 195,125,27
```

Using programs etc as data files

Normally, if you try and OPEN a Microdrive channel with the filename of a program, or any other non-data file type, the error **Wrong file type** will occur. It can be very useful indeed to read such alternate files from BASIC, and this can be done with the following machine-code routine, called open #anything. It is almost equivalent to the OPEN statement, but will open any file, regardless of its type. It will also inform you of whether the file exists or not, which in itself is very useful. Before the GO SUB 8350, the variable **open** should be the required location for the routine (recommended 32320 for 16K and 65090 for 48K). The GO SUB will POKE the code in, but not execute it.

OPEN #anything listing

```
8344 REM ************************
8345 REM *   OPEN#anything    *
8346 REM ************************
```

```
8347 REM open=start location
8348 REM recommended:65090/32320
8350 RESTORE 8400: LET c=0
8360 FOR i=open TO open+93
8370 READ a: LET c=c+a
8375 POKE i,a
8380 NEXT i
8385 IF c<>10725 THEN PRINT "Che
cksum error": STOP
8390 RETURN
8400 DATA 58,216,92,205,39,23,33
,17
8410 DATA 0,175,237,66,1,0,0,216
8420 DATA 50,215,92,33,10,0,34,2
18
8430 DATA 92,42,123,92,34,220,92
,58
8440 DATA 216,92,135,33,22,92,95
,22
8450 DATA 0,25,217,229,217,229,2
07,34
8460 DATA 221,203,24,70,40,13,17
5,207
8470 DATA 33,207,44,225,217,225,
217,1
8480 DATA 1,0,201,221,203,4,190,
175
8490 DATA 229,207,33,209,225,115
,35,114
8500 DATA 221,126,67,230,4,198,2
,79
8510 DATA 6,0,217,225,217,201
```

When you wish to use it, you must tell it which drive, stream and file you are interested in. To do this, you should POKE 23766 with the drive number, POKE 23768 with the stream number, and POKE the first 10 locations in the user-defined graphics area with the filename. For example, if you wish to OPEN #6;"m";2;"test" (where "test" is any type of file) the instructions would be

```
8500 POKE 23766,2: REM drive number
8510 POKE 23768,6: REM stream number
8520 LET a$="test"
8530 FOR i=1 TO 10
```

```
3540 IF i>LEN a$ THEN POKE USR "a"+i-1,32: GO TO
3560
3550 POKE USR "a"+i-1,CODE a$(i)
3560 NEXT i
3570 LET a=USR open: REM call the routine
```

Note that if the filename is less than 10 characters long, the appropriate number of spaces (CHR$ 32s) should be added and POKEd (line 3540). The value returned from the **USR open** function (a in the example) is as follows:

0 — stream already open
1 — file not found
2 — data file
6 — not a data file

If the file did not exist, the stream is closed and the file not created. Thus the method is for read files only — write files are not created by it, unlike the OPEN # command.

If the returned number is 6 ie it is not a data file, how do you know what sort of file it is? The answer lies in the first nine characters read from the file. These characters hold all the attributes for the file, including the type. The first character code determines the type, thus:

0 BASIC program
1 numeric array
2 string array
3 bytes

The following eight bytes hold details such as length, start, line number etc, and are actually taken from the system variables HD_00 to HD_11 (see Appendix A for exact details). After the first nine bytes, the rest is the actual file. The next program, True Catalogue, is a much improved version of the CAT function. As well as giving each filename, it gives all other details about the file — its type, length, auto-start line number etc. It requires the machine-code routines Stream14-z$ and Open#anything to function. It is slower than a CAT, but gives a great deal of useful information.

True Catalogue listing

```
4997 REM ************************
4998 REM TRUE CATALOGUE
4999 REM ************************
5000 LET z$="": CAT #14,d
5010 CLS : POKE 23766,d: POKE 23
768,15
5020 PRINT INVERSE 1;"Title:";z$
( TO 10)
5030 LET z$=z$(13 TO )
5040 IF LEN z$<10 THEN GO TO 533
0
5050 FOR i=1 TO 10
5060 POKE USR "a"+i-1,CODE z$(i)
5070 NEXT i
5080 CLOSE #15: LET z=USR open
5090 PRINT z$( TO 10);"  ";
5100 IF z=2 THEN PRINT "Data fil
e": GO TO 5300
5110 LET a$="": FOR i=1 TO 9
5120 LET a$=a$+INKEY$#15: NEXT i
5130 LET z=CODE a$(1)
5140 GO TO 5150+z*40
5149 REM z=0 Program
5150 PRINT "Program   LINE ";COD
E a$(8)+256*CODE a$(9)
5160 GO TO 5300
5189 REM z=1  Numeric array
5190 PRINT "Array  ";CHR$ (CODE
a$(6)-32);"()"
5200 GO TO 5300
5229 REM z=2  String array
5230 PRINT "Array  ";CHR$ (CODE
a$(6)-96);"$()"
5240 GO TO 5300
5269 REM z=3  Code
5270 PRINT "Code  ";CODE a$(4)+2
56*CODE a$(5);
5280 PRINT ",";CODE a$(2)+256*CO
DE a$(3)
5300 CLOSE #15
5310 LET z$=z$(12 TO )
5320 GO TO 5040
5330 PRINT 'z$(2 TO LEN z$-1);"K
```

```
 bytes left"
5340 RETURN
```

Generating non-data type files

As well as the ability to read non-data type files from cartridge, it is possible to generate any type of file from a BASIC program. It requires no machine-code, only a POKE command. Using this facility can be quite difficult, and beginners are not recommended to try this, as mistakes in its use can crash the computer.

The method to do this is to follow an OPEN # command to create a file with lines such as

```
100 DEF FN p(p) = PEEK p + 256*PEEK (p + 1)
110 POKE FN p(s*2 + 23566 + 8) + FN p(23631) + 66,4
```

where s is the stream number in the OPEN command. This tricks the system into creating non-data type file segments when any data on the stream is written to a cartridge. After the POKE, you should PRINT nine characters to the chosen stream. The first what type of file it is to be, and the following eight determine the parameters of the file, and are the same as those mentioned above for reading files — the system variables HD_00 to HD_11 (see Appendix A).

As an example, the following program saves all the BASIC variables in a program like the routine in Chapter 2, but with a big difference — instead of saving them as two CODE files, it saves them as a program file, so they can be MERGEd into other programs. It can be rather slower than the other method, but is much more flexible.

Save variables Listing

```
3999 REM save variables as progr
am
4000 OPEN #4;"m";1;"variables"
4010 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
4020 LET d=FN p(4*2+23566+8)+FN
p(23631)-1
4030 IF PEEK (d+4)<>CODE "M" THE
N PRINT "Error": STOP
4040 POKE d+67,4: REM fiddle it
4045 FOR i=1 TO 2: NEXT i
4050 LET z=FN p(23641)-FN p(2362
7)-1
4055 LET z=FN p(23641)-FN p(2362
```

```
7)-1
4060 PRINT #4;CHR$ 0;: REM progr
am flag
4070 PRINT #4;CHR$ (z-256*INT (z
/256));CHR$ INT (z/256);
4080 PRINT #4;CHR$ PEEK 23627;CH
R$ PEEK 23628;: REM start
4090 PRINT #4;CHR$ 0;CHR$ 0;: RE
M program length
4100 PRINT #4;CHR$ 255;CHR$ 255;
: REM 'line number'
4110 FOR i=FN p(23627) TO FN p(2
3627)+z-1
4120 PRINT #4;CHR$ PEEK i;
4130 NEXT i
4150 CLOSE #4
4160 RETURN
```

Line 4000 creates the write file, then line 4040 does the necessary POKE. Lines 4045 and 4050 may seem superfluous, but both are vital. The variable z then holds the number of bytes in the variables area (line 4055), then line 4060 prints the first character, to signal a program file. Line 4070 then prints the two characters to define the length of the file, and line 4080 prints two to define the start of the variable area. Line 4090 sets the program length as 0, and line 4100 sets 65535 to be the auto-start line number (ie no auto start). The loop from 4110 to 4130 then sends each byte of the variables to the file, before line 4150 closes it.

Other ways of reading data

Although INPUT # and INKEY$# are the main command for reading files, the MOVE can be more useful. If the Stream14 – z$ routine is active, the line

LET z$="": MOVE "m";1;"file" TO #14

will read the whole data file from the cartridge at once and place it in the string variable z$, memory permitting. It can then be manipulated and modified as required, then written back to another file with a single PRINT statement eg

OPEN #4;"m";1;"file2": PRINT #4;z$;: CLOSE #4

This method has the least effect on wear and tear of the cartridges, which can be an important factor when dealing with large data files that are used frequently, as the drive will only operate twice — once to read, and once to write.

Status routine

The next routine, called Status, allows a program to determine whether a particular Microdrive is connected or not, whether a cartridge is in it, and whether it is write-protected. Before doing a GOS SUB 8550, the variable **stat** should be set to the desired memory location.

Status listing

```
8545 REM ********************
8546 REM *      Status        *
8547 REM ********************
8548 REM stat=start address
8549 REM recommended:64960/32190
8550 RESTORE 8630: LET c=0
8560 LET x2=INT ((stat+100)/256)
: LET x1=stat-256*x2+100
8570 FOR i=stat TO stat+128
8580 READ a: LET c=c+a
8590 POKE i,a
8600 NEXT i
8610 IF c<>14341+4*(x1+x2) THEN
PRINT "Checksum error": STOP
8620 RETURN
8630 DATA 58,214,92,243,24,33,33
,136
8640 DATA 19,43,125,180,32,251,3
3,136
8650 DATA 19,6,6,219,239,230,4,3
2
8660 DATA 4,16,248,24,84,43,124,
181
8670 DATA 32,239,1,0,0,24,84,17
8680 DATA 0,1,237,68,198,9,79,6
8690 DATA 8,13,32,20,122,50,247,
0
8700 DATA 62,238,211,239,205,x1,
x2,62
8710 DATA 236,211,239,205,x1,x2,
24,17
```

```
8720 DATA 62,239,211,239,123,211
,247,205
8730 DATA x1,x2,62,237,211,239,2
05,x1
8740 DATA x2,16,214,122,211,247,
62,238
8750 DATA 211,239,24,162,197,245
,1,135
8760 DATA 0,11,120,177,32,251,24
1,193
8770 DATA 201,219,239,230,1,1,1,
0
8780 DATA 32,1,3,197,175,207,33,
193
8790 DATA 201
```

The GO SUB only puts the machine-code in place, it does not execute it. Before using, POKE 23766 with the drive number then use some lines such as

```
2500 LET a = USR stat
2510 IF a = 0 THEN PRINT "Microdrive not connected"
2520 IF a = 1 THEN PRINT "Cartridge present"
2530 IF a = 2 THEN PRINT "Cartridge write-protected"
```

Colour commands

In the Sinclair Interface manual, a fleeting reference is made to the fact that colour commands may not work after using channels other than "K", "S" or "P". In fact, it is a serious problem that one should be particularly aware of when file handling.

The problem is that after using Interface channels for output, permanent colour statements (eg PAPER 3) have no apparent effect. More importantly, they actually send data to write files. To correct this, before setting permanent colours do a dummy **PRINT;** statement. To show this fault in action, try the following:

```
10 OPEN #4;"m";1;"coltest"
20 PRINT #4;"First line"
30 FLASH 1: PAPER 4: CLS
40 PRINT #4;"Second line"
50 CLOSE #4
60 MOVE "m";1;"coltest" TO #2
```

Line 10 creates a write file "coltest", and line 20 sends a line of data to it. Line 30 should make the whole screen flashing green, but instead sends the colour codes to the cartridge. Lines 40 and 50 send another line of data to

the file then close it. Finally line 60 prints the whole file to the screen revealing the unwanted insertion of the colour codes. The way to fix it in this case is to add the line

25 PRINT;

Microdrive additions to Stream Lister

The following lines can be added to Stream Lister to allow it to give greater detail about Microdrive streams.

Stream Lister 2

```
1630 IF F$="M" THEN GO TO 2500
1840 IF FN P(D)<>8 AND FN P(D+2)
<>8 THEN RETURN
1850 PRINT "Shadow ROM output:";
FN P(D+5)
1860 PRINT "Shadow ROM input :";
FN P(D+7)
2499 REM Channel M
2500 PRINT INVERSE 1;"MICRODRIVE
"
2510 GO SUB 1800
2520 PRINT "Drive number      :";
PEEK (D+25)
2530 LET M=FN P(D+26)
2540 PRINT "Tape loop map:"
2550 FOR I=0 TO 31: FOR J=1 TO 6
2560 POKE 16384+2048+2*32+J*256+
31-I,PEEK (M+I)
2570 NEXT J: NEXT I
2575 PLOT 0,95: DRAW 255,0: PLOT
 0,88: DRAW 255,0
2580 PRINT '"Map area          :"
;M;"-";M+31
2590 PRINT "Cartridge name    :";
2600 FOR I=D+44 TO D+53
2610 PRINT CHR$ PEEK I;
2620 NEXT I: PRINT
2630 PRINT "Filename           :";
2640 FOR I=D+14 TO D+23
2650 PRINT CHR$ PEEK I;
2660 NEXT I: PRINT '"Free space
     :";
2670 LET F=0
```

```
2680 FOR I=0 TO 255
2690 LET F=F+NOT POINT (I,90)
2700 NEXT I
2710 PRINT F/2;"Kbytes"
2720 GO TO 1500
```

The extra details include the drive number, cartridge number, cartridge name and the filename, as well as the free space on the chosen cartridge. Also shown is a 'tape loop map', which is a sort of bar-code that shows graphically which sections of the tape are used. **Figure 3** shows the output from the program with a brand new cartridge. Any black bars on the tape loop map mark areas that are either used, or non-existent. The large black area on the left shows a section of the tape that simply doesn't exist, as the system can cater for longer tapes. The bars in the middle show where the join in the tape is, and the bar on the far right shows the first sector, which is never used. The tape loop map for a used cartridge will contain more vertical bars, showing where programs and data are stored.

Figure 3 Stream Lister output for new cartridge

```
STREAM NUMBER 10

Channel specifier:M
MICRODRIVE
Output routine     :8
Input routine      :8
Shadow ROM output  :4568
Shadow ROM input   :4386
Drive number       :1
Tape loop map:
Map area           :23792-23823
Cartridge name     :21.8.83  3
Filename           :test
Free space         :90Kbytes
```

The first main addition to Stream Lister is lines 1840–1860. These check for Interface channel types — if they are then the shadow ROM output and input routine locations are printed. These extra lines are also needed by the RS232 and Networking additions to the program. Lines 2500–2720 print various details about the stream. The way the tape loop map is printed is by POKEing directly to the screen the data stored in memory, in lines 2530–2575. The free space is calculated by counting the number of white bars in the map on the screen (2670–2710).

Games High Score routines

To demonstrate an application of data files, there follows some routines that can add a High-Score to a game, by storing the table in a file. It is in the form of two main subroutines — lines 9000 – read in the table, and should be used before the game starts, and lines 9100 – which updates the table if a new high score has been attained.

High Score routine

```
8996 REM ************************
8997 REM * High Score routine *
8998 REM ************************
8999 REM Read in the data
9000 LET ns=5: DIM s(ns): DIM s$
(ns,10)
9010 CLOSE #4: OPEN #4;"m";1;"hi
score"
9020 FOR i=1 TO ns
9030 INPUT #4;s(i); LINE s$(i)
9040 NEXT i: CLOSE #4
9050 GO TO 9300
9098
9099 REM Redo the file if a high
 score
9100 LET ns=5: IF sc<s(ns) THEN
RETURN
9110 PRINT ' FLASH 1;"      A ne
w high score!!!!      "
9120 INPUT "Enter your name (max
 10 letters)"; LINE a$
9130 FOR i=1 TO ns
9140 IF sc<=s(i) THEN NEXT i
9150 ERASE "m";1;"hiscore"
9155 OPEN #4;"m";1;"hiscore"
9160 FOR j=1 TO ns
9170 IF j<i THEN PRINT #4;s(j)'s
$(j)
9180 IF j=i THEN PRINT #4;sc'a$
9190 IF j>i THEN PRINT #4;s(j-1)
's$(j-1)
9200 NEXT j
9210 CLOSE #4
9220 RETURN
9299 REM Print the score table
```

```
9300 CLS
9310 PRINT FLASH 1; INK 7; PAPER
 0;TAB 8;"HIGH SCORES";TAB 31;"
"
9320 FOR i=1 TO ns
9330 PRINT PAPER i; INK 9,,TAB 6
;i;" ";s$(i);TAB 20;s(i),,,
9340 NEXT i
9350 RETURN
9988
9989 REM Set up new file
9990 OPEN #4;"m";1;"hiscore"
9992 FOR i=1 TO 5
9994 PRINT #4;0'" "
9996 NEXT i: CLOSE #4
```

It stores the table as 10 elements, in the order 1st score, 1st name, 2nd score, 2nd name etc. Lines 9900– onwards should only be executed once as they set up the file with null data. To read in the table, lines 9000–9050 are used. Line 9000 sets the number of items in the table (in **ns**), and the arrays s(), for each score, and s$(), for each name. Line 9010 opens the file for reading, and lines 9020–9040 read in each score and name and place them in the arrays. The file is closed, and the table printed from line 9300– . After the game has finished, the score should be placed in the variable **sc**, and a GO SUB 9100 executed. Line 9100 checks to see if the score was less than the lowest in the table. If it is then a RETURN is made, as the player has not qualified for entry in the table. If the player has qualified, then he enters his name (line 9120) and his new position in the table is calculated in lines 9130–9140, and placed in **i**. Line 9150 erases the old file, and line 9155 creates a new write file. The loop from 9160 to 9200 then prints into the file the new table, by inserting the new score and name and shifting any data below it down by one position. Finally line 9210 closes the file.

CHAPTER 4
Unifile

The program in this chapter is a major database program, using the Microdrives and, optionally, a printer (either the ZX type or RS232). It is closely based on the program Unifile 1 by David Lawrence, in his book *The Working Spectrum*, with his permission.

It is intended for 48K owners, though details are given later of a reduced version for 16K owners. The program has not been renumbered so that the lines taken from the original program are the same.

Unifile listing

```
1000 PAPER 7: CLS : BORDER 7: IN
K 6: PAPER 0: PRINT PAPER 2;"
UNIFILE ";F$;TAB 31;" "
1010 PRINT '"FUNCTIONS AVAILABLE
:"
1020 PRINT '"      1)SET UP NEW F
ILE"
1030 PRINT '"      2)ENTER INFORM
ATION"
1040 PRINT '"      3)SEARCH/DISPL
AY/CHANGE"
1050 PRINT '"      4)STOP"
1055 PRINT '"      5)SAVE/LOAD/CA
T"
1060 PRINT ''"PLEASE ENTER WHICH
 YOU REQUIRE."
1070 PAUSE 0: LET Z$=INKEY$
1080 CLS
1090 IF Z$="1" THEN GO SUB 1210
1100 IF Z$="2" THEN GO SUB 1440
1110 IF Z$="3" THEN GO SUB 2180
1120 IF Z$="4" THEN GO SUB 1150
1125 IF Z$="5" THEN GO TO 3500
1130 CLS
1140 GO TO 1000
1150 PRINT AT 10,5; INK 7; PAPER
 2;"FILING SYSTEM CLOSED"
```

