Pan

PERSONAL COMPUTER NEWS

COMPUTER NEWS LIBRARY

STUART COOKE

# The COMPANION to the SINCLAIR ZX MICRODRIVE and INTERFACES

Pan/Personal Computer News
Computer Library

Stuart Cooke

# The Companion to the Sinclair ZX Microdrive and Interfaces

Pan Books London and Sydney

First published 1984 by Pan Books Ltd,
Cavaye Place, London SW10 9PG
in association with Personal Computer News
9 8 7 6 5 4 3 2 1

ISBN 0 330 28662 5
Photoset by Parker Typesetting Service, Leicester
Printed and bound in Great Britain by
Richard Clay (The Chaucer Press) Ltd, Bungay, Suffolk

## Dedication

Thanks are due to Jane and Dave at Pan/PCN Books without whom this book would never have seen the light of day, to Chris Youngs for his help in getting this book out, on time and to my dear wife, Sue, who is currently nursing me back to health.

# Contents


| | | |
|---|---|---|
| **1** | Introduction | 7 |
| **2** | The ZX Interface 1 | 10 |
| **3** | The Microdrive | 19 |
| **4** | File Handling and Storage | 28 |
| **5** | Building a Microdrive Database | 39 |
| **6** | Interfacing | 55 |
| **7** | Using the Spectrum RS-232 Interface | 61 |
| **8** | Networking | 73 |
| **9** | Machine Code | 91 |
| **10** | Hook Codes | 106 |
| **11** | Extending Spectrum BASIC | 123 |
| **12** | Jolly Joysticks | 145 |
| | Appendices A to I | 154 |

# 1 Introduction

With the introduction of the Interfaces 1 and 2 together with the revolutionary Microdrive storage system, the Sinclair Spectrum can truly be said to have come of age. When you connect these add-ons to your home computer you can finally unleash its full power and potential, giving it high capacity, fast access storage for programs and data, links to the outside world through the RS-232 or NET ports and much, much more. With this book you will be able to understand and use to the full all of these added facilities, bringing a new level of professionalism to your programs and a previously undreamt-of degree of sophistication to every aspect of this fascinating hobby.

The ZX Spectrum, always the subject of heated debate since it was little more than a gleam in Uncle Clive's eye, has made both friends and enemies in the short time that it has been readily available to the public. It can never be denied, however, that the Spectrum was and still is an enormously successful computer. The overwhelming reason for its commanding market position must surely be that the Spectrum is such very good value for money; so much high technology for such a low price! From the start, however, the lack of fast storage and any connections for quality printers and other peripherals, such as joysticks and modems, has been recognised as a serious shortcoming of the Spectrum when compared with other popular computers, the BBC Micro, the VIC-20 and so on. Until now, that is!

The introduction of the Interface 1 and the Microdrive system allows the Spectrum owner to upgrade his computer from a games machine to a full system. The Microdrive gives the user the opportunity to use a fast-access, high volume, data storage system whilst the additional interfacing connections available on the Interface 1 allow communications between several computers via the unique Network port, and with a whole range of additional equipment through the RS-232 port. The Interface 2 allows the use of instantly loading ROM cartridges, whether for games or for sophisticated business applications. For avid games players, the addition of two standard joystick ports to this interface will prove to be both a boon and a relief to fingertips and keyboards.

In the pages that follow we shall be looking in depth at all this additional

hardware, taking it apart for you and, hopefully, leaving you with a thorough and in-depth knowledge of these extra facilities and how to use them to the full. Chapter 2 is devoted to an overview of Interface 1 and the novel but important concepts of streams and channels. Although possibly unfamiliar to you at present, by the end of this section you should be treating the whole subject of data flow with studied ease.

Chapters 3 and 4 bring us on to the Microdrive, Sinclair's unique solution to the problem of supplying cheap, fast-access mass storage of programs and data on a simple home microcomputer. The use and misuse of sequential data files, another enhancement available when using the Microdrive, will be explained in detail, while in the following Chapter 5 a database program is described that has been especially designed to be as useful and as practical as it is demonstrative.

Interfacing is the meat and bone of the next section of this treatise. Connecting your Spectrum system to the outside world is to many people the most fascinating and rewarding aspect of microcomputing today. We won't be explaining how to break into your bank's computer system, of course, but we will happily show you how to use the tools provided by the Interface 1 to communicate with other computers and quality peripheral equipment such as printers and modems. A detailed look at the extra commands provided with the built-in ROM of the Interface 1 in Chapters 9 to 11 is followed, finally, by subjecting the Interface 2 to a close scrutiny, forcing it to reveal its innermost secrets.

In addition to full descriptions of the new hardware and commands, with example listings to help you get the best out of them, this book contains a number of full demonstration programs to allow, say, club secretaries to maintain up-to-date membership lists, or simply computerised address books or card-index files. Chapter 8 contains a 'Battleships' – style number guessing game for use between two Spectrums over a network. There is also a graphics drawing program in Chapter 12 for use with joysticks.

Whilst several chapters dwell heavily upon machine code usage upon the Spectrum, especially when using the hook codes that are provided by the additional ROM inside the Interface 1, this book does not expect that the reader should have a thorough working knowledge of this subject. Indeed, in Chapter 9 there is a brief dissertation, introducing you to the joys and pleasures of machine code programming so as to equip you with all of the necessary accoutrements to allow you freely and easily to use the routines built into the Interface 1 ROM. It is, of course, assumed that the reader has a thorough working knowledge of the BASIC language that is built into the Spectrum but the new, extra commands are explained in detail as they are encountered.

A final note on the various program listings contained in the book. To

make them easier for you to type in accurately and quickly we have listed them all at a line length of 32 characters, so that they appear in the book just as they should on the screen if correctly entered. Where Spectrum graphics characters have been used, they appear in the listings as italic characters.

# 2 The ZX Interface 1

'My computer's got better graphics than yours', 'I've got an RS-232 port', 'My computer's got two disk drives.' Comments such as these abound in computer circles, and for some time Spectrum owners have had their 'un' fair share. Now they need be endured no longer for Sinclair have taken a lead from the makers of Star Wars and have launched a new epic: "Sinclair Strikes Back – with the Microdrive".

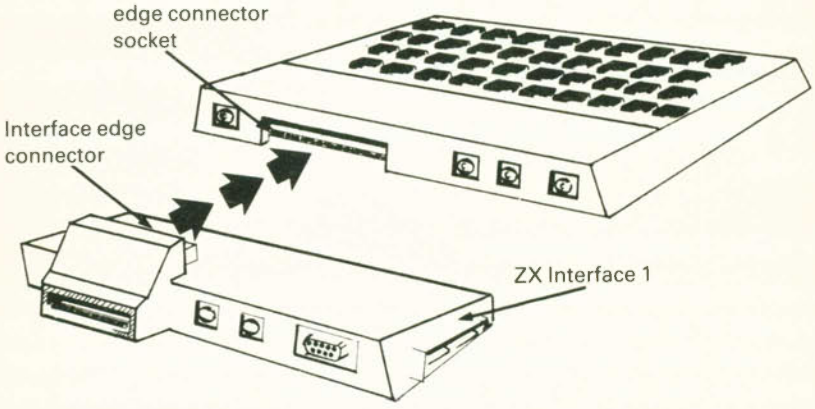
When Sinclair Research