```
1160 BEEP 2,2
1180 INPUT "Have you input new i
nformation  you wish to save? (Y
/N)";Q$: IF Q$="N" THEN STOP
1190 SAVE "UNIFILE": PRINT '"Rew
ind tape, then press any key to
VERIFY": PAUSE 0: VERIFY "UNIFIL
E": STOP
1200 REM *************************
1210 REM ENTRY STRUCTURE
1220 REM *************************
1230 PRINT PAPER 2;"          FILE
 STRUCTURE            "
1235 GO SUB 4200
1240 PRINT '"HOW MANY ITEMS IN E
ACH ENTRY?"
1250 INPUT X
1260 CLS
1270 DIM A$(X,20)
1280 PRINT PAPER 2;"          NAMES
 OF ITEMS             "
1290 FOR I=1 TO X
1300 PRINT "ITEM ";I;":";
1310 GO SUB 2780
1320 PRINT Q$(2 TO )
1330 LET A$(I)=Q$
1340 NEXT I
1350 DIM B$(28000)
1360 LET B$(1 TO 4)=CHR$ 2+CHR$
0+CHR$ 2+CHR$ 255
1370 DEF FN A()=256*CODE Y$(2*S-
1)+CODE Y$(2*S)
1380 DEF FN A$()=B$(C TO C+CODE
B$(C)-1)
1390 LET P=5
1400 LET Y$=CHR$ 0+CHR$ 1+CHR$ 0
+CHR$ 3
1410 LET N=2
1420 RETURN
1430 REM *************************
1440 REM NORMAL INPUT
1450 REM *************************
1460 LET R$=""
1470 PRINT PAPER 2;"            EN
```

```
TRIES                    "
1480 PRINT '"COMMANDS AVAILABLE:
"
1490 PRINT '">ENTER ITEM SPECIFI
ED"''">"#ZZZ"" TO QUIT"
1500 PRINT "**********************
************"
1510 PRINT "FILE SIZE:";P-1;"/";
LEN B$
1520 FOR I=1 TO X
1530 GO SUB 2810
1540 GO SUB 2780
1580 PRINT Q$(2 TO )
1590 IF Q$(2 TO )="ZZZ" THEN RET
URN
1600 LET R$=R$+Q$
1610 NEXT I
1620 CLS
1630 GO SUB 1660
1640 GO TO 1440
1650 REM ************************
1660 REM PLACE DATA IN FILE
1670 REM ************************
1680 IF P+LEN R$-1<LEN B$ THEN G
O TO 1730
1690 PRINT AT 14,10;"FILE NOW FU
LL"
1700 PRINT '''"   Press any key t
o continue"
1710 PAUSE 0
1720 RETURN
1730 LET POWER=INT (LN (N-1)/LN
2)
1740 LET S=2^POWER
1750 LET T$=R$(2 TO CODE R$(1))
1760 FOR K=POWER-1 TO 0 STEP -1
1770 LET C=FN A()
1780 LET U$=FN A$()(2 TO )
1790 LET S=S+(2^K)*(T$>U$)-(2^K)
*(T$<U$)
1810 IF S>N-1 THEN LET S=N-1
1820 IF S<2 THEN LET S=2
1830 NEXT K
1840 LET C=FN A()
```

```
1850 LET U$=FN A$()(2 TO )
1860 IF T$<U$ THEN LET S=S-1
1870 LET B$(P TO P+LEN R$-1)=R$
1880 LET N=N+1
1890 LET Y$=Y$(1 TO 2*S)+CHR$ IN
T (P/256)+CHR$ (P-256*INT (P/256
))+Y$(2*(S+1)-1 TO )
1900 LET P=P+LEN R$
1910 RETURN
1920 REM ************************
1930 REM CHANGE ENTRY
1940 REM ************************
1950 LET S=S-1
1960 LET C=FN A()
1970 LET R$=""
1980 PRINT "ENTRY ";S-1;":-"
1990 FOR I=1 TO X
2000 GO SUB 2810
2010 GO SUB 2830
2020 PRINT AT 17,0; PAPER 2;"
       AMEND                    "
2030 PRINT "COMMANDS AVAILABLE:"
2040 PRINT ">""ENTER"" LEAVES IT
EM UNCHANGED"'">""ZZZ"" DELETES
WHOLE ENTRY"'">ENTER NEW ITEM"
2050 GO SUB 2780
2060 IF LEN Q$=1 THEN LET R$=R$+
B$(C TO C+CODE B$(C)-1)
2070 LET C=C+CODE B$(C)
2080 CLS
2090 IF LEN Q$=1 THEN GO TO 2120
2100 IF Q$(2 TO )="ZZZ" THEN GO
TO 2130
2110 LET R$=R$+Q$
2120 NEXT I
2130 GO SUB 3130
2140 IF Q$(2 TO )="ZZZ" THEN RET
URN
2150 GO SUB 1660
2160 RETURN
2170 REM ************************
2180 REM SEARCH
2190 REM ************************
2200 LET S=2
```

```
2205 LET ST=2
2210 PRINT PAPER 2;"
SEARCH                "
2220 PRINT ''"COMMANDS AVAILABLE
:"
2230 PRINT ">INPUT ITEM FOR NORM
AL SEARCH"'">PRECEDE WITH ""SSS"
" FOR"'" SPECIAL SEARCH"'">PRECE
DE WITH ""III"" TO SEARCH"'" FOR
 FIRST CHARACTER OF ENTRY"'">""E
NTER"" FOR FIRST ITEM ON FILE"
2235 PRINT ">""PPP"" FOR ";"PRIN
TER" AND ST=2;"SCREEN" AND ST=3;
" OUTPUT"
2240 PRINT "**********************
************"
2250 PRINT ''"INPUT SEARCH ITEM:
";
2260 GO SUB 2780
2265 IF Q$=CHR$ 4+"PPP" THEN LET
 ST=5-ST: CLS : GO TO 2210
2270 PRINT Q$(2 TO )
2280 LET S$=Q$
2290 IF LEN S$=1 THEN GO TO 2510
2300 LET C=FN A()
2310 IF LEN S$<5 THEN GO TO 2430
2320 IF S$(2 TO 4)<>"III" THEN G
O TO 2390
2330 FOR I=S TO N
2340 LET S=I
2350 LET C=FN A()
2360 IF B$(C+1)=S$(5) THEN GO TO
 2510
2370 NEXT I
2380 RETURN
2390 IF S$(2 TO 4)<>"SSS" THEN G
O TO 2430
2400 GO SUB 2920
2410 IF C4=1 THEN GO TO 2510
2420 RETURN
2430 FOR I=1 TO X
2440 IF FN A$()=S$ THEN GO TO 25
10
2450 IF FN A$()=CHR$ 2+CHR$ 255
```

```
THEN RETURN
2460 LET C=C+CODE B$(C)
2470 NEXT I
2480 LET S=S+1
2490 LET C=FN A()
2500 GO TO 2430
2510 LET C=FN A()
2520 LET C4=0
2530 IF FN A$()=CHR$ 2+CHR$ 255
THEN RETURN
2540 CLS
2545 IF ST<>2 THEN GO TO 4300
2550 PRINT "ENTRY ";S-1;":-"
2560 GO SUB 2850
2570 LET S=S+1
2580 PRINT AT 16,0; PAPER 2;"
       SEARCH                    "
2590 PRINT "COMMANDS AVAILABLE:"
2600 PRINT ">""ENTER"" TO DISPLA
Y NEXT ITEM"'">""ZZZ"" TO QUIT F
UNCTION"'">""AAA"" TO AMEND"'">"
"CCC"" TO CONTINUE SEARCH"
2610 INPUT P$
2620 CLS
2630 IF P$="CCC" THEN GO TO 2300
2640 IF P$="" THEN GO TO 2510
2650 IF P$<>"AAA" THEN GO TO 271
0
2660 LET C=FN A()
2670 CLS
2680 GO SUB 1930
2710 IF P$="ZZZ" THEN RETURN
2720 IF P$="AAA" THEN RETURN
2730 CLS
2740 GO TO 2260
2750 REM ************************
2760 REM FUNCTIONAL SUBROUTINES
2770 REM ************************
2780 INPUT Q$
2790 LET Q$=CHR$ (LEN Q$+1)+Q$
2800 RETURN
2810 PRINT A$(I,2 TO CODE A$(I,1
));":";
2820 RETURN
```

```
2830 PRINT FN A$()(2 TO )
2840 RETURN
2850 FOR I=1 TO X
2860 GO SUB 2810
2870 GO SUB 2830
2880 LET C=C+CODE B$(C)
2890 NEXT I
2900 RETURN
2910 REM ************************
2920 REM SPECIAL SEARCH
2930 REM ************************
2940 LET C4=0
2950 FOR H=S TO N-1
2960 LET S=H
2970 LET C=FN A()
2980 LET C1=C
2990 FOR I=1 TO X
3000 LET C1=C1+CODE B$(C1)
3010 NEXT I
3020 FOR J=C+1 TO C1-LEN S$+5
3030 IF B$(J TO J+LEN S$-5)<>S$(
5 TO ) THEN GO TO 3060
3040 LET C4=1
3050 RETURN
3060 NEXT J
3070 NEXT H
3080 LET C4=0
3090 RETURN
3100 REM ************************
3110 REM TELESCOPE FILE
3120 REM ************************
3130 LET C=FN A()
3140 LET SHIFT=1000
3150 LET C1=C
3160 LET C3=C
3170 FOR I=1 TO X
3180 LET C1=C1+CODE B$(C1)
3190 NEXT I
3200 LET C2=C1-C
3210 FOR I=C1 TO LEN B$-1 STEP S
HIFT
3220 IF LEN B$-I+1<SHIFT THEN LE
T SHIFT=LEN B$-I+1
3230 LET S$=B$(I TO I+SHIFT-1)
```

```
3240 LET B$(C TO C+SHIFT-1)=S$
3250 LET C=C+SHIFT
3260 NEXT I
3270 LET Y$=Y$(1 TO 2*(S-1))+Y$(
2*(S+1)-1 TO )
3280 FOR I=1 TO N-1
3290 LET S=I
3300 LET C=FN A()
3310 IF C<=C3 THEN GO TO 3350
3320 LET C=C-C2
3330 LET Y$(2*I-1)=CHR$ INT (C/2
56)
3340 LET Y$(2*I)=CHR$ (C-256*INT
 (C/256))
3350 NEXT I
3360 LET P=P-C2
3370 LET N=N-1
3380 RETURN
3390 FOR I=1 TO 20: PRINT CODE Y
$(I): NEXT I
3500 CLS
3510 PRINT "UNIFILE - MICRODRIVE
 OPERATIONS"
3520 PRINT '"  1) SAVE "; INVERS
E 1;F$
3530 PRINT '"  2) LOAD new data"
3540 PRINT '"  3) CATalogue a ca
rtridge"
3545 PRINT '"  4) RETURN to main
 menu"
3550 PAUSE 0: LET Z$=INKEY$
3560 IF Z$="1" THEN GO SUB 3900
3570 IF Z$="2" THEN GO TO 3800
3580 IF Z$="3" THEN GO SUB 3700
3590 IF Z$="4" THEN GO TO 1000
3600 GO TO 3500
3670 REM **********************
3680 REM CATALOGUE ROUTINE
3690 REM **********************
3700 CLS : PRINT "CARTRIDGE CATA
LOGUE:"
3710 INPUT "DRIVE NUMBER? (0 to
exit)";D
3720 IF D=0 OR D>8 THEN RETURN
```

```
3730 CAT D
3740 PRINT #0;AT 0,0; FLASH 1;"
  Press any key to continue     "
;
3750 PAUSE 0
3760 RETURN
3770 REM ********************
3780 REM LOAD ROUTINE
3790 REM ********************
3800 PRINT '' FLASH 1;"WARNING:L
OADING NEW DATA CLEARS CURRENT D
ATA"
3805 PRINT '"Press C to continue
,or any otherkey to exit."
3810 PAUSE 0: LET Z$=INKEY$: IF
Z$<>"C" AND Z$<>"c" THEN GO TO 3
500
3820 GO SUB 4200
3830 INPUT "DRIVE NUMBER (1-8)?
";D
3840 IF D<0 OR D>8 THEN GO TO 38
30
3850 LOAD *"M";D;F$+CHR$ 128
3860 GO TO 1000
3870 REM ********************
3880 REM SAVE ROUTINE
3890 REM ********************
3900 PRINT ''"(EXISTING NAME=";F
$;")"
3905 GO SUB 4200
3910 INPUT "DRIVE NUMBER? ";D
3920 IF D<0 OR D>8 THEN RETURN
3930 CLS
3940 PRINT "   UNIFILE - ";F$
3950 PRINT AT 15,0;"Do you want
";F$'"on drive ";D;" to be ERASE
D? (Y/N)"
3960 INPUT LINE z$
3965 PRINT AT 15,0;,,,,
3970 IF Z$<>"Y" AND Z$<>"y" THEN
 GO TO 4000
3980 ERASE "M";D;F$+CHR$ 128
3990 PRINT '"FILE ERASED"
4000 SAVE *"M";D;F$+CHR$ 128 LIN
```

```
E 1000
4010 PRINT "DATA SAVED - NOW VER
IFYING"
4020 VERIFY *"M";D;F$+CHR$ 128 L
INE 1000
4030 RETURN
4170 REM ********************
4180 REM ENTER FILENAME
4190 REM ********************
4200 INPUT "FILENAME? "; LINE F$
4210 IF LEN F$<10 AND LEN F$>0 T
HEN RETURN
4220 INPUT "(MAX LENGTH 9) "; LI
NE F$
4230 GO TO 4210
4270 REM ********************
4280 REM PRINTER OUTPUT
4290 REM ********************
4300 LPRINT "ENTRY ";S-1;":-"
4310 FOR I=1 TO X
4320 LPRINT A$(I,2 TO CODE A$(I,
1));":";
4330 LPRINT FN A$()(2 TO )
4340 LET C=C+CODE B$(C)
4350 NEXT I
4360 GO TO 2570
```

When you have entered it, type the following:

CLEAR
LET F$=""

then SAVE the program with LINE 1000. To start it initially, do **GO TO 1** — never use RUN as it will clear the data.

Each data file can hold up to 28,000 characters composed of a number of items, defined when the file is initialised. An example would be an address book file, with three items per entry — Name, Address, and Phone number.

Option 1 is to set up a new file, and also prompts you for a name for your file.

Option 2 allows you to enter information onto the file.

Option 3 is the most useful option — it enables you to search the whole file, printing appropriate entries to the screen or printer. The 'special search' option will search every part of each entry for the occurrence of the specified string, and is not very fast. For example, if you entered **SSS-**

KENT with the address book file everyone living in Kent, or whose name was Kent would be printed. A quicker option is the normal search, which just checks the first character of each entry.

Option 4 allows the program and data to be saved on cassette for back-up purposes.

Option 5 allows the current data to be saved, or other data loaded, or a cartridge catalogued. Files are stored on cartridge with a CHR$ 128 added to the end of them, to distinguish them from normal program files. The extra character is not noticed in a CATalogue as it prints as a space.

I will not attempt to explain the operation of this program — it is covered well by David Lawrence in his book. I will however explain the printer subroutine so that users may alter it to suit their files requirements. The routine lies from 4300 to 4350 — **S-1** is the position of the entry in the file and **X** is the number of items per entry. Line 4320 prints the title of each entry, and line 4330 prints the actual data. Line 4340 increments a pointer. For example, if you had the file set up as an address book as above and you wanted to print out the entries in an address label format, one possibility could be:

```
4300 LPRINT "Ref no.";S-1
4310 LPRINT FN A$()(2 TO): REM the Name
4320 LET C=C+CODE B$(C)
4330 LPRINT FN A$()(2 TO): REM the Address
4340 LPRINT
4350 LET C=C+CODE B$(C)
4360 LPRINT "Phone:";FN A$()(2 TO): REM the number
4370 LET C=C+CODE B$(C)
4380 GO TO 2570
```

Further additions

There are still more features that could be added to this. One would be a CAT of only Unifile programs, using Stream14-z$ routine and examining Z$ for CHR$ 128s. Another improvement would be to convert lines 3130–3380 into machine-code, which would greatly increase the speed of deletions and amendments.

16K Version

A limited feature Unifile can be used on a 16K Spectrum with the following modifications:

Delete all REM statements, and lines 3540, 3580, and 3700–3760

Amend lines:

```
1350 DIM B$(1000)
3140 LET SHIFT=100
3375 LET S$=""
```

CHAPTER 5
Program Protection

When you have written a very good piece of software, and you wish to sell it, you do not want every Tom, Dick or Harry to be able to make as many copies as they like of your masterpiece to give to all their friends. Although no program can possibly be 100% uncopyable, many simple techniques can be used to make it as difficult as possible. There are a couple of inbuilt features that will be discussed, as well as some other, sneakier methods. It must be stressed that no method is unbeatable — a determined machine-code programmer will always be able to defeat the protection methods of any program given the time, and the knowledge.

Auto-execution

The simplest way of making programs harder to copy is to make it automatically execute, using the SAVE *....LINE function. It cannot be MERGed, so will always auto-start when loaded. If you call a program "run" (all lower case) then, after a switch on or a NEW, typing RUN will load the program and start it (if it was saved with LINE).

It is also very useful to have program names starting with CHR$ 0, as they will not be shown in a cartridge CATalogue. If you make the "run" program load the 'invisible' one, the potential software copier won't even know the main program's name. Another way is to make the name of a program all spaces — most users will not notice it in a catalogue.

Crash POKEs

One thing that should be prevented is the user breaking into a program once it has started, or in the middle of a LOAD. In the above case this would let him see the name of the invisible file, by examining the listing. One way to do this, which is not very subtle but works beautifully, is to make the first line

POKE 23659,0

What this does is to tell the Spectrum that there are no lines in the bottom section of the screen. Then, when any error occurs (such as **Break into program**) the whole system will 'hang', because there is nowhere to print the

error message. The user will then have to pull the plug out to reset the computer, thus preventing him from examining the listing or copying it.

Beware when using this particular POKE: a CLS command, or any attempt to print on the lower two lines (eg using INPUT) will crash the computer. The POKE is only really useful for machine-code programs while they load — for BASIC programs it is really unsuitable, as it is too limiting. An alternate way is with the line

LET p = PEEK 23613 + 256*PEEK 23614: POKE p,0: POKE p + 1,0

which will still crash the system when any error occurs, but does allow printing on the lower screen.

On Error GOTO

Many other computers have an On Error Goto function, which forces a jump to a certain line when an error occurs. Unfortunately, the Spectrum lacks such a function, so the following machine-code routine was written for the same purpose:

On Error listing

```
8794 REM **********************
8795 REM *  ON ERROR GOTO    *
8796 REM **********************
8797 REM err=start address,line=
error jump
8798 REM recommended:64780/32010
8800 RESTORE 8900: LET c=0
8810 FOR i=err TO err+170
8820 READ a: LET c=c+a
8830 IF a<256 THEN POKE i,a
8840 NEXT i
8850 IF c<>17439 THEN PRINT "Che
cksum error": STOP
8860 POKE err+149,line-256*INT (
line/256)
8870 POKE err+150,INT (line/256)
8875 RETURN
8880 RANDOMIZE USR err
8885 POKE 23734,0: RETURN
8890 RANDOMIZE USR err
8895 POKE 23734,4: RETURN
8900 DATA 33,17,0,9,235,42,61,92
8905 DATA 115,35,114,207,49,1,0,
0
```

```
8910 DATA 201,34,201,92,42,61,92
,17
8915 DATA 3,19,213,205,176,22,25
3,203
8920 DATA 55,174,205,110,13,33,1
85,23
8925 DATA 34,237,92,58,58,92,50,
211
8930 DATA 92,245,207,50,33,129,0
,237
8935 DATA 91,201,92,241,167,237,
82,32
8940 DATA 20,253,203,124,70,32,1
4,254
8945 DATA 7,40,10,62,100,50,211,
92
8950 DATA 62,63,215,24,21,60,71,
254
8955 DATA 10,56,2,198,7,205,239,
21
8960 DATA 62,32,215,120,17,145,1
9,205
8965 DATA 10,12,175,17,54,21,205
,10
8970 DATA 12,237,75,69,92,237,67
,201
8975 DATA 92,205,27,26,62,58,215
,253
8980 DATA 78,13,6,0,205,27,26,33
8985 DATA 59,92,203,174,251,203,
110,40
8990 DATA 252,33,66,92,17,260,27
0,115
8994 DATA 35,114,35,54,1,253,54,
0
8995 DATA 255,253,54,124,0,205,1
10,13
8996 DATA 195,125,27
```

It is 171 bytes long, and position independent. Before the GO SUB 8800 to set it up, the variable **err** should be set to the required memory location, and **line** to the error line number. For normal (non-Interface) operations, do a GO SUB 8880 to activate the routine. When an error occurs, the relevant message will be printed in the normal place. It will then wait for a key to be

pressed, before clearing the message and jumping to the relevant line. The feature is deactivated when an error occurs. The line number where the error occurred can be read by

PEEK 23753 + 256*PEEK 23754

and the error number with

PEEK 23763

For Interface operations, you should do a GO SUB 8890 to activate the routine. Note that if an Interface error occurs (ie one with no report code) the relevant message will not be produced — a question mark will be printed instead, and location 23763 will contain 100.

After any Interface command, the routine is disabled as a result of its operation.

If you wish to disable the routine, do

LET p = PEEK 23613 + 256*PEEK 23614: POKE p,3: POKE p + 1,19: POKE 23734,0

This routine is not compatible with the On EOF GOTO routine in the previous chapter. To check for EOF when an error occurs, PEEK 23763 will be 7 if it is an EOF.

While debugging, beware of infinite loops. If you had line 100 as the error line, and it was

1000 GO SUB 8880

you might never be able to break out of the program, so always include somewhere in your program the option to do the POKEs to disable it.

For inclusion of this routine in a commercial program, it is best to add the following line:

8855 POKE err + 23,94: POKE err + 24,35: POKE err + 25,86

These POKEs alter the routine so that it is not cancelled when an error occurs.

If you wish to include this, or any other machine-code routine in this book in a commercial program, my permission **must** be obtained prior to its release, as they are my copyright.

As has been seen, machine-code and some POKEs can be used very effectively to protect programs, but the converse is also true — machine code can also be used to crack programs by undoing the protection methods. To crack a particular method, the programmer must know what has been used. This is why it is relatively simple for a determined machine-coder to, for example, get a catalogue of invisible files, or merge auto-start programs. It is an unfortunate by-product of the abundant availability of technical data on all Sinclair products.

CHAPTER 6
Using the RS232 Interface

Another feature of the Interface 1 is the facility to transmit and receive data via RS232. RS232 is an (almost) universal standard for serial transmission, and there are a very large number of peripherals, as well as most other computers, that can communicate via RS232. For use with the Spectrum, the most common application is to drive 'proper' wide, plain paper printers, to produce listings and neat program output.

The speed at which the data is sent or received is known as the *baud* rate, and is approximately ten times the number of bytes sent per second. There are nine standard baud rates, and the Spectrum supports all of them, namely 50,110,300,600,1200,2400,4800,9600 and 19200. Teletypes normally use the slow rates of 110 or 300 baud, with CRT (Cathode-ray) terminals running at the high speeds of 2400 baud and over.

As well as different baud rates, different devices may require slightly different data formats. On the Spectrum, the format is fixed at:

— No parity
— Eight data bits
— One stop bit

To use the RS232 facility on the Spectrum, two more channel specifiers are used — "t" and "b". These streams are different in their operation by the way they treat each character.

Text channel "t"

The text channel is most suitable for driving printers, as it does certain things to each character. With reference to the character set in Appendix A in the Spectrum manual, a "t" channel prints the following with each character code:

0–12	(control codes) are ignored
13	(newline) sends a carriage return (13) then a linefeed (11)
14–31	(control codes) are ignored
32–127	(ASCII codes) are sent unmodified
128–164	(graphics codes) are sent as "?"
165–255	(tokens) are de-tokenised into individual characters

This character conversion is ideal for sending program listings, as well as program output, but many printers use codes below 32 to achieve special features — for example, the Epson series use CHR$ 14 to print expanded text, but such a character cannot be sent via a "t" channel. In addition, the extremely useful TAB function is ignored which makes neat program output much harder. An alternate method is to use the "b" channel.

Binary channel "b"

The binary channel is very suitable communicating with devices other than printers, because no character conversion is done. Each character is sent unmodified, so it is ideal for sending raw data, and programs. It is also necessary to use it to send control codes to printers, as they cannot be sent by a "t" channel.

Before your Spectrum can send or receive any data via RS232, it must know the baud rate of your peripheral. This is done by the FORMAT "b"; statement, followed by the baud rate. For example, to use the baud rate of 110, use

```
FORMAT "b";110
```

(You could also use FORMAT "t";110 — they have the same effect). If you enter an unrecognised baud rate, no error message will occur, and the Spectrum will use the next highest recognised rate. To check the rate currently selected use

```
PRINT INT(3500000/((PEEK 23747 + 256*PEEK 23748 + 2)*26))
```

(For the higher speeds, this may be inaccurate by a few baud). If the baud rate is not specified, the system will not usually notice, and strange speeds of transfer may result. Initially the baud rate is set to the standard rate of 9600.

Driving an RS232 printer

Before connecting an RS232 printer to the Spectrum for the first time it should be set up to suit the data format the Spectrum uses. This can usually be accomplished by changing some miniature switches inside the printer, and should only have to be done once. It should be set for the highest baud rate allowed (and write it down in case you forget!), and the data format options should be set to no parity, 8 data and one stop bit. If it offers the option of automatic line-feed after carriage return (auto CR after LF in the jargon) the option should be disabled, as the Spectrum can generate the necessary line-feeds.

When all the required options have been chosen the printer can be connected via a suitable lead to the 9-pin D-type socket on the Interface.

When driving an RS232 printer, it is advisable to have both "t" and "b" channels open simultaneously — the former for text and listings, and the latter for control codes. It is best to use stream 3 for printer "t" channels, to use the LLIST and LPRINT statements in existing programs eg

```
10 FORMAT "t";1200: REM baud rate
20 OPEN #3;"t": REM text on stream 3
30 OPEN #15;"b": REM binary on stream 15
40 LLIST : REM list on channel "t"
50 LPRINT "Normal size"
60 PRINT #15;CHR$ 14; #3;"Double width"
70 CLOSE #3: CLOSE #15
```

In the above example stream 15 is used as a "b" file for sending control codes, and stream 3 is used as a "t" channel for text and listings.

When transmitting RS232 the border goes black if the receiving device (in this case a printer) is 'busy', and cannot accept the data. The Spectrum will wait until it is ready before it sends the data, or until BREAK is pressed. When any RS232 steam is CLOSEd a character 13 is transmitted, to empty printer buffers and the like. A character 13 is not sent when the CLEAR # command is used.

It is not possible to have more than one RS232 device connected at a time, although different streams can connect to the same channel. It is also not possible to have different baud rates on different channels or streams.

The only major problem when driving printers via RS232 is the lack of the TAB command. The following machine-code subroutine will open stream 3 as a "t" channel, but with a slight modification. It will count the number of characters sent, and thus allow the TAB function to be fully implemented. One wonders why this small improvement was not included in the original machine. It also corrects the fault in the software that incorrectly produces double spaces in listings (eg THEN PRINT). The code is written to lie in the ZX printer buffer area of memory, which is never used when stream 3 is re-defined. Note that neither a CLOSE #3 or a CLEAR # will make stream 3 revert to the ZX printer, and nor will it send a carriage-return.

TAB routine

```
8997 REM **********************
8998 REM *     RS232 TAB      *
8999 REM **********************
9000 RESTORE 9100: LET c=0
9010 FOR i=23296 TO 23467
9020 READ a: LET c=c+a
9030 POKE i,a
9040 NEXT i
```

```
9050 POKE 23540,80: REM printer
width
9055 IF c<>19420 THEN PRINT "Che
cksum error": STOP
9060 RANDOMIZE USR 23296
9070 RETURN
9100 DATA 42,79,92,1,15,0,9,17
9110 DATA 23,91,115,35,114,1,0,0
9120 DATA 33,245,91,112,35,112,2
01,254
9130 DATA 32,48,93,254,13,32,26,
33
9140 DATA 246,91,203,70,203,134,
192,33
9145 DATA 246,91,203,134,43,54,0
,62
9150 DATA 13,205,169,91,62,10,19
5,169
9160 DATA 91,254,23,63,192,17,71
,91
9170 DATA 42,81,92,115,35,114,20
1,50
9180 DATA 15,92,17,79,91,24,241,
17
9190 DATA 23,91,205,64,91,58,15,
92
9200 DATA 87,33,244,91,150,210,1
08,4
9210 DATA 35,122,150,213,220,39,
91,209
9220 DATA 122,253,150,187,200,71
,62,32
9230 DATA 197,217,215,217,193,16
,247,201
9240 DATA 254,165,56,5,214,165,1
95,16
9250 DATA 12,253,203,188,134,33,
59,92
9260 DATA 203,134,254,32,32,2,20
3,198
9270 DATA 254,128,56,2,62,63,205
,169
9280 DATA 91,33,245,91,52,126,43
,190
```

```
9290 DATA 192,205,39,91,253,203,
188,198
9300 DATA 201,207,30,201
```

Screen copies

Another useful facility for use with a wide printer is to produce hard copy of the screen display. In its simplest form, this consists of the following small program that works on any printer (with at least 32 characters per line) that reads each character, then sends it to the printer.

```
100 FORMAT "T";BAUD : REM suitable value
110 OPEN #4;"T"
120 FOR Y=0 TO 21
130 FOR X=0 TO 31
140 LET A$=SCREEN$(Y,X)
150 IF A$="" THEN LET A$=" "
160 PRINT #4;A$;
170 NEXT X
180 PRINT #4
190 NEXT Y
```

It works by scanning each character position using SCREEN$, and either prints it or a space if it is unrecognised (line 150). After each line of 32 characters, a newline is sent (line 180). Although this is pretty crude, it can still be a very useful subroutine, particularly as it is printer-independent.

If you want a copy of a high-resolution picture, or a listing with inverse and graphic characters, how can you do it? If you have a suitable printer, you can do true copies of it, as with the COPY command on the ZX Printer. In BASIC though, it is a slow process, but the results are well worth it. I present here two versions — one for the Epson range of printers (with an RS232 board fitted), and one for the Seikosha GP100.

Epson Hi-Res copy

```
1000 FORMAT "b";1200: REM change
 to suit interface
1010 OPEN #3;"b"
1020 LPRINT CHR$ 27;"A";CHR$ 8;
1030 FOR y=175 TO 0 STEP -8
1040 LPRINT CHR$ 27;"K";CHR$ 0;C
HR$ 1;
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (128*POINT (x,y
)+64*POINT (x,y-1)+32*POINT (x,y
```

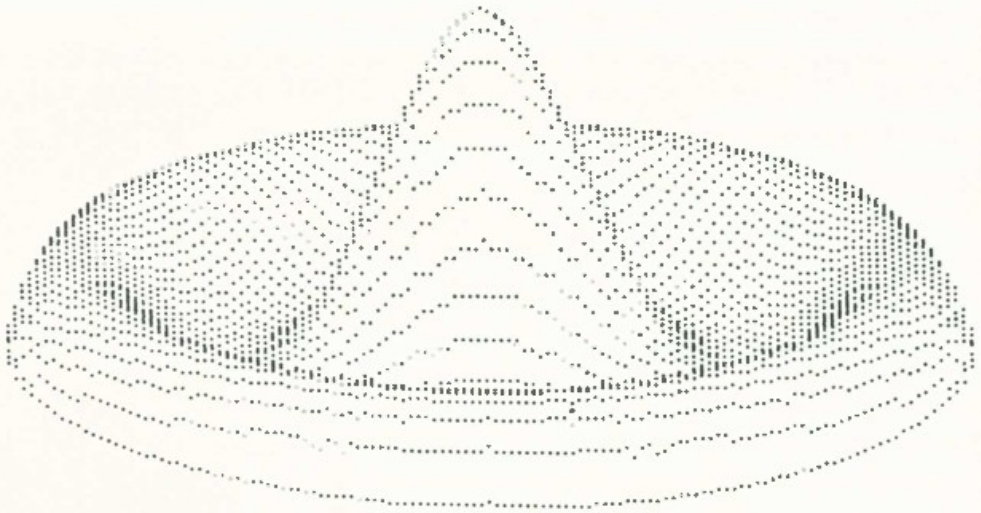
```
-2)+16*POINT (x,y-3)+8*POINT (x,
y-4)+4*POINT (x,y-5)+2*POINT (x,
y-6)+POINT (x,y-7));
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 27;"A";CHR$ 12
```

Seikosha Hi-Res copy

```
1000 FORMAT "b";1200: REM change
 to suit interface
1010 OPEN #3;"b"
1020 LPRINT CHR$ 18
1030 FOR y=174 TO 0 STEP -7
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (POINT (x,y)+2*
POINT (x,y-1)+4*POINT (x,y-2)+8*
POINT (x,y-3)+16*POINT (x,y-4)+3
2*POINT (x,y-5)+64*POINT (x,y-6)
+128);
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 30
```

They work by using the POINT function to calculate a binary number that is sent to the printer. Note that a "b" channel is used, because no character conversion should occur. **Figure 4** shows an example of the output of the Epson routine.

Figure 4. Sample of Epson copy



Other uses for RS232

Channel "b" can also be used for sending programs, data, machine-code and screens, in a similar way to "m" channels, but filenames are not required. This can be very useful for sending programs over the telephone, for example, if an RS232 modem is purchased. With the relevant hardware, to send a program via a 300 baud modem to a friend at the other end of the line, he should first type

FORMAT "b";300: LOAD *"b"

so that his Spectrum is set up to wait for it. You should then type

FORMAT "b";300: SAVE *"b"

Channel "t" should only be used for transmitting listings or text. The commands LOAD *, SAVE *, MERGE *, and VERIFY * can all be used with RS232, and so can OPEN # and CLOSE #. Filenames and following numbers are not required with RS232 channels, though they are accepted by the Spectrum, but ignored. For example, SAVE *"b";5;"name" is accepted, but the extra details are ignored.

Another application could be to use the Spectrum as an RS232 terminal, to transmit characters typed to another computer:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 PAUSE 0: LET a$ = INKEY$
40 IF a$> CHR$ 127 THEN GO TO 30
50 IF a$ = CHR$ 6 THEN POKE 23658,8-PEEK 23658: GO TO 30
60 PRINT #4;a$;
70 GO TO 30
```

It works by scanning the keyboard (line 30), ignoring tokens (eg STOP) that are non-standard (40), and using CAPS LOCK to shift between C and L modes (line 50). If it is valid, it gets sent to stream 4, the receiving computer. Some computers may react strangely to some of the Spectrum's codes, such as INV VIDEO, so additional testing may be necessary at the Spectrum end.

A further application could be to use the Spectrum as a display terminal, printing all RS232 input to it:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 LET a$ = INKEY$ #4
40 IF a$ = "" THEN GO TO 30
50 IF a$ = CHR$ 12 THEN CLS: GO TO 30
60 PRINT a$;
70 GO TO 30
```

When using RS232 for input, pressing the SPACE key alone will **Break** — CAPS SHIFT is not needed.

It is possible to link two Spectrums directly together via RS232 via a suitable lead, but there are few advantages when compared to Networking which is much easier to accomplish, as it does a similar thing more quickly while leaving the very useful RS232 port free for peripherals.

RS232 Additions to Stream Lister

The following lines should be added to the Stream Lister program to allow it to handle RS232 channels:

Stream Lister 3

```
1640 IF F$="T" THEN GO TO 3000
2999 REM Channel T
3000 PRINT INVERSE 1;"RS232  ";
3010 IF FN P(D+5)=3132 THEN PRIN
T "Text": GO TO 3040
3020 IF FN P(D+5)=3162 THEN PRIN
T "Binary": GO TO 3040
3030 PRINT "(Unknown)"
3040 GO SUB 1800
3050 PRINT "Baud rate:";INT (350
0000/((FN P(23747)+2)*26));" (ap
prox)"
3060 GO TO 1500
```

Both types of RS232 channels have the same specifier in memory — "T", so lines 3010 and 3020 work out if it is Text or Binary. The baud rate is also printed.

CHAPTER 7
Using the Network

Networking is a method by which many computers can be linked together so that they communicate quickly between each other. On the Spectrum this speed is over 3K bytes per second, which is much faster than the greatest RS232 baud rate of 19200. Up to 64 Spectrums may be connected together, and there are several uses of the facility — the first is in classrooms, where there can be a number of Spectrums linked together so that they can 'share' peripherals, like printers and Microdrives. Another use is for connecting two friends' computers together so that they can quickly swap programs and data. A further use is for program development, particularly machine-code. One Spectrum can have the Assembler and toolkit programs in it, which then downloads the development program into the second Spectrum. The advantage is that if the second one crashes because of an error, then it can quickly have a modified version reloaded from the first.

To use the Network (or *net*) streams and channels are used, with the channel specifier "N" (or "n"), for Network. Before using it, you must decide on a number for your particular Spectrum (or *station*). This is set up with the command

FORMAT "n";t

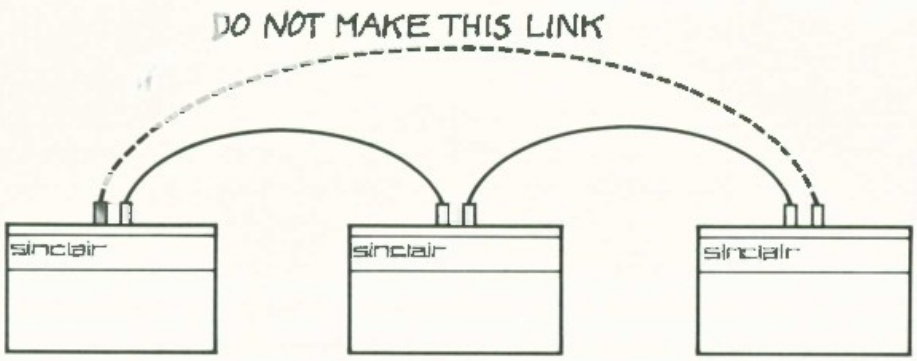
where t is your desired number, from 1 to 64. When the Spectrum is switched on, it is set to number 1. If there are only two Spectrums on the net, a FORMAT is not required on either, as both can be station number 1. If you forget which station number you are, the function

PEEK 23749

will give you your number.

To connect the stations together, use the leads supplied with each Interface, and plug them into the sockets on it, under the cassette sockets, forming a chain. There should always be one spare lead left over, as you should not form a loop — **Figure 5** shows the connection details for three stations.

Figure 5. An example Network



Sending programs over the Net

As with Microdrives and RS232, programs and data can be sent via the Network using commands similar to those used by RS232, but with the "N" specifier, followed by the station number. For example, if you are at station 10 and a friend who would like your program is at station 20, you would type

```
SAVE *"n";20
```

and your border will go black while it waits for your friend's station 20 to accept the program. Your friend should then type

```
LOAD *"n";10
```

to accept the program. When he does this, both stations' borders will flash while the program is transferred. If he had typed LOAD first, his Spectrum would have waited for you to type the SAVE, and the program would have been transferred perfectly. For reasons we shall see later on, it is good practice for the receiver to type LOAD before the sender types SAVE.

If your friend wishes to confirm that the program has been transferred correctly, he should type

```
VERIFY *"n";10
```

and you should type

```
SAVE *"n";20
```

again in either order. All the usual functions can be added to the commands eg

```
SAVE *"n";10 LINE 10
LOAD *"n";20 SCREEN$
```

Sending data over the Net

To send data over the net to another station, streams and channel "N" are used. To set up a stream, the command

OPEN #s;"n";x

is used where s is the stream number (0–15) and x is the other station's number, from 0 to 64 (0 has a special function which we shall see later). This creates a *buffer*, similar to the one used with "M" type channels, but about half the size — 255 characters to be exact. When a net stream is opened, it can be either a read or a write file, but not both. You determine which it is by the first action that is done with it — if you PRINT # into it, it becomes a write file, or if you read from it (using INPUT # or INKEY$ #) it becomes a read file. As with Microdrives, CLOSE # commands are vitally important with write files — if you don't CLOSE the last remaining data in the buffer will not be sent and, more importantly, the receiving station may wait forever for it!

While any network communications are going on (ie the border is flashing) the CAPS key is not required to **Break** — pressing SPACE only will suffice. A Spectrum should never be switched on or off while communication is going on.

Station 0 — Broadcasting

Normally, when sending or receiving data to another station, 'handshaking' is performed — this means in effect that if the receiving station is not ready for it, the sender will wait until it is, and the data will not be lost. However, a very useful facility has been included in the Interface — that of *broadcasting*, using station number zero, which does not obey the above rule.

When you send data to 'station 0', it is sent to all the Spectrums connected to the net, but does not wait for any of them to be ready. If they are waiting for input from station 0, then they will get the data, but if they are not then it is 'lost'. When transmitting programs over the net to several stations, every receiving station should enter

LOAD *"n";0

and their borders will go black as they wait for the program. Then the station with the program should enter

SAVE *"n";0

and all the 'listening' stations will load the program. When you broadcast anything, there is no easy way to tell who, if anyone, has received it. However, if you read something from the net, to find out which station it came from, you can

PEEK 23759

which will give the sender's station number.

The printer & Microdrive server program

A great advantage of networking is that many Spectrums can share peripherals, such as printers and Microdrives. There follows two programs to enable each 'slave' station (such as pupils') to use the printer and Microdrives at the 'master' station, (such as a teacher's). They have been written to allow the peripherals to be used as easily as possible, with commands such as SAVE, and LPRINT. One program is for the master station, and the other for each slave. To begin with, each station should load the slave program, and the easiest way is for the master station to broadcast it to every other one. The master program should then be loaded into the station with the Microdrives and printer, and the program RUN. When the machine-code has been POKEd in, the border will go black, and the system is ready.

Each slave station can FORMAT itself to be any station number except 32 and 64. Programs can be written and debugged at the slaves, so long as line numbers are kept to under 9000, to avoid clashing with the slave program. If a slave wishes to use one of the master peripherals he should enter **GO TO 9000**. A menu will be presented and the user can then use the printer, save and load programs, or catalogue his programs.

If he chooses option 1, he is connected via the master to any printer connected to it — either ZX, or an RS232 type. He can then use LPRINT, LLIST etc, until he has finished, when he should CLOSE #3. Option 2 allows him to SAVE a program of any type of file (eg SCREEN$) onto a cartridge at the master station. Option 3 allows him to LOAD any file saved with option 2 from the master's cartridge. Option 4 prints a catalogue of all the user's programs, and option 5 is to exit.

Slave program listing

```
9000 CLS
9010 PRINT '"  1. Use printer"
9020 PRINT '"  2. SAVE a program
"
9030 PRINT '"  3. LOAD a program
"
9040 PRINT '"  4. CAT your cartr
idge"
9050 PRINT '"  5. Quit"
9060 PAUSE 0: LET a$=INKEY$
9070 IF a$="1" THEN GO TO 9150
9080 IF a$="2" THEN GO TO 9300
9090 IF a$="3" THEN GO TO 9400
9100 IF a$="4" THEN GO TO 9500
```

```
9110 IF a$<>"5" THEN GO TO 9060
9120 GO TO 9000
9147 REM ********************
9148 REM * Send to printer *
9149 REM ********************
9150 LET a$="print": GO SUB 9200
9160 CLOSE #3: OPEN #3;"n";64
9170 PRINT "Printer is ready. Us
e LPRINT,   LLIST etc. When fini
shed do     CLOSE #3"
9180 GO TO 16000
9197 REM ************************
9198 REM * Broadcast a$ & wait *
9199 REM ************************
9200 PRINT INVERSE 1;"Contacting
 Master station now."
9205 CLOSE #4: OPEN #4;"n";0
9210 PRINT #4;a$
9220 CLOSE #4
9230 OPEN #4;"n";64
9240 IF INKEY$#4="" THEN GO TO 9
205
9250 CLOSE #4
9260 PRINT "Contact made."
9270 RETURN
9297 REM ********************
9298 REM * SAVE a program *
9299 REM ********************
9300 LET a$="save": GO SUB 9200
9310 INPUT "Filename to SAVE:";
LINE f$
9320 OPEN #4;"n";64
9330 PRINT #4;f$
9340 CLOSE #4
9350 PRINT "Microdrive ready. Do
            SAVE *""n"";64 etc"
9360 GO TO 16000
9397 REM ********************
9398 REM * LOAD a program *
9399 REM ********************
9400 LET a$="load": GO SUB 9200
9410 INPUT "Filename to LOAD:";
LINE f$
9420 OPEN #4;"n";64
```

```
9430 PRINT #4;f$
9440 CLOSE #4
9450 OPEN #4;"n";64
9460 INPUT #4;st
9470 CLOSE #4
9480 IF st<>6 THEN PRINT "File n
ot found": STOP
9490 PRINT "Microdrive ready. Do
            LOAD *""n"";64 etc"
9495 GO TO 16000
9497 REM **************
9498 REM * CATalogue *
9499 REM **************
9500 LET a$="cat": GO SUB 9200
9510 CLS
9520 PRINT "CATALOGUE:"
9530 MOVE "n";64 TO #2
9540 PRINT ' FLASH 1;"Press any
key to continue"
9550 PAUSE 0
9560 GO TO 9000
```

Master program listing

```
   1 REM ************************
   2 REM Master station program
   3 REM ************************
  10 CLEAR 65089
  20 LET st=65260: GO SUB 8000
  30 LET open=65090: GO SUB 8350
  40 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
 100 FORMAT "n";64
 110 CLOSE #4
 120 OPEN #4;"n";0
 130 INPUT #4; LINE a$
 140 CLOSE #4
 150 LET n=PEEK 23759
 160 IF a$="print" OR a$="save"
OR a$="load" OR a$="cat" THEN GO
 TO 180
 170 GO TO 120
 180 OPEN #4;"n";n
 190 PRINT #4
```

```
200 CLOSE #4
210 IF a$="save" THEN GO TO 300
220 IF a$="load" THEN GO TO 600
230 IF a$="cat" THEN GO TO 800
237 REM ********************
238 REM * Printer needed *
239 REM ********************
240 PRINT "Printer being used b
y station ";n
250 MOVE "n";n TO #3
260 LPRINT '''''''
270 PRINT "(Printing finished)"
280 GO TO 110
297 REM ****************
298 REM * SAVE needed *
299 REM ****************
300 OPEN #4;"n";n
310 PRINT "SAVING for station "
;n
320 DIM f$(10): INPUT #4; LINE
f$
330 CLOSE #4
340 GO SUB 500: LET f$(10)=CHR$
n
350 FOR i=1 TO 10
360 POKE USR "a"+i-1,CODE f$(i)
370 NEXT i
380 POKE 23766,d: POKE 23768,5
390 LET a=USR open: CLOSE #5
400 IF a<>1 THEN ERASE "m";d;f$
410 OPEN #5;"m";d;f$
420 POKE FN p(5*2+23566+8)+FN p
(23631)-1+67,4
430 MOVE "n";n TO #5
440 CLOSE #5
450 PRINT "(Program SAVED)"
460 GO TO 110
497 REM ************************
498 REM * calculate d from n *
499 REM ************************
500 LET d=1
510 RETURN
597 REM ********************
598 REM * LOAD a program *
```

```
 599 REM ********************
 600 OPEN #4;"n";n
 610 DIM f$(10): INPUT #4; LINE
f$
 620 CLOSE #4: LET f$(10)=CHR$ n
 630 PRINT "LOADing for station
";n
 640 GO SUB 500
 670 FOR i=1 TO 10
 680 POKE USR "a"+i-1,CODE f$(i)
 690 NEXT i
 700 POKE 23766,d: POKE 23768,5
 710 LET a=USR open
 720 OPEN #4;"n";n
 730 PRINT #4;a
 740 CLOSE #4
 750 IF a<>6 THEN GO TO 770
 760 MOVE #5 TO "n";n
 770 CLOSE #5
 780 PRINT "(Program loaded)"
 790 GO TO 110
 797 REM ********************
 798 REM * CAT a cartridge *
 799 REM ********************
 800 PRINT "CATalogue for statio
n ";n
 805 GO SUB 500
 810 LET z$="": CAT #14,d
 820 OPEN #4;"n";n
 830 LET z$=z$(13 TO )
 840 IF LEN z$<10 THEN GO TO 880
 850 IF z$(10)=CHR$ n THEN PRINT
 #4;z$( TO 9)
 860 LET z$=z$(12 TO )
 870 GO TO 840
 880 PRINT #4: CLOSE #4
 890 PRINT "(CAT finished)"
 900 GO TO 110
 910 REM
```

Note that the master program requires the OPEN #anything and Stream 14–z$ routines to be present from lines 7995–8510.

I will explain both programs together, so you can see what happens at both stations:

The first thing the master does is to set up the machine-code programs — 16K master stations should substitute

```
10 CLEAR 32489
20 LET st = 32490: GO SUB 8000
30 LET open = 32490: GO SUB 8350
```

The master must be station 64, so line 100 sets it up as such.

A slave must first make contact with the master, and this is done by broadcasting a word over the net, which the master acknowledges when it has noticed it. A slave's files are stored on a cartridge as nine letters of the name, and the 10th letter is CHR$ N, where N is the slave station number. This provides a method of keeping slaves' programs separate.

When a slave uses option 1 to request access to the printer, he repeatedly broadcasts "print", until a character is received from station 64. This contact-establishing is done by a subroutine at 9200, which opens stream 4 as a broadcast (9205), sends **a$** (9210), then closes the stream. It then sees if there has been any response from station 64 (9230–9240). If there has been no response, a$ is again broadcast. When there has been a response, the stream is closed (9250). Once contact has been established, line 9160 opens stream 3 so that it sends data to the master station. The **GO TO 16000** at 9180, and other lines, is to halt the program with the report **Program finished**.

The master program continually reads strings from the broadcast stream, using lines 120–140. It then checks to see if it was a request from a slave (160). If it was not, it continues to monitor the broadcasts (170), else the slave's number is calculated (150) and a response sent to it (180–200). Each request word is then tested, and an appropriate GOTO made (210–230). If the word "print" is broadcast, a suitable message is printed on the master's television, and the powerful MOVE command is used to transfer all input from the slave station to whatever device is connected to stream 3 (line 250). When the slave enters CLOSE #3 the MOVE command will finish, and another message printed. It then loops round to monitor the broadcast channel again.

When a slave chooses option 2 to SAVE his program, the word "save" is broadcast until the master responds (line 9300). He is asked for a filename for the program, which is sent to the master (9310–9340). A message is printed for him, and he can then do a SAVE *"n";64 followed by whatever the type of file is. If only programs are to be saved, substitute the line

```
9350 SAVE *"n";64: GO TO 9000
```

When the master reads the word "save" from a station on the broadcast channel it opens a stream to the slave and prints a message (300–310). Line 340 then uses the subroutine at 500 to calculate which Microdrive corre-

sponds to the calling station. In this version, drive 1 is always selected, but if there are a large number of slaves this may be impractical. For example, if there are 63 slaves, and the master has eight drives, each drive could be used by eight stations, and line 500 would be

```
500 LET d = 1 + INT(n/8)
```

Having found the drive number, the 10th character in the filename is set to mark the station whose program it is (340) and it is POKEd into the user-graphics area (350–370). This, in conjunction with line 380 set everything up to use the OPEN #any routine on stream 5, in line 390. If the required file already exists then it is ERASEd (400). Line 410 opens a write file with a suitable name, and 420 tricks the system into thinking it is not a data file. The program which is sent from the slave is written into file, again using MOVE (430). When the SAVE has finished, the stream is closed, and a message printed (440–450).

When the slave chooses option 3, LOAD a program, the word "load" is broadcast, and a response awaited (9400). A filename is then asked for, and sent to the master (9450–9470) and a check made to see if the file exists. If it doesn't, a suitable error is produced (9480). If it does exist, it can be loaded with

```
LOAD *"n";64
```

followed by whatever attributes are required.

If only programs are required, substitute the line

```
9490 LOAD *"n";64: GO TO 90000
```

When the master receives the word "load" (220) it reads the filename from the slave (600–620) and sets the 10th character to mark the source. The drive number is calculated (640) and the filename POKEd into the user-graphics area. Line 700 does more POKEs before using the OPEN # anything (710) to check for the presence of the requested program. The returned number is sent back to the slave station. Only if the file is of the correct type is it loaded from the drive and sent to the slave, using MOVE (760). The Microdrive stream is then closed and a message printed (770–780).

If a slave station chooses option 4 to catalogue his programs, the word "cat" is broadcast until an acknowledgement is received (9500). The screen is cleared and a title printed, then all input from the master station (ie the filenames) are printed, using MOVE (9530).

When the master station receives a "cat" request (230), then a message is printed and the drive number calculated (800–805). Using Stream 14–z$, a CATalogue is directed into z$ (810), and a stream to the slave opened (820). The title is removed from the string (830), and if there is a cartridge name

following then the tenth character is tested. If it is found to be CHR$ N ie the calling slave's program, then its name is sent to the slave (850). When there are no more filenames, the stream is closed and the message printed (880–890).

Further improvements

Some users may require further features in these programs. One improvement would be to store a list of passwords for each station (either in an array or a data file) so that a station pretending to be another could not destroy someone else's programs. If the names of each station's users were also stored, the printouts could have a heading page with the user's name on it to separate them.

Network additions to the Stream Lister program

The following lines should be added to the Stream Lister program to allow it to recognise Network streams. It will print the station number and the destination number, and indicate whether the broadcast feature is being used.

Stream 4 listing

```
1650 IF F$="N" THEN GO TO 3500
3499 REM Channel N
3500 PRINT INVERSE 1;"NETWORK"
3510 GO SUB 1800
3520 PRINT "Station number :";PE
EK 23749
3530 PRINT "transmitting to:";PE
EK (D+11);" (broadcasting)" AND
PEEK (D+11)=0
3540 GO TO 1500
```

CHAPTER 8
Machine-Language and the Interface

This chapter is written for those readers with a knowledge of Z80 machine-language, though readers who do not may still find some parts useful.

The 'Shadow' 8K ROM

In addition to the 16K BASIC ROM in the Spectrum, the Interface 1 has an 8K ROM that is paged into the same memory area ie from 0000–1FFFH (numbers followed by H are in hex). The very clever design of the paging mechanism has made it easy for the Spectrum's operating system to use the extra routines in the shadow ROM, but makes life rather trickier for machine-code buffs who wish to use them. Before delving deeper into the operating system and the ROM, the clever paging mechanism should be explained. To avoid any confusion, shadow ROM locations are preceded in this book by an 'X'.

The paging mechanism

With one exception, the shadow ROM is only paged in when the Z80 program counter reaches 0008 ie when a RST 8 occurs. In the 16K ROM, this restart is followed by a byte of data that determines which error message is to be produced while a BASIC program is running (or to produce a flashing question mark if the syntax of a line is wrong). If an attempt is made to use any of the additional commands (eg CAT) or added syntax (eg LOAD *) the 16K BASIC will force an error, using RST 8. With the Interface connected this then pages BASIC out, and pages the shadow ROM in. It then calculates what caused the error, and if it was an extended BASIC operation. If it was not, the normal error handling routine is used (in the 16K ROM). If it was an extended command, the shadow ROM then acts on it (by doing it or by checking its syntax), and then pages itself out after clearing the error that caused it to page in. It must be said that it is a brilliant method for adding commands without changing a single byte in the Spectrum's 16K ROM, but it does make it rather tricky to use the shadow ROM routines from one's own programs.

To help you over this obstacle, the shadow ROM authors have used what they called 'Hook codes' that allow a routine to have access to certain

shadow ROM routines. These hook codes are bytes of data that follow a RST 8 command, and range from 1BH to 32H inclusive. Codes FFH to 1AH produce the normal error messages, while codes over 32H produce a **Hook code error**, not documented in the manual.

Some of the hook codes are very useful, and others not so, but the programmer should be aware of the state of the Maskable Interrupts on entry and exit to some routines. If a return to BASIC is to occur (after a USR function) then the value of the H'L' register pair should always be preserved when using Microdrive routines. Failure to preserve it may crash the floating-point calculator, and hence the system.

Hook code 32H is perhaps the most useful, though it is officially designated as **Reserved for future expansion by SRL**. This is done to conceal the fact that it allows a user routine to call most shadow ROM routines, as well as allowing the shadow ROM to be paged in and out at will.

Some readers may be interested in disassembling the 8K ROM, but of course a PEEK or a LD A,(HL) will only read the 16K ROM. If a full disassembly is required the way I did it was to do the following program on a 48K Spectrum:

```
10 CLEAR 32767
20 SAVE *"m";1;"8K ROM"CODE 0,8192
30 LOAD *"m";1;"8K ROM"CODE 32768
```

When the SAVE* is executed, the shadow ROM is in place so it saves a copy of it to cartridge. Line 30 just loads it back 8000H above where it normally is. It can then be disassembled from location 8000H, but remember to subtract 8000H from all the absolute addresses.

Apart from RST 8, the other time the shadow ROM is paged is when PC = 1708H, which is the middle of the CLOSE # command routine. This is because it contains a fatal bug and it is rather inadequate to deal with the extra channel types used with the Interface.

How streams and channels work

In a similar way to BASIC, streams and channels are available to machine-code programs. Streams use the STRMS area of memory, and are linked to the channels which use the CHANS area. The STRMS area is fixed and lies from 5C10H to 5C35H inclusive, and consists of 19 pairs of bytes, one pair for each of streams FDH to 0FH. Each pair forms a 16-bit displacement, which is 0000 if the stream has no channel associated with it. If it is not zero, it is used to index into the CHANS area. When initialised, the CHANS area consists of 20 bytes, of four groups of five, with five bytes per channel. The CHANS area starts at the location pointed to by the system variable CHANS at 5C4FH, and finishes two bytes before PROG.

Initially the channel information contains data on four channels, with five bytes for each, in the order "K","S","R" and "P". The first sixteen

bytes of the STRMS area contain the bytes shown in the following table, with the selected channel:

STREAM NO.	STREAM LOCN.	CHANNEL DISP.	CHANNEL
FD	5C10	0001	"K"
FE	5C12	0006	"S"
FF	5C14	000B	"R"
00	5C16	0001	"K"
01	5C18	0001	"K"
02	5C1A	0006	"S"
03	5C1C	0010	"P"
04	5C1E	0000	none

To find the relevant channel data for a stream, the channel displacement for it is added to CHANS (after checking it is not zero), then decremented by one. The result then points to the start of the relevant data, which is at least five bytes. The first part of bytes point to the ROM output routine, the second pair to the input routine, and the fifth byte is the upper-case channel specifier. For example, channel "S" is stored as follows:

```
CHANS+5   DEFW 09F4H
          DEFW 15C4H
          DEFM "S"
```

In fact, 09F4H is used as the output routine for channels "K", "S" and "P", with different actions being taken depending on the channel specifier.

The Interface channels

To use any of the Interface features, the system requires 58 extra bytes for systems variables, to be inserted from location 5CB6H. This area must be set up before any shadow ROM or Hook code routines are used, and before any extra channel types are created (such as the stream14-z$ routine). The way to do this is with the instructions

```
CF RST 08
31 DEFB 31H
```

If the Interface is not connected, the non-existent error report "i" will be produced. If it is connected, the 58 bytes are inserted, and no error will occur. The variables are explained in Appendix A.

The extra system variables are normally created when

i) any error occurs (in syntax or execution)
ii) any Extended BASIC command is entered or executed
iii) a CLOSE # command is executed

Initially the variable VECTOR at 5CB7H is set to 01F0H, but this can be changed to allow extra commands to be added to the BASIC. This aspect of the Interface could fill a book in itself, though some details are given in a later chapter.

The variable IOBORD at 5CC6H is used to determine the border colour during certain I/O operations, namely RS232, Networking and when writing to a cartridge. If you prefer a certain border colour, you could POKE 23750, with the required colour. On the other hand, if you don't like the border flashing at all, you should

POKE 23750,INT(PEEK 23624/8)

which makes the I/O colour the same as the normal border colour.

The extra Interface channels "M","N","T" and "B" use an extended format for storing their details in the CHANS area.

"M" channel

This takes up 595 bytes in the CHANS area, and is addressed by the IX register in the shadow ROM. The byte allocation is as follows (all displacements in decimal):

0	DEFW 8	'output' routine
2	DEFW 8	'input' routine
4	DEFM "M"	channel specifier
5	DEFW 11D8H	shadow ROM output
7	DEFW 1122H	shadow ROM input
9	DEFW 595	length of this CHANS area
11	CHBYTE	current buffer position (0–512)
13	CHREC	position of record in file (0–255)
14	CHNAME	10 bytes of filename
24	CHFLAG	bit 0 — reset — read file set — write file bits 1–7 — unused (all set)
25	CHDRIVE	drive number (1–8)
26	CHMAP	drive MAP location
28	HPREAM	12 bytes of header preamble
40	HDFLAG	bit 0 — set to signal header bits 1–7 — unused (1,3,4,5 set, 2,6,7 reset)
41	HDNUM	sector number (0–255)
42	DEFW 0E31H	unused
44	HDNAME	10 bytes of cartridge name
54	HDCHK	header checksum
55	DPREAM	12 bytes of data preamble

67	RECFLG	bit 0 — reset to signal 'not a header' bit 1 — reset — no EOF set — EOF bit 2 — reset — a PRINT file set — not a PRINT file bits 3–7 — unused (all reset)
68	RECNUM	record number (0–255)
69	RECLEN	number of bytes in record (0–512)
71	RECNAM	10 bytes of filename
81	DESCHK	checksum of RECFLG to DESCHK – 1
82	CHDATA	first of 512 byte buffer
593		last byte of buffer
594	DCHK	checksum of buffer

Every Microdrive in use also has 32 bytes set aside for it in the Microdrive Maps area, from 5CF0H to CHANS – 1. Each block of 32 bytes is a bit-map of each sector on the cartridge. If a bit is set, then the sector is unusable, because it either contains data or is damaged.

"N" channel

This takes up 276 bytes in the CHANS area, and is addressed by the IX register in the shadow ROM. The byte allocation is as follows:

0	DEFW 8	
2	DEFW 8	
4	DEFM "N"	channel specifier
5	DEFW 0D6CH	shadow output routine
7	DEFW 0D0CH	shadow input routine
9	DEFW 276	length of this CHANS area
11	NCIRIS	destination station number (0–64)
12	NCSELF	this station number (0–64)
13	NCNUMB	block number (0–65535)
15	NCTYPE	packet type — 0-data, 1-end of file
16	NCOBL	bytes in data block (or 0 if output)
17	NCDCS	data checksum
18	NCHCS	header checksum
19	NCCUR	input buffer position

20	NCIBL	number of bytes in input buffer (0 if output)
21	NCB	start of 255 byte input buffer

"T" and "B" channels

These each occupy 11 bytes in the CHANS area, the minimum for channels other than the standard ones. The byte allocation is:

0	DEFB 8	
2	DEFB 8	
4	DEFB "T"	specifier (for "b" and "t")
5	DEFW 0C3CH (T) or 0C5AH (B)	shadow output
7	DEFW 0B6FH (T) or 0B75H (B)	shadow input
9	DEFW 11	length of this CHANS area

When using SAVE * etc., MOVE, FORMAT and ERASE from BASIC or machine-code, 'temporary' channels are set up. These are similar to the above channels, but with four important differences:

i) the specifier byte has the upper bit set
ii) they are never connected to a stream
iii) they are deleted when the command has finished, or when any error occurs
iv) they are always at the end of CHANS — a permanent channel should never lie higher in memory than any temporary channel.

With channels other than the standard ones, the minimum number of bytes in the CHANS area per channel is 11, as used by "T" and "B" channels. The first two pairs of bytes, usually the output and input routine pointers, are both 0008, and cause the shadow ROM to page in. It then detects that a channel routine is needed, and then uses the pair of bytes located five further on in the selected CHANS area ie the shadow ROM routine. When each routine finishes, the shadow ROM pages itself out.

The pair of bytes in the 9th and 10th positions are a 16-bit number that tells the system the number of bytes the channel occupies in the CHANS area. They are an extremely important pair of bytes, as they are used when any error occurs to step through the CHANS area to search for temporary channels to CLOSE. If they are omitted, or incorrect, the system will crash on the first error that occurs.

Using streams

To use a stream for input or output, the STRM_OPEN routine at 1601H is used. (It is in no way similar to the BASIC OPEN # statement). On entry, the accumulator should contain a valid stream number (FDH to 0FH). From it, the relevant displacement in STRMS is found, and an **Inva-**

id stream error produced if it is zero. If it is not zero, the relevant location in CHANS is calculated, and its value stored in the system variable CURCHL at 5C51H. The value of CURCHL is also set by all the machine-code OPEN routines.

To output characters to the current stream, load the accumulator with the code, and call the OUTPUT routine at 0010H with a RST 10 instruction. Registers BC,DE and HL are unaltered.

To input characters from the current stream, call the INPUT routine at 15E6H. Registers BC,DE and HL will remain unaltered. On return, the carry flag will be set if A holds a valid read character. The zero flag will be set if no character has been read, else carry and zero will both be reset if an **End of file** situation has occurred.

Using channels

There is, unfortunately, no simple machine-code equivalent to the OPEN # statement. For experimenting, the required streams could be opened from BASIC, and used from machine-code, but this is probably unsatisfactory for final use. The machine-code equivalents for each type of OPEN # statement are given later in the chapter, in the relevant section. The machine-code equivalent to CLOSE # is as follows:

```
LD A,stream
RES 1,(1Y + 124)      ;signal 'not a CLEAR #'
LD HL,1718H           ;(shadow ROM CLOSE routine)
LD (HD_11),HL
RST 8
DEFB 32H
RET
```

This uses hook code 32H to call the shadow ROM CLOSE # routine.

ROM routines

There now follows a list of shadow ROM routines to use the Interface facilities. Details are given on how to use the routine (usually by hook codes), and Entry and Exit conditions of registers and interrupts. Also given are the registers whose values are altered by the routine, as well as the actual location of the routines.

Microdrive routines

OPEN_M — create a temporary "M" channel in the CHANS area

```
To use      :RST 8
            DEFB 22H
Entry       :INTs on
```

Exit :IX = CURCHL = start of area, HL = stream displacement (= IX − CHANS + 1), INTS on
Regs :AF,BC,DE,HL,B'C',D'E',H'L',IX
Location :XIB29H

Operation: Before using this, the system variable D__STR1 should contain the drive number, N__STR1 the length of the filename, and T__STR1 should point to the start of the filename. A 595 byte area is first inserted at the end of the CHANS area (memory space permitting) and the relevant attributes copied into it. If a Map area does not exist for the chosen drive, one is created and filled with FFS. The drive is then operated and the cartridge searched for the filename. Bit 0 of IX + 25 (CHFLAG) is reset if it is a PRINT file, else it is set (for program, bytes etc). Note that the write-protect tab is not checked. To do that, after the routine do

```
IN A,(EFH)
AND 1              ;Z if protected
```

Note that the drive motor will not be switched off by this routine. The channel created is a temporary one. To make it permanent and incorporate it into a stream, do the following:

```
LD A,stream
ADD A,A,
LD HL,5C16H
LD E,A
LD D,0
ADD HL,DE
PUSH HL            ;save the stream location
RST 8
DEFB OPEN__M
PUSH HL            ;save the stream displacement
XOR A
RST 8
DEFB MOTOR         ;switch motors off
POP DE
POP HL
LD (HL),E
INC HL
LD (HL),D          ;store new data in STRMS
RES 7,(IX + 4)     ;make it permanent
RET
```

Author's note: Apparently hook code 2BH was supposed to do the above, but due to a typing error in the shadow ROM source code it jumps to the same routine as hook code 22H!

MOTOR — turn one drive motor on or switch all off

To use	:RST 8
	DEFB 21H
Entry	:A = 1 to 8 to turn drive on, A = 0 to turn all off
Exit	:INTs off if A<>0, or on if A = 0
Regs	:AF,BC,DE,HL
Location	:X17F7H

Operation: If the accumulator is not zero, the relevant Microdrive motor is turned on and all the others off, and interrupts are disabled. If A is zero, all drives are turned off and interrupts enabled. If the selected drive is not connected, the error **Microdrive not present** will occur.

CLOSE_M — close a "M" channel

To use	:RST 8
	DEFB 23H
Entry	:IX = start of "M" channel area
Exit	:INTs on
Regs	:AF,BC,DE,HL
Location	:X12A9

Operation: If the chosen channel was a write file, the remaining contents in the buffer are sent. The drive motor is switched off, and the 595 byte area is reclaimed. The relevant Map area is also reclaimed if it is not used by another channel.

ERASE_M — erase a Microdrive file

To use	:RST 8
	DEFB 24H
Entry	:none
Exit	:INTs on
Regs	:AF,BC,DE,HL,B'C',D'E',H'L',IX
Location	:X1D6EH

Operation: Before using this, the system variables D_STR1, T_STR1 and N_STR1 should be set up to the required drive number and string of the file to be deleted. A temporary "m" channel is created and the cartridge searched for the file. If it is found it is erased (though only one copy of it is deleted if there are multiple copies) provided that the write-protect tab is present. If it is not present, an error will occur. When it has finished, the motor is turned off and the temporary channel deleted.

RECLAIM_M — reclaim a 595 byte "M" area

```
To use      :RST 8
             DEFB 2CH
Entry       :IX = start of "M" channel area
Exit        :none
Regs        :AF,BC,DE,HL
Location    :X10C4H
```

Operation: First of all the 595 byte area is reclaimed, and any streams pointing to it are closed. The 32 byte Map area is reclaimed only if it is not used by another channel.

CAT — catalogue a Microdrive cartridge

```
To use      :LD HL,1C58
             LD (HD__11),HL
             RST 8
             DEFB 32H
Entry       :INTs on
Exit        :None
Regs        :AF,BC,DE,HL,A'F',B'C',D'E',H'L',IX
Location    :X1C58H
```

Operation: Before use, S__STR1 should contain the stream number for the catalogue, and D__STR1 the drive number. As there is no hook code for CAT, code 32H is used in conjunction with HD__11. Firstly, the chosen stream is opened (using 1601H), and then a temporary "M" channel created. Each header on the cartridge is examined, and the filename stored in the 512 byte buffer in the CHANS area, unless it is already there or if it starts with CHR$ 0. It is inserted into the correct alphabetical position in the buffer. When the whole cartridge has been completely read, or when 50 filenames have been found, the drive is switched off, and the cartridge name printed on the chosen stream. Each filename is then extracted from the buffer, and printed, and finally the number of kilobytes free is printed. This last number is calculated from half the number of clear bits in the relevant Map area. Lastly the "M" area is reclaimed.

FORMAT__M — FORMAT a cartridge

```
To use      :LD HL,1B6EH
             LD (HD__11),HL
             RST 8
             DEFB 32H
Entry       :none
Exit        :INTs on
Regs        :AF,BC,DE,HL,B'C',D'E',H'L',IX
Location    :X1B6EH
```

Operation: Before using this, D__STR1 should contain the drive number, and N__STR1 & T__STR1 point to the desired cartridge title. Initially, a temporary "M" area is created, and the drive turned on. After a short delay, the write-protect tab is tested, and an error produced if required. The buffer area is filled with FCH, and the buffer is repeatedly written to the cartridge with a numbered header and an 'invisible' filename. When the whole cartridge has been written to, the whole thing is checked several times. Finally the motor is stopped, and the "M" area reclaimed.

Cartridge data format

Before explaining the more advanced Microdrive routines the actual layout of data on the tape should be explained. A cartridge is divided into about 180 sectors, each holding 512 bytes of data. The routines actually cater for 256 sectors, but sectors 0 and those above around 180 either do not exist, or are never used. In addition two or three sectors are unusable as they lie where the join in the tape loop is.

Before each sector, there is a header consisting of 12 bytes of preamble (which is ten zeros and two FFs), then 15 bytes of the variables HDFLAG-HDCHK. Each header has a different number (HDNUMB) from 1 to about 180, stored sequentially on the cartridge.

Following the header is the sector itself, consisting of 12 bytes of preamble (as above), the 15 bytes of variables (RECFLG to DESCHK), then the 512 bytes of data, and finally a checksum byte (DCHK). A sector is unused if bit 1 of RECFLG and bit 1 of RECLENhi (IX + 70) are both reset.

As most files are too big to fit in a sector, they are chopped into 512-byte records, and each record is numbered sequentially, starting from zero. If a sector contains less than 512 bytes, bit 1 of RECFLG is set to show that it is the last one. Non — PRINT type FILES, such as programs, store their attributes in the first nine bytes in record number 0. These attributes are HD_00 to HD__11, explained in Appendix A.

The remaining Microdrive hook codes are as follows:

READ__P — read a record from a PRINT file

To use	:RST 8
	DEFB 27H
Entry	:IX = start of "M" channel area
Exit	:INTs off, motor on
Regs	:AF,BC,DE,HL
Location	:X1A17H

Operation: Before using this routine, CHDRIV should contain the drive number, CHREC the record number required, and CHNAME the filename. Firstly, the chosen drive is turned on, and SECTOR set to 04FB (= 5 complete revolutions). The cartridge is then searched for the chosen

record number of the chosen file. If it is found and it is a PRINT file then the sector is read in to the buffer and a return made. If it is found not to be a PRINT file, then the area is reclaimed and **Wrong file type** occurs. IF the file record is not found after five searches of the tape, the error **File not found** will occur, and the area reclaimed if it was temporary.

READ_NP — read the next record of a PRINT file

To use	:RST 8
	DEFB 25H
Entry	:IX = start of "M" area
Exit	:INTs off, motor on
Regs	:AF,BC,DE,HL
Location	:X1A09H

Operation: This routine is similar to READ_P, and CHDRIV should contain the drive number, and CHNAME the filename. Firstly, bit 1 of RECFLG is checked to see if the current record in memory was the last one — if it was, then **End of file** will occur. If it was not the last one then the contents of CHREC are incremented, and then control continues into READ_P.

READ_S — read next sector

To use	:RST 8
	DEFB 29H
Entry	:IX = start of "M" area
Exit	:INTs off, motor on
Regs	:AF,BC,DE,HL
Location	:X1A86H

Operation: Before calling this set CHDRIV and CHNAME to suit. The next header and sector on the tape are read in to the channel area. If it is a PRINT file a return is made with the carry flag reset. If it is not a PRINT file, the buffer contents are obliterated and a return made with carry set.

READ_R — read a PRINT sector randomly

To use	:RST 8
	DEFB 28H
Entry	:IX = start of "M" area
Exit	:INTs off, motor on
Rega	:AF,BC,DE,HL
Location	;x1A4BH

Operation: The cartridge is searched for sector number CHREC. If it is found, the buffer is loaded in. If it is a PRINT file, the carry flag is cleared and a return made. If it is not a PRINT file then the buffer contents are obliterated and carry set. If the relevant sector is not found after one revolution, then **File not found** will occur.

WRITE_S — write a sector serially

To use	:RST 8
	DEFB 26H
Entry	:IX = start of "M" area
Exit	:INTs on, motor off
Regs	:AF,BC,DE,HL
Location	:X11FFH

Operation: Before calling, set up the variable CHBYTE, CHREC, CHNAME and CHDRIVE and the buffer area to suit. Firstly, the relevant motor is switched on, and a test for full cartridge made. The filename is then copied from CHNAME to RECNAM, and CHBYTE and CHREC copied to RECLEN and RECNUM. The checksums DESCHK and DCHK are calculated, and the area RECFLG to DCHK is written onto the first unused sector on the cartridge (provided it is not write-protected). The relevant bit in Map is then set, CHBYTE zeroed and CHREC incremented. Finally the motor is turned off.

WRITE_R — write a sector randomly

To use	:RST 8
	DEFB 2AH
Entry	:IX = start of "M" area
Exit	:INTs off, motor on
Regs	:AF,BC,DE,HL
Location	:X1A91H

Operation: Before calling, CHREC and CHDRIV should be set to the desired values, and CHNAME contain the name of the sector to be saved. The cartridge is searched once for the sector no. CHREC. If it cannot be found, **File not found** will occur. If it is found, and the write-protect tab is tested. If it is present the sector and data (RECFLG-DCHK) bytes are written to the cartridge, and the relevant bit in Map is set. Note that the Map is not tested before the write, so the test on it should be done by the user if desired.

RS232 Routines

Before using any RS232 routines, the extra system variables must be present, and a suitable baud rate selected. The former can be done with hook code 31H, and the latter by alerting BAUD at 5CC3H. For reference, the look-up table to convert each baud rate to a suitable value in BAUD lies from X0AEFH to X0B12H, and consists of nine pairs of bytes. The first pair is the baud rate, and the second pair the corresponding 16-bit value to go in BAUD.

There are no hook codes for opening or closing RS232 channels, only for character input and output, using the "B" channel.

232IN — read a byte from RS232

To use	:RST 8
	DEFB 1DH
Entry	:none
Exit	:carry set if read a byte, A = byte, INTs on
Regs	:AF,BC,DE,HL
Location	:X0B81H

Operation: Reads a byte from the RS232 port, unless it has to wait too long or if SPACE is pressed (when **Break into program** will occur). The carry flag is set if a byte has been read.

232OUT — send a byte from RS232

To use	:RST 8
	DEFB 1EH
Exit	:A = byte code
Entry	:INTs on
Regs	:AF,BC,DE,HL
Location	:X0C5AH

Operation: A byte is sent via the port at the correct speed with the necessary handshaking. If BREAK is pressed during the transmission, an error will occur.

If you wish to use the equivalent of the "T" channel for RS232 output, use hook code 32H to call X0C3CH, with A containing the character code.

OPEN_R — open an RS232 channel

```
To use      :LD HL,0B13H
            LD (HD_11),HL
            RST 8
            DEFB 32H
```

Entry	:INTs on
Exit	:DE = start of new CHANS area
Regs	:AF,BC,DE,HL
Location	:X0B13H

Operation: This creates an 11 byte area at the end of CHANS for a "B" or "T" channel. Normally it will create a "T" channel, unless L__STR1 contains "B", when it will create a "B" channel.

Networking routines

Networking is implemented on the Spectrum by a continuous connection between the ports on each station. Only one communication can occur at a time, and all data is transmitted serially (but, unlike RS232 there are no start or stop bits, and no parity bits). Data is divided into numbered packets, each of 255 bytes maximum. A packet is transferred in the following way: firstly, the sender checks to see if the network is already busy — if it is, then it waits. When it is not busy, it sends eight bytes of a header down the cable, composed of the system variables NTDEST and NTHCS. This header contains various information about the data following it, and in particular the sender's and receiver's numbers, as well as two checksums. Unless it is broadcasting, the sender then tests for an acknowledgement from the receiver, in the form of a single byte of the receiving station's number. If the byte is not received then the header is again transmitted. If broadcasting, no test is made for the response byte, and the data is transmitted regardless. The data section is not sent if there are no bytes in it ie NTLEN = 0, else the relevant number of bytes are sent. Another response byte is then tested for (unless it is a broadcast) and if one is not received then the process repeats, and the header is re-transmitted. There are four hook codes for using the Network, though one is unfortunately unusable.

OPEN__N — open a temporary "N" channel

To use	:RST 8
	DEFB 2DH
Entry	:none
Exit	:IX = DE = (CURCHL) = start of "N" area in CHANS
Regs	:AF,BC,DE,HL
Location	:X0EA9H

Operation: Before calling this routine, D__STR1 should contain the destination number, and NSTAT the Spectrum's own number. A 265 byte area is created at the end of the CHANS area, and data stored in it. The destination number is copied from D__STR1 to NCIRIS, the station number

from NSTAT to NCSELF, and the area from NCNUMB to the last character in the buffer is filled with zeros. The channel is made temporary by setting bit 7 of the channel specifier. The variable CURCHL is made to point to the start of the newly created area.

To make a permanent Network channel, use the following code (after setting D_STR1 and NTSTAT):

```
LD A,stream
ADD A,A
LD HL,5C16H
LD E,A
LD D,0
ADD HL,DE          ;HL = suitable location in STRMS
PUSH HL            ;save it
RST 8
DEFB OPEN_N        ;Open a temporary "N"
RES 7,(IX + 4)     ;make it permanent
LD HL,(CHANS)
EX DE,HL
AND A
SBC HL,DE
INC HL             ;HL = stream displacement
POP DE             ;DE = STRMS location
EX DE,HL
LD (HL),E          ;store the displacement in STRMS
INC HL
LD (HL),D
RET
```

CLOSE_N — close a "N" channel

To use	:RST 8
	DEFB 2EH
Entry	:(CURCHL) = start of "N" area in CHANS
Exit	:INTs on
Regs	:AF,BC,DE,HL,IX
Location	:X1A24H

Operation: If the channel is a write file (ie NCOBL<>0) then the remaining buffer contents are sent in a packet marked EOF. The 265 byte area is then reclaimed, but any streams connected to it are *not* closed. The "N" area should be either temporary, or the last channel in the CHANS area, else other streams or channels may become corrupted.

WRITE_N — send a packet over the Network

To use	:RST 8
	DEFB 30H
Entry	:A = 0 for data, 1 for EOF, IX = start of "N" area
Exit	:INTs on, A = destination number (Z if broadcasting)
Regs	:AF,BC,DE,HL
Location	:X0DB2H

Operation: Before calling this, variables NCOBL and NCNUMB, the buffer contents, and the header area (excluding the checksums) should be set to suit. On entry, the accumulator should normally contain 0, or 1 if the packet is the last in a sequence ie an EOF marker. The value is stored in NCTYPE, and the border colour set to IOBORD. A checksum of the buffer contents is stored in NCDCS, and a checksum of NCIRIS to NCDCS stored in NCHCS. Interrupts are disabled, and the packet sent. When the packet has been received (if not broadcasting) then NCNUMB is incremented, the border colour restored and interrupts enabled.

READ_N — (intended to) read a packet from the Network

To use	:RST 8
	DEFB 2FH
Entry	:IX = start of "N" area
Exit	:INTs on
Regs	:AF,BC,DE,HL
Location	:X1A31H

Operation: This hook code was supposed to read a packet of bytes from the Network, but is unusable due to an error at the end of it. The carry flag was intended to indicate whether a packet was read or not, but it gets corrupted as the exit is via the border restore routine, which destroys its previous value.

The best way to read characters from the Network is via the 16K ROM routine INPUT, at 15E6H, as mentioned previously. Remember to preserve CURCHL if you use any other channels between calls to it.

Miscellaneous hook codes

PAUSE — wait for a key to be pressed (or to repeat)

To use	:RST 8
	DEFB 1BH
Entry	:none
Exit	:INTs on, A = key code
Regs	:AF,BC,DE,HL
Location	:X19D9H

Operation: Interrupts are enabled and every 50th of a second the keyboard scan routine in the 16K ROM is called, until a key is pressed. The key value is read from LAST_K.

PRINT — print a character on the screen

To use	:RST 8
	DEFB 1CH
Entry	:A = character code, INTs on
Exit	:none
Regs	:AF,BC,DE,HL,A'F',B'C',D'E'
Location	:X19ECH

Operation: The variable SCR_CT is set to FF to enable automatic scrolling, stream FEH (channel "S") is opened and the character printed.

LPRINT — print a character on the ZX printer

To use	:RST 8
	DEFB 1FH
Entry	:A = character code, INTs on
Exit	:none
Regs	:AF,BC,DE,HL,A'F',B'C',D'E'
Location	:X19FCH

Operation: Stream 3 (channel "P" usually) is opened and the character printed.

SCAN_K — scan the keyboard matrix

To use	:RST 8
	DEFB 20H
Entry	:none
Exit	:NZ if key pressed
Regs	:AF,BC,DE,HL
Location	:X1A01H

Operation: The keyboard is scanned directly and if any key (including any shift keys) is held down then the Z flag is reset. Interrupts are not required.

NEWVARS — create the extra system variables

To use	:RST 8
	DEFB 31H
Entry	:none
Exit	:none
Regs	:AF,BC,DE,HL
Location	:X19A8H

Operation: If the extra 58 system variables (in Appendix A) are not in existence then they are created, memory permitting. (In fact routine X19A8H contains just the instruction RET — the extra variables are created by the RST 8 itself).

SHADOW — call any shadow ROM routine

To use	:RST 8
	DEFB 32H
Entry	:(HD_11) = location
Exit	:none
Regs	:Unspecified (depends on routine)
Location	:X19A4H

Operation: The shadow ROM routine pointed to by HD_11 is called. Officially designated **Reserved for future expansion by SRL** it is the most powerful hook code in the ROM. Only the value of the A register can be passed to the routine, though all the values are returned from it. It can even be used to actually page the BASIC ROM out, with the following code:

```
        :LD HL,PAGOUT
        :LD (HD_11),HL
        :RST 8
        :DEFB 32H
PAGOUT:POP HL           ;remove 0700H from stack
        :POP HL         ;remove PAGOUT from stack
          '             ;rest of program
          '
```

When you want to page the 16K ROM back, do a CALL 0700H.

Hook code summary

This is a summary of each hook code, its name, shadow ROM location and function.

CODE	NAME	LOCATION	FUNCTION
1B	PAUSE	19D9	Wait for a key to be pressed — value in A
1C	PRINT	19EC	print CHR$ A on stream FE (the screen)
1D	232IN	0B81	RS232 input in A — carry if valid
1E	232OUT	0C51	RS232 output — character code in A
1F	LPRINT	19FC	print CHR$ A on stream 3 (the printer)

20	SCAN_K	1A01	scan the keyboard matrix — NZ if any pressed
21	MOTOR	17F7	switch Microdrive motors on or off
22	OPEN_M	1B29	open a temporary "M" channel
23	CLOSE_M	12A9	close an "M" channel
24	ERASE	1D6E	erase a file from a cartridge
25	READ_NP	1A09	read in the next PRINT buffer
26	WRITE_S	11FF	send the "M" buffer to cartridge
27	READ_P	1A17	read in a PRINT buffer
28	READ_R	1A4B	search for random file
29	READ_S	1A86	search for serial file
2A	WRITE_R	1A91	write a block randomly
2B	OPEN_M	1B29	same as 22 — a mistake — FAULTY
2C	RECLAIM	10C4	reclaim a 595 byte "M" area
2D	OPEN_N	0EA9	open a temporary "N" channel
2E	CLOSE_N	1A24	close a "N" channel
2F	READ_N	1A31	read a Network packet — FAULTY
30	WRITE_N	0DB2	write a Network packet
31	NEWVARS	19A8	create the extra system vars if reqd
32	SHADOW	19A4	call the shadow ROM routine address by HD_11

Notes

There are certain things that a machine-code programmer should be aware of when using the Interface:

i) Only use the USR function in LET or RANDOMIZE statements. If it is used in extended BASIC statements, and an error occurs, the shadow ROM may crash the system.

ii) Beware of the state of interrupts on entry and exit to some shadow ROM routines. Interrupts should always be ON before returning to BASIC.

iii) Always preserve the value of H'L' when using Microdrive routines if returning to BASIC.

iv) Never use important cartridges when debugging Microdrive routines. It can be very embarrassing if you accidentally FORMAT the one containing all your source code! Write — protect tabs are no guarantee of safety from wild machine-code errors.

v) Make sure the extra system variables have been created, using hook code 31H.

vi) Do not use REM statements for Interface routines. The creation of the CHANS areas will move them about in the memory map while they execute, and will crash the system.

CHAPTER 9
Adding BASIC Commands

As well as the Extended BASIC commands added to the Spectrum by the Interface 1, it also allows the programmer to add his own. This is a very useful facility, present on many other computers, but always missing on the Sinclair range. The main reason for its previous omission is the fact that, until now, all commands on the Spectrum had to be keywords, and the character set has no gaps in it to add commands. What the Interface 1 does is to add the option of jumping to an added machine-code program if it finds a syntax error in a line, either when it is entered, or when it is executed.

This allows the syntax of most of the existing commands to be changed to alter their function, though the Interface commands cannot normally be changed. For non-Interface commands, there are several ways to add functions:

i) add characters in strange places eg **COPY #**
ii) change parameter types eg **POKE 30000,"x"**
iii) add parameters eg **PLOT x,y,"BANG"**
iv) leave out parameters eg **CIRCLE 10,30**
v) use E mode functions eg **LINE 10,30**

You can also use your own commands, typed out in full but preceded by a non-alphabetic character (eg **!REPEAT**). The non-alphabetic character is necessary to take the Spectrum out of K mode when typing in the line.

The keywords whose syntax cannot normally be altered are LOAD, SAVE, MERGE, VERIFY, CAT, FORMAT, OPEN, CLOSE, CLS, CLEAR and MOVE. I say normally because they can actually be altered if you don't mind a bit of finger exercise when you type them in! The way they can be modified is to precede them with a shifted character, such as "*". For the keywords this means typing the word, doing a cursor-left, typing "*", doing a cursor right, then typing the rest of the line!

To create your own commands, you must have a good knowledge of machine-code and of the Spectrum's operating system, and for such readers the second half of this chapter describes the methods. This first half contains a program which alters the syntax of the common Microdrive commands, and can be entered and used by any reader.

To inform the Interface that some new commands have been added, you have to do two POKEs which *must* be done in a multi-statement line. They should not be done before any Interface operations have been done after a NEW. The easiest way is to **CLOSE #**, which is really a dummy statement. To make the Interface revert to the normal commands only, you should

POKE 23735,240: POKE 23736,1

again in a single line.

I must admit that I am not a great fan of the syntax chosen for loading and saving programs to Microdrive. In my view it is unnecessarily unwieldy, and requires an awful lot of key presses, so I wrote the following program to add easier commands for them. They are all in the form of an asterisk followed by the relevant command letter (in upper or lower case) and details. To load a program called *test* from drive 2 the new command would be ***L"2test"** which is equivalent to **LOAD *"m";2;"test"**. Similarly, SAVE *"m", VERIFY and MERGE are supplemented by ***S, *V, *M** followed by a single string. The first letter in the string must be the drive number. Another added command is ***C**, equivalent to **CAT 1**, and ***C** followed by a number to catalogue the other drives on the screen. The final new command is ***run** which will load a program called *run* from drive 1, and is equivalent to **NEW** followed by RUN.

Extra commands listing

```
  10 REM ********************
  20 REM * Extend M syntax *
  30 REM ********************
  40 REM                for 16K
  50 CLEAR 65169:  REM 32401
  60 LET ex=65170: REM 32402
  65 LET x2=INT ((ex+124)/256):
LET x1=ex+124-256*x2
  70 RESTORE 200: LET c=0
  80 FOR i=ex TO ex+194
  90 READ a: LET c=c+a
 100 POKE i,a
 110 NEXT i
 115 IF c<>22003+2*(x1+x2) THEN
PRINT "Checksum error": STOP
 120 CLOSE #0
 130 POKE 23735,ex-256*INT (ex/2
56): POKE 23736,INT (ex/256)
 140 PRINT "New commands:"
 150 PRINT "*L LOAD""*S SAVE""
*M MERGE""*V VERIFY"
```

```
 160 PRINT "*C CAT"'"*RUN"
 170 STOP
 200 DATA 254,92,194,240,1,215,3
2,0
 210 DATA 246,32,254,115,40,22,2
54,108
 220 DATA 40,28,254,118,40,34,25
4,109
 230 DATA 40,36,254,99,40,38,254
,114
 240 DATA 40,62,24,222,253,203,1
24,238
 250 DATA 205,x1,x2,195,54,8,253
,203
 260 DATA 124,230,205,x1,x2,195,
165,8
 270 DATA 253,203,124,254,24,244
,253,203
 280 DATA 124,246,24,238,33,214,
92,54
 290 DATA 1,35,54,0,35,54,2,215
 300 DATA 32,0,254,13,40,7,254,5
8
 310 DATA 40,3,205,30,6,195,169,
4
 320 DATA 215,32,0,246,32,254,11
7,194
 330 DATA 40,0,215,32,0,246,32,2
54
 340 DATA 110,32,244,215,32,0,20
5,183
 350 DATA 5,195,149,10,62,77,50,
217
 360 DATA 92,215,32,0,215,140,28
,215
 370 DATA 24,0,223,202,35,7,215,
241
 380 DATA 43,33,11,0,167,237,66,
218
 390 DATA 76,6,120,177,202,76,6,
26
 400 DATA 214,48,254,1,56,27,254
,9
 410 DATA 48,23,50,214,92,175,50
```

```
,215
 420 DATA 92,11,19,237,67,218,92
,237
 430 DATA 83,220,92,215,24,0,195
,35
 440 DATA 7,231,4
```

I find these new commands much easier and faster to use, and have the program saved as "run" on my main program development cartridge. When entering this program, alter lines 50 and 60 to suit to change the new position of RAMTOP and the memory location where the program will go. The recommended locations are incompatible with the recommended locations of some other routines in this book. If you wish them all to be compatible then add the lines

50 CLEAR 64579: REM 31809 for 16K
60 LET mc = 64580: REM 31810 for 16K

Any extra commands will be disabled after a NEW, so you should set up an empty program file on cartridge called "new".Then, to delete a current program while retaining the commands do

***L"1new"**

How to add commands

This half of the chapter is for machine-code programmers. A knowledge of the Spectrum operating system would also be useful — for this I can recommend *The Complete Spectrum ROM Disassembly* by Dr Ian Logan, whose 16K ROM routine Labels I have used in this book.

Crucial to the extra command facility is the Interface variable VECTOR at 5CB7H. This normally contains 01F0H, which is a shadow ROM location that executes the normal error handler routine (after copying CHADD_ back to CH_ADD).

To explain how to alter VECTOR, the actions of the shadow ROM when an error occurs should be explained. The numbers in brackets refer to the ROM locations in hex.

Firstly HL is made to contain the return address (0013) (from the RST 8), and is checked to see if it is 0000 or 15FE. If it is 0000, then a 16K ROM routine has been called from the shadow ROM, or if it is 15FE then an Interface channel has been requested. If it was neither of these then the Interface variables are created if they don't yet exist, and certain signals are sent to the ULA in it. The error number is then found (00C7) (from the byte following the RST 8), and checked to see if it is a hook code byte. If it is not, it is checked to see if the error was **Nonsense in BASIC**, **Invalid filename**, or

Invalid stream. If it was none of these, then the old ROM error handler is used (00F8). If it was one of the above errors, then the contents of CH_ADD are stored in CHADD_ (what confusing mnemonics were chosen!). Bit 3 of FLAGS3 is checked — if it is set then the normal error handler is used. After calculating the first character in the line where the error occurred (0126–01E9) it is compared with the Interface tokens (01B5–01E9). If it is not recognised, then the routine pointed to by VECTOR is jumped to. Normally this produces an error, but if VECTOR is changed then you can add commands that fail the normal syntax check.

So, if you have changed VECTOR to point to your extra routine, what should the routine do? Firstly it should check the A register for the first token or character code of your chosen extra commands, less 206. If it is not recognised, then JP 01F0H to produce an error. If it is, you should jump to the command routine which should check its syntax, evaluate any parameters and execute it (the latter two only during run-time). There are a couple of things to be wary of when writing such a routine:

1. The shadow ROM is paged in, not the 16K BASIC.
2. All the Z80 restarts are different.
3. Do NOT attempt to use the hook codes — call the routines directly.
4. Even while interrupts are on, the keyboard is *not* scanned and FRAMES is not incremented.

The shadow restarts do the following:

RST 0 — resets FLAGS3 and returns to 16K ROM
RST 8 — NEVER USE
RST 10 — CALLs 16K ROM routines — followed by two data bytes of the location
RST 19 — BIT 7,(FLAGS) — Z if syntax check, NZ if running
RST 20 — do a shadow ROM error. Follow with a data byte:

FF	Program finished	00E7
00	Nonsense in BASIC	0139
01	Invalid stream number	0663
02	Invalid device expression	062D
03	Invalid name	064C
04	Invalid drive number	0681
05	Invalid station number	05F6
06	Missing name	068D
07	Missing station number	06A1
08	Missing drive number	0683
09	Missing baud rate	06B7
0A	Header mismatch error	(never used in ROM)
0B	Stream already open	052F
0C	Writing to a 'read' file	0D78
0D	Reading a 'write' file	0D1C

0E	Drive 'write' protected	128D
0F	Microdrive full	1219
10	Microdrive not present	1828
11	File not found	11A3
12	Hook code error	1985
13	CODE error	092E
14	MERGE error	07D8
15	Verification has failed	0930
16	Wrong file type	0902

(The above hex locations are where to jump to to produce each error).

RST 28 — do a 16K ROM error. Load ERRNR with the required report code prior to the RST.
RST 30 — create Interface variables if non-existent
RST 38 — interrupts on (this is why the keyboard is not scanned).

Line scanning routines

Whatever commands you add, you will have to scan the BASIC line for syntax or execution. Your command routine will be called twice for each line entered — once for syntax checking, and the other when it is executed. To tell which it is, use RST 18. There are a lot of line scanning routines in the old ROM that can be used, using RST 10. To scan the line character by character, use these old ROM routines:

GET_CHAR 0018H — A register is current byte in line.
NEXT_CHAR 0020H — A register is next byte in line, ignoring control codes and spaces.

The main ones for evaluating are:

CLASS_06 1C82H — check/evaluate numeric parameter and put value on calculator stack if run-time, else insert invisible bytes in the line.

CLASS_0A 1C8CH — check/evaluate string parameter and put value on calculator stack if run-time.

On entry to the latter two routines CH_ADD should point to the first character of the expression. On return, it points to the first 'out of place' character, with A containing its code (eg ",").

There are also some scanning routines in the shadow ROM which can be called directly.

PARAMS #0701 — the routine to check for *string;number <;string: parameters>* ie the characters following LOAD, SAVE etc. All parameters are passed

to D__STR1, N__STR1, S__STR1, L__STR1 and HD__00 to HD__11 during run- time.

EVALBC #061E — evaluate a 16-bit number, returned in BC and in D__STR1.

CHKEND #05B7 — checks for end of statement. If not then an error will occur. If it is an end, then a RET is made during run-time, else the return address is removed and control returns to the 16K ROM via 05C1.

CHKDRV 066D — checks content of D__STR1 to be in the range 1–8. If not, the **Invalid drive number** occurs.

CHKST$ #062F — evaluate a string — if runtime then checked to be under 11 characters, and parameters stored in N__STR1 and T__STR1.

When the whole syntax has been checked and found to be correct a jump to 05C1H should be made (unless an indirect return is made via CHKEND). When a a command has been successfully executed, a similar jump should be made. The stack and error code is cleared, and a return made to the 16K ROM.

As can be seen, adding commands is quite difficult, but well worthwhile. It opens the Spectrum up to all manners of toolkit-type programs, as well as offering the possibility of using the BASIC line editor for entering lines for other languages, or assemblers.

Here is the assembly listing of the extra commands added in the first half of the chapter. It is not position independent or self-relocating — the BASIC loader alters the CALL bytes automatically as it POKEs it in.

```
0010 ;                  Vector to alter syntax
0030          ORG   40000     debugging origin
0040          CP    5CH       check for "*" – 206
0050 ERROR    JP    NZ,01F0H  error if not
0060          RST   10H
0070          DEFW  0020H     call NEXT-CHAR
0080          OR    20H       make it lower case
0090          CP    "s"
0100          JR    Z,SAVE    jump if "s" or "S"
0110          CP    "l"
0120          JR    Z,LOAD    jump if "l" or "L"
0130          CP    "v"
0140          JR    Z,VERIF   jump if "v" or "V"
0150          CP    "m"
0160          JR    Z,MERGE   jump if "m" or "M"
0170          CP    "c"
0180          JR    Z,CAT     jump if "c" or "C"
```

```
0190            CP      "r"
0200            JR      Z,RUN           jump if "r" or "R"
0210            JR      ERROR           else do an error
0220 SAVE       SET     5,(IY+124)      signal SAVE
0230            CALL    VARS            evaluate the string
0240            JP      0836H           do the SAVE
0250 LOAD       SET     4,(IY+124)      signal LOAD
0260 DOIT       CALL    VARS            evaluate the string
0270            JP      08A5H           do LOAD/VERIFY/MERGE
0280 VERIF      SET     7,(IY+124)      signal VERIFY
0290            JR      DOIT            do it
0300 MERGE      SET     6,(IY+124)      signal MERGE
0310            JR      DOIT            do it
0320 CAT        LD      HL,5CD6H        HL=D__STR1
0330            LD      (HL),1          make it drive 1
0340            INC     HL
0350            LD      (HL),0          zero D__STR1 hi
0360            INC     HL
0370            LD      (HL),2          make the stream 2
0380            RST     10H
0390            DEFW    0020H           call NEXT-CHAR
0400            CP      13
0410            JR      Z,CAT2          jump if newline
0420            CP      ":"
0430            JR      Z,CAT2          jump if separator
0440            CALL    061EH           evaluate a number
0450 CAT2       JP      04A9H           do the CAT
0460 RUN        RST     10H
0470            DEFW    0020H           call next-char
0480            OR      20H             make it lower case
0490            CP      "u"
0500 ERR2       JP      NZ,0028H        error if not "u"
0510            RST     10H
0520            DEFW    0020H           call NEXT-CHAR
0530            OR      20H
0540            CP      "n"
0550            JR      NZ,ERR2         error if not "n"
0560            RST     10H
0570            DEFW    0020H           call NEXT-CHAR
0580            CALL    05B7H           call CHKEND
0590            JP      0A95H           load the "run" program
0600 VARS       LD      A,"M"
0610            LD      (5CD9H),A       make L__STR1="M"
0620            RST     10H
```

```
0630            DEFW  0020H       call NEXT__CHAR
0640            RST   10H
0650            DEFW  1C8CH       call CLASS__0A (string)
0660            RST   10H
0670            DEFW  0018H       call GET__CHAR
0680            RST   18H         check syntax
0690            JP    Z,0723H     skip this if syntax only
0700            RST   10H
0710            DEFW  2BF1H       call STK__FETCH (BC = length,
0720            LD    HL,11       DE = start)
0730            AND   A
0740            SBC   HL,BC
0750            JP    C,064CH     error if more than 11 chars
0760            LD    A,B
0770            OR    C
0780            JP    Z,064CH     error if empty string
0790            LD    A,(DE)      A = first character code
0800            SUB   "0"         A = drive number
0810            CP    1
0820            JR    C,INVDR     error if < 1
0830            CP    9
0840            JR    NC,INVDR    error if > 8
0850            LD    (5CD6H),A   store the number in D__STR1
0860            XOR   A
0870            LD    (5CD7H),A   clear D__STR1 hi
0880            DEC   BC          decrease the length
0890            INC   DE          increase the start position
0900            LD    (5CDAH),    store the length
                      BC
0910            LD    (5CDCH),    store the start
                      DE
0920            RST   10H
0930            DEFW  0018H       call GET__CHAR
0940            JP    0723H       check arguments then RET
0950 INVDR      RST   20H         do a new error
0960            DEFB  4           byte for "Invalid drive no"
```

As can be seen, I have jumped to some pretty obscure shadow ROM locations. These locations should not be called using hook code 32H, as they use line scanning routines and return to BASIC, not to the calling machine code.

Appendices

APPENDIX A
The Interface System Variables

As well as the normal system variables, the following 58 bytes are added to the memory map, with the following functions:

DECIMAL	HEX	NAME	USE
23734	5CB6	FLAGS3	bit 0 — set while doing Extended command bit 1 — set while doing a CLEAR # bit 2 — set when ERRSP altered bit 3 — set when networking bit 4 — set while LOADing bit 5 — set while SAVEing bit 6 — set while MERGEing bit 7 — set while VERIFYing
23735	5CB7	VECTOR	Used to extend interpreter — normally 01F0

23737	5CB9	SBRT	Routine used by shadow ROM to call 16K routines, of the form:
			LD HL,value
			CALL routine
			LD (5CBAH),HL
			RET

23747	5CC3	BAUD	RS232 baud rate — initially 000CH, 19200 — roughly (3500000/(26*rate)) – 2
23749	5CC5	NTSTAT	Network station number (1–64)
23750	5CC6	IOBORD	Border colour during I/O operations
23751	5CC7	SER_FL	RS232 workspace — 1st byte is a flag, second is an inputted character
23753	5CC9	SECTOR	Microdrive workspace — normally used to count sectors, starting from FFH or 04FBH

23755	5CCB	CHADD__	Temporary store for CH__ADD while checking Extended syntax of lines
23757	5CCD	NTRESP	Network response code — receiving stations number — the acknowledgement Start of 8 byte Network header:
23758	5CCE	NTDEST	Network destination station number
23759	5CCF	NTSRCE	Network source station number
23760	5CD0	NTNUMB	Network block number (0–65535)
23762	5CD2	NTTYPE	Network header type code — 0 data, 1 EOF
23763	5CD3	NTLEN	Network data block length (0–255)
23764	5CD4	NTDCS	Network data block checksum
23765	5CD5	NTHCS	Network header checksum (of NTDEST–NTDCS)
23766	5CD6	D__STR1	Start of first 8 byte file specifier: 2 byte drive number (1–8) or Network destination number or baud rate
23768	5CD8	S__STR1	Stream number (0–15)
23769	5CD9	L__STR1	Channel specifier (upper case)
23770	5CDA	N__STR1	Length of filename
23772	5CDC	T__STR1	Start of filename (normally in workspace)
23774	5CDE	D__STR2	Start of second 8 byte file specifier
23782	5CE6	HD__00	File type: 0 – program 1 – numeric array 2 – string array 3 – bytes
23783	5CE7	HD__0B	Length of data
23785	5CE9	HD__0D	Start of data
23787	5CEB	HD__0F	Program length (or array name)
23789	5CED	HD__11	Auto start line number (also used by Hook code 32H)
23791	5CEF	COPIES	Number of multiple copies made by SAVE — reset to 1 after a SAVE

Notes: FLAGS3 is normally zeroed whenever the shadow ROM pages out. It can be usefully addressed by IY + 124.

The parameters in the routine SBRT are POKEd in by the shadow ROM.

The RETurn goes to location 8, but there is always a 0000 entry next on the stack to differentiate it from an error.

The variable names HD__00 to HD__11 are taken from their directly equivalent ones used by the 16K ROM cassette routines, addressed by IX+0 to IX+11H.

APPENDIX B
Assembly Listings

This Appendix contains Z80 Assembler listings of the routines used in this book. They are based on the originals created with the Picturesque Editas assembler, which denotes hex numbers by following them with 'H'. The line numbers are used by the assembler, and are of no meaning here. Comments have been added down the right hand side. All ORGs are arbitrary, as most of the routines are either position independent or self-relocating. Those that are self-relocating use the facility that, on entry to them, the BBC register contains their start address.

Stream14–z$

This creates a new channel "Z", then attaches it to stream 14, which is assumed to be already closed. It is also assumed that the 58 extra system variables have already been created. This is why the command CLOSE # is included in the BASIC loader, as it does both these jobs.

```
0010            ;               "Print to string routine"
0030            ;               "routine to make #14"
0040            ;               "insert chars into z$"
0050            ;               "(c) A.Pennell 1983"
0060            PROG  EQU       5C53H
0070            VARS  EQU       5C4BH
0080                  ORG       40000
0090            SETUP LD        HL,(PROG)
0100            DEC   HL        HL = PROG - 1 = end of
                                CHANS
0110            PUSH  BC        save the start address
0120            PUSH  HL        save the new data area start
0130            LD    BC,11     11 bytes needed
0140            CALL  1655H     call MAKE_ROOM
0150            POP   DE        DE = new data start
0160            LD    HL,OUTCH-SETUP
0170            POP   BC        get SETUP back
0180            ADD   HL,BC     HL = OUTCH
0200            PUSH  DE        save the data start
0210            EX    DE,HL
```

```
0220            LD      (HL),E          store OUTCH in the new area
0230            INC     HL
0240            LD      (HL),D
0250            INC     HL
0260            EX      DE,HL
0270            LD      BC,LABEL-OUTCH
0280            ADD     HL,BC           HL = LABEL
0290            LD      BC,9            9 more bytes of data
0300            LDIR                    copy the data into CHANS
0310            POP     HL              HL = data start
0320            INC     HL
0330            LD      BC,(5C4FH)      BC = CHANS
0340            AND     A
0350            SBC     HL,BC
0360            LD      (5C32H),HL      store the STRMS displacement
                                        in STRMS for #14
0370            LD      BC,0
0380            RET                     return to BASIC with 0
                                        The remaining data to go in
                                        CHANS:
0390 LABEL      DEFW    15C4H           the 'input' routine
0400            DEFM    "Z"             the channel name
0410            DEFW    28H             'shadow ROM o/p'
0420            DEFW    28H             'shadow ROM i/p'
0430            DEFW    11              the number of bytes
                                        The output routine:
0440 OUTCH      PUSH    AF              save the character code
0450            LD      HL,(VARS)
0460 L1         LD      A,(HL)
0470            CP      5AH
0480            JR      Z,FOUND         jump if found Z$
0490            CP      80H
0500            JP      Z,0670H         'Var not found' if no more
0510            CALL    19B8H           call NEXT_ONE
0520            EX      DE,HL           HL = start of next variable
0530            JR      L1              loop round
0540 FOUND      INC     HL              HL = string length lo
0550            LD      C,(HL)
0560            INC     HL              HL = string length hi
0570            LD      B,(HL)          BC = string length
0580            INC     BC              increase length
0590            PUSH    BC              save the new length
0600            PUSH    HL              save the 1st character location
0610            ADD     HL,BC           HL = end of string
```

```
0620            CALL  1652H     call ONE_SPACE (make
                                room for character)
0630            INC   HL        HL = new space
0640            EX    DE,HL     DE = new space
0650            POP   HL        HL = 1st character location
0660            POP   BC        BC = new length
0670            LD    (HL),B    store the new length
0680            DEC   HL
0690            LD    (HL),C
0700            POP   AF        get the character code back
0710            LD    (DE),A    store the char in the string
0720            AND   A         clear carry (to prevent errors)
0730            RET             back to the O/S
0740            END
```

On EOF GOTO

This routine alters ERRSP to point to a suitable error handling routine.

```
0010 ;                  "ON EOF goto"
0020 ;                  "PIC code"
0040            ORG     40000
0050 SETUP      LD      HL,START-SETUP Alter ERRSP:
0060            ADD     HL,BC       HL = START
0070            EX      DE,HL       DE = START
0080            LD      HL,(ERRSP)  HL = ERRSP
0090            LD      (HL),E      store the new error routine
0100            INC     HL          in the relevant stack
0110            LD      (HL),D      postion
0120            RST     8           make sure the Interface
0130            DEFB    31H         variables are there
0140            LD      BC,0
0150            RET                 back to BASIC with 0
                                    Error handler:
0160 START      LD      HL,(ERRSP)  HL = stack location
0170            LD      A,(5C3AH)   A = error code
0180            CP      EOF         was it an EOF?
0190            JP      NZ,1303H    jump to ROM handler if it
                                    wasn't
0200            LD      E,(HL)      else DE = START
0210            INC     HL
0220            LD      D,(HL)
0230            PUSH    DE          put START on bottom of stack
0240            CALL    16B0H       clear workspace and editing
                                    areas
```

```
0250            RES   5,(IY+37H)  signal 'ready for a new key'
0260            CALL  0D6EH       clear the lower screen and
                                  open stream 0
0270            LD    HL,(5C45H)  HL = current line number
0280            LD    (5CC9H),HL  store it in SECTOR
0290            LD    DE,LINE     DE = line to jump to
0300            LD    HL,5C42H    HL = NEWPPC
0310            LD    (HL),E      store the new line number
0320            INC   HL
0330            LD    (HL),D
0340            INC   HL
0350            LD    (HL),1      make NSPPC = 1 ie 1st state-
0370            JP    1B7DH       ment jump to the ROM
0380 ERRSP      EQU   5C3DH
0390 EOF        EQU   7           eof report code
0400 LINE       EQU   1000        error line number
0410            END
```

OPEN #anything routine

This routine opens a specified stream to a Microdrive channel, but the file type is not restricted to data files. It is, basically, the OPEN # command routine with the error checks modified.

```
0010 ;                  "OPEN anything"
0030 ;
0040 ;
0050 OPENM      EQU   22H         hook codes
0060 CLOSM      EQU   23H
0070 MOTOR      EQU   21H
0080            ORG   40000
0090 START      LD    A,(5CD8H)   A = S_STR1 = stream number
0100            CALL  1727H       call STR_DATA1
0110            LD    HL,17
0120            XOR   A
0130            SBC   HL,BC
0140            LD    BC,0
0150            RET   C           back to BASIC with 0 if the
                                  stream is already open
0160            LD    (5CD7H),A   zero D_STR1 hi
0170            LD    HL,10
0180            LD    (5CDAH),HL  make filename length = 10
0190            LD    HL,(5C7BH)
0200            LD    (5CDCH),HL  make name start = User
                                  graphics area
```

```
0210            LD      A,(5CD8H)   A = stream number
0220            ADD     A           calculate the relevant STRM
0230            LD      HL,5C16H
0240            LD      E,A
0250            LD      D,0
0260            ADD     HL,DE       HL = stream location
0270            EXX
0280            PUSH    HL          save H'L'
0290            EXX
0300            PUSH    HL          save the STRM location
0310            RST     8           open a temp "M" channel
0320            DEFB    OPENM
0330            BIT     0,(IX+24)
0340            JR      Z,READ      jump if its a read file
0350            XOR     A           file not found so
0360            RST     8           all motors off
0370            DEFB    MOTOR
0380            RST     8           and reclaim the area
0390            DEFB    2CH
0400            POP     HL          restore the stack
0410            EXX
0420            POP     HL          get H'L' back
0430            EXX
0440            LD      BC,1
0450            RET                 return with 1 if not found
0460 READ       RES     7,(IX+4)    make it permanent
0470            XOR     A
0480            PUSH    HL          save the stream displacement
0490            RST     8           turn all motors off
0500            DEFB    MOTOR
0510            POP     DE          DE = displacement
0520            POP     HL          HL = location
0530            LD      (HL),E      store the new displacement
0540            INC     HL          in STRMS area
0550            LD      (HL),D
0560            LD      A,(IX+67)   A = RECFLG
0570            AND     4           mask the PRINT file bit
0580            ADD     2           add 2
0590            LD      C,A         put in BC
0600            LD      B,0
0610            EXX
0620            POP     HL          get H'L'back
0630            EXX
```

```
0640            RET             return with 2 if a PRINT file
                                or 6 if not
0650            END
```

Status routine

This routine is similar to the MOTOR routine in the shadow ROM, from locations X182AH to X1871H, but with the error handling modified. Its purpose is to determine whether a given Microdrive is present or not, and whether the cartridge in it is write-protected. The labels are based on the equivalent shadow ROM locations. The ORG is a debugging origin — the code is *not* position independent — the BASIC loader does the relocating of the four CALLs.

(I cannot claim that I know exactly what each I/O port instruction does — the notes here are informed guesswork!)

```
0010 ;                  STATUS routine
0020 MOTOR      EQU     21H             motor hook code
0030            ORG     40000           debugging origin
0040            LD      A,(5CD6H)       A = D__STR1 = drive number
0050            DI                      interrupts off
0060            JR      X182A           jump past this
0070 X1806      LD      HL,1388H        HL = 5000
0080 X1809      DEC     HL
0090            LD      A,L
0100            OR      H
0110            JR      NZ,X1809        wait a while
0120            LD      HL,1388H
0130 X1811      LD      B,6
0140 X1813      IN      A,(0EFH)        read port EF
0150            AND     4               bit 2 only
0160            JR      NZ,X1820        jump if drive not present
0170            DJNZ    X1813           make sure its there 6 times
0180            JR      PRESN           it is so return
0190 X1820      DEC     HL              decrement the counter
0200            LD      A,H
0210            OR      L
0220            JR      NZ,X1811        try it 5000 times
0230            LD      BC,0            it can't be connected so
0240            JR      NOTPR           return with 0
0250 X182A      LD      DE,0100H        D = 1, E = 0
0260            NEG                     negate the drive number
0270            ADD     9               A = 9 – drive number
0280            LD      C,A             C = drive selected
0290            LD      B,8             B = drive counter
```

```
0300 X1835      DEC   C           decrement the drive selected
0310            JR    NZ,X184B    jump if not the one that is
                                  under investigation
                                  Put a motor ON:
0320            LD    A,D
0330            LD    (0F7H),A    send 1 to port F7 — ON
0340            LD    A,0EEH
0350            OUT   (0EFH),A    send EE to port EF
0360            CALL  X1867       wait a while
0370            LD    A,0ECH
0380            OUT   (0EFH),A    send EC to port EF
0390            CALL  X1867       wait a while
0400            JR    X185C       do the next one
                                  Switch a drive OFF:
0410 X184B      LD    A,0EFH
0420            OUT   (0EFH),A    send EF to port EF
0430            LD    A,E
0440            OUT   (0F7H),A    send 0 to port F7 — OFF
0450            CALL  X1867       wait a while
0460            LD    A,0EDH
0470            OUT   (0EFH),A    send ED to port EF
0480            CALL  X1867       wait a while
0490 X185C      DJNZ  X1835       do all 8 drives
0500            LD    A,D
0510            OUT   (0F7H),A    send 1 to port F7
0520            LD    A,0EEH
0530            OUT   (0EFH),A    send EE to port EF
0540            JR    X1806       test its status

0550 X1867      PUSH  BC          Delay routine:
0560            PUSH  AF          save regs
0570            LD    BC,0087H    BC = 135
0580 X18FB      DEC   BC
0590            LD    A,B
0600            OR    C
0610            JR    NZ,X18FB    loop 135 times
0620            POP   AF          restore regs
0630            POP   BC
0640            RET               return
0650 PRESN      IN    A,(0EFH)    the drive is present so
0660            AND   1           test the write-protect tab
0670            LD    BC,1
0680            JR    NZ,NOTPR    jump if tab is there with 1
0690            INC   BC          else it is not there so 2
```

```
0700 NOTPR      PUSH  BC          save the return value
0710            XOR   A
0720            RST   8
0730            DEFB  MOTOR       switch all motors off
0740            POP   BC          restore the value
0750            RET               back to BASIC
0760            END
```

ON ERROR GOTO

This routine alters ERR_SP to point to a new error handler. This alone is not sufficient for Interface errors — for them, bit 2 of FLAGS3 must be set as well, else the normal error handler will be used. If the error occurred while the shadow ROM was paged in, HL will equal 81H.

```
0010 ;          ON ERR GOTO"
0020 ;          Interface version
0030 ERRSP      EQU   5C3DH       set constants
0040 SECTR      EQU   5CC9H
0050 LINE       EQU   1000        error line jump
0060            ORG   40000       arbitrary origin
0070 SETUP      LD    HL,START-SETUP
0080            ADD   HL,BC       HL = START
0090            EX    DE,HL       DE = START
0100            LD    HL,(ERRSP)  HL = error stack position
0110            LD    (HL),E      store the new error handler
0120            INC   HL          on the stack
0130            LD    (HL),D
0140            RST   8
0150            DEFB  31H         create the extra variables
0160            LD    BC,0
0170            RET               return with 0
                                  The new error handler:
0180 START      LD    (SECTR),HL  store the value of HL
0190            LD    HL,(ERRSP)
0200            LD    DE,1303H    DE = old ROM handler
0210            PUSH  DE          push it on stack to disable this
                                  function next time
0220            CALL  16B0H       clear various areas
0230            RES   5,(IY+37H)  clear the key bit in FLAGS
0240            CALL  0D6EH       clear the lower screen and
                                  open stream 0
0250            LD    HL,17B9H    HL = shadow routine to re-
                                  claim all temporary channels
                                  and switch all motors off
```

```
0260            LD      (5CEDH),HLstore in HD__11
0270            LD      A,(5C3AH)  A = error number (old error)
0280            LD      (23763),A  store in NTLEN
0290            PUSH    AF         save the error code
0300            RST     8
0310            DEFB    32H        call X17B9H
0320            LD      HL,0081H
0330            LD      DE,(SECTR) DE = value of HL after error
0340            POP     AF         A = old ROM error code
0350            AND     A          clear carry
0360            SBC     HL,DE
0370            JR      NZ,OLD     jump if HL on entry <>81H
                                   ie an old ROM error
0380            BIT     0,(IY+124) test FLAGS3
0390            JR      NZ,OLD     to an old error if in the middle
                                   of a statement
0400            CP      7
0410            JR      Z,OLD      jump if it was an EOF
0420            LD      A,100      Interface error so store
0430            LD      (23763),A  100 in NTLEN
0440            LD      A,"?"
0450            RST     10H        print a "?"
0460            JR      PRNTN      then print line numbers etc
                                   Old ROM error:
0470            INC     A
0480            LD      B,A        store the error no. in B
0490            CP      10
0500            JR      C,2
0510            ADD     7          form the report character
0520            CALL    15EFH      print it
0530            LD      A," "
0540            RST     10H        print a space
0550            LD      A,B
0560            LD      DE,1391H
0570            CALL    0C0AH      print the error message
0580 PRNTN      XOR     A
0590            LD      DE,1536H
0600            CALL    0C0AH      print " ,"
0610            LD      BC,(5C45H) BC = PPC = current line
0620            LD      (SECTR),BC store in SECTOR
0630            CALL    1A1BH      print the line number
0640            LD      A,":"
0650            RST     10H        print a colon
0660            LD      C,(IY+13)
```

```
0670            LD      B,0
0680            CALL    1A1BH       print the statement no.
0690            LD      HL,5C3BH    HL = FLAGS
0700            RES     5,(HL)
0710            EI                  interrupts on
0720 WAIT       BIT     5,(HL)
0730            JR      Z,WAIT      wait for a key to be pressed
0740            LD      HL,5C42H    HL = NEWPPC
0750            LD      DE,LINE     DE = error line to jump to
0760            LD      (HL),E      store the line
0770            INC     HL
0780            LD      (HL),D
0790            INC     HL
0800            LD      (HL),1      make it statement 1
0810            LD      (IY+0),255  clear the error
0820            LD      (IY+124),0  clear FLAGS3
0830            CALL    0D6EH       clear the lower screen
0840            JP      1B7DH       jump to the 16K ROM
0850            END
```

RS232 TAB routine

This routine alters the channel "P" data so that LPRINT etc will be sent to the RS232 port, similar to the "T" channel, but with the TAB function implemented. It uses three of its own system variables — WIDTH = carriage width of printer, POSN = current position of print head, CONTR = a flag. When TAB is interpreted, firstly a character 23 is sent, then the LSB of the following number, the MSB.

```
0010 ;          "RS232 TAB routine"
0020            ORG     23296       locate in printer buffer
0030 SETUP      LD      HL,(23631)  HL = CHANS
0040            LD      BC,15
0050            ADD     HL,BC       HL = channel "P" area
0060            LD      DE,START    DE = new output routine
0070            LD      (HL),E      store the new routine
0080            INC     HL          location in "P" area
0090            LD      (HL),D
0100            LD      BC,0
0110            LD      HL,POSN
0120            LD      (HL),B      zero POSN
0130            INC     HL
0140            LD      (HL),B      zero CONTR
0150            RET                 return with 0
                                    The O/P routine: A = code
```

```
0160 START    CP     " "
0170          JR     NC,NORM    jump if > = " "
0180          CP     13
0190          JR     NZ,NNL     jump if not newline
0200          LD     HL,CONTR   Newline:
0210          BIT    0,(HL)     test the auto flag
0220          RES    0,(HL)     clear the auto flag
0230          RET    NZ         don't do a N/L if set
0240 NEWLI    LD     HL,CONTR
0250          RES    0,(HL)     clear the auto flag
0260          DEC    HL         HL = POSN
0270          LD     (HL),0     zero POSN
0280          LD     A,13
0290          CALL   OUTCH      send a N/L (13)
0300          LD     A,10
0310          JP     OUTCH      send a Line-feed (10)
0320 NNL      CP     23         is it the TAB character?
0330          CCF
0340          RET    NZ         return if it isn't
0350          LD     DE,TAB2    TAB command:
0360 REDO     LD     HL,(5C51H) HL = CURCHL
0370          LD     (HL),E     store DE in CURCHL ie alter
0380          INC    HL         the O/P location to DE
0390          LD     (HL),D
0400          RET               return
                                 Come here with LSB of TAB
                                 position:
0410 TAB2     LD     (5C0FH),A  store the LSB in TVDATA2
0420          LD     DE,TAB3
0430          JR     REDO       make the O/P routine TAB3
                                 Come here with MSB of TAB
                                 position:
0440 TAB3     LD     DE,START
0450          CALL   REDO       restore O/P to normal
0460          LD     A,(5C0FH)  A = TAB position
0470          LD     D,A        store in D
0480 TAB4     LD     HL,WIDTH
0490          SUB    (HL)       A = TAB-WIDTH
0500          JP     NC,046CH   "Int out of range" if carriage
                                 not wide enough
0510          INC    HL         HL = POSN
0520          LD     A,D        A = TAB
0530          SUB    (HL)       A = TAB-POSN
0540          PUSH   DE         save D
```

```
0550            CALL  C,NEWLI     do a newline if it won't fit
0560            POP   DE          restore D
0570            LD    A,D         A = TAB
0580            SUB   (IY + POSY) A = TAB - POSN
0590            RET   Z           return if in right place
0600            LD    B,A         B = no of spaces needed
0610 TABB       LD    A," "
0620            PUSH  BC          save BC
0630            EXX               swap regs
0640            RST   10H         print a space
0650            EXX               swap back
0660            POP   BC          restore BC
0670            DJNZ  TABB        print the appropriate spaces
0680            RET               return when done
                                  Come here with 'normal'
                                  codes
0690 NORM       CP    0A5H
0700            JR    C,NORM2     jump if not a token
0710            SUB   0A5H
0720            JP    0C10H       detokenise it
0730 NORM2      RES   0,(IY + CONY) clear the auto flag
0740            LD    HL,5C3BH    HL = FLAGS
0750            RES   0,(HL)      clear the 'preceding space'
                                  flag (what the ROM doesn't)
0760            CP    " "
0770            JR    NZ,NSPAC    leave it if not a space
0780            SET   0,(HL)      else set the flag
0790 NSPAC      CP    128
0800            JR    C,OUTC2     print the character if not
                                  graphics
0810            LD    A,"?"       graphics char so print "?"
0820 OUTC2      CALL  OUTCH       send the byte to the printer
0830            LD    HL,POSN
0840            INC   (HL)        increment POSN
0850            LD    A,(HL)      A = new POSN
0860            DEC   HL          HL = WIDTH
0870            CP    (HL)        compare with WIDTH
0880            RET   NZ          return if not the same
0890            CALL  NEWLI       at end of line so do a NL
0900            SET   0,(IY + CONY) and set the auto flag
0910            RET               return
                                  Send a character to RS232:
0920 OUTCH      RST   8           Hook code
0930            DEFB  1EH         2320P byte
```

```
0940                RET                 return
                                        Constants:
0950 WIDTH          EQU     23540       in printer buffer
0960 POSN           EQU     WIDTH+1
0970 CONTR          EQU     WIDTH+2
0980 POSY           EQU     0BBH        IY displacements
0990 CONY           EQU     OBCH
1000                END
```

APPENDIX C
Interface Bugs

With the Interface connected, an additional 8K ROM is used for the new features. This is generally very well written, though there are a few errors (or 'bugs') in it.

1. SYNTAX CHECKING FAILURE

When the original 16K BASIC ROM was written, the syntax of the Interface commands was incorrectly anticipated. Unfortunately this incorrect syntax is still accepted by the checker, but will not run. These incorrect syntax forms are:

ERASE < *string*>	eg ERASE "test"
MOVE < *string*> , < *string*>	eg MOVE "a","b"
FORMAT < *string*>	eg FORMAT "test"
CAT (no number)	

2. COLOUR COMMANDS

Although not strictly an error caused by the ROM in the Interface, the problem explained in Chapter 3 regarding colour commands is caused by a sloppy piece of code in the 16K BASIC. To cure it, precede each permanent colour command with PRINT;.

3. BREAK INCONSISTENCY

There seems to be a general inconsistency about the method of breaking into Interface operations. During Microdrive operations and RS232 output, CAPS SHIFT and SPACE both have to be pressed, but during Networking and RS232 input the SPACE key on its own is sufficient.

4. CASSETTE OPERATIONS

If a filename in a cassette operation is invalid in some way (eg longer than 10 characters) the error that should occur **(Invalid filename)** is overridden with the unhelpful **Nonsense in BASIC**.

5. MACHINE-CODE HOOK CODES

Although the majority of the shadow ROM is well written, the hook codes

seem to have been added as a hurried afterthought. There are two faults in them — hook code READ_N is unusable as the carry flag is corrupted at the end of the routine, and hook code 2B is the same as 22 — a waste of a hook code caused by an incorrect entry in the hook code jump table at location X19C9H.

6. RS232 DOUBLE SPACES

When using a "t" channel to list programs, a double space is printed between keywords eg PRINT PAPER or THEN PRINT. It is caused by the failure to do anything with bit 0 of FLAGS.

7. CLOSE # PROBLEM

If you **Break** into the middle of a CLOSE # statement referring to an Interface channel type, the memory used by the stream is not reclaimed from the memory map. This can use up large amounts of memory, and cannot easily be countered. A CLEAR # will not reclaim it, only a NEW. It could be corrected by the addition of the instruction SET 7,(HL) around location X1741H.

BIBLIOGRAPHY

The Working Spectrum by David Lawrence	Sunshine Books
Programming the Z80 by Rodney Zaks	Sybex Inc.
ZX Spectrum BASIC Programming by Steven Vickers and Robin Bradbeer	Sinclair Research Ltd.
ZX Microdrive and Interface 1 manual	Sinclair Research Ltd.
The Complete Spectrum ROM Disassembly by Dr Ian Logan and Dr Frank O'Hara	Melbourne House Ltd.

Other titles from Sunshine

THE WORKING SPECTRUM

David Lawrence

0 946408 00 9 £5.95

THE WORKING DRAGON 32

David Lawrence

0 946408 01 7 £5.95

THE WORKING COMMODORE 64

David Lawrence

0 946408 02 5 £5.95

DRAGON 32 GAMES MASTER

Keith Brain/Steven Brain

0 946408 03 03 £5.95

FUNCTIONAL FORTH

for the BBC Computer

Boris Allan

0 946408 04 1 £5.95

COMMODORE 64 machine code master

David Lawrence and Mark England

0 946408 05 X £6.95

ADVANCED SOUND AND GRAPHICS for the dragon computer

Keith and Steven Brain

0 946408 06 8 £5.95

SPECTRUM ADVENTURES

Tony Bridge and Roy Carnell

0 946408 07 6 £5.95

THE DRAGON TRAINER

Brian Lloyd

0 946408 09 2 £5.95

COMMODORE 64 ADVENTURES

Mike Grace

0 946408 11 4 £5.95

Sunshine also publishes

POPULAR COMPUTING WEEKLY

The first weekly magazine for home computer users. Each copy contains Top 10 charts of the best-selling software and books and up-to-the-minute details of the latest games. Other features in the magazine include regular hardware and software reviews, programming hints, computer swap, adventure corner and pages of listings for the Spectrum, Dragon, BBC, VIC 20 and 64, ZX 81 and other popular micros. Only 35p a week, a year's subscription costs £19.95 (£9.98 for six months) in the UK and £37.40 (£18.70 for six months) overseas.

DRAGON USER

The monthly magazine for all users of Dragon microcomputers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news related to the Dragon. A year's subscription (12 issues) costs £8.00 in the UK and £14.00 overseas.

For further information contact:
Sunshine
12–13 Little Newport Street
London WC2R 3LD
01-734 3454

Printed in England by Commercial Colour Press, London E7.

Master the ZX Microdrive contains all the information you will ever need to use the ZX Microdrive to the full. Clearly explained, with many examples, it is equally suitable for the relative newcomer to BASIC through to the experienced machine code programmer.

The fundamentals of streams and channels are explained. There is a major section devoted to file handling which explains many features normally only available on disc based machines, including the creation of program files from BASIC.

The book also includes a major database program for use with the microdrive and a chapter explaining how to protect programs. As well as the microdrives two other features of the ZX Interface 1 are explained — the RS232 port and networking.

Extensive details are given about how to use the microdrives, RS232 and network from machine code including explanations of the major ROM routines. A program that adds easy to use commands for the microdrive is included with a full explanation of how to add more of your own. It also details the differences between the three types of Interface 1, and how they affect the user.

Andrew Pennell is a university student, freelance programmer and a regular contributor to Popular Computing Weekly.

GB £ NET +006.95

ISBN 0-946408-19-X

00695

9 780946 408191

SUNSHINE

ISBN 0 946408 19 X

£6.95 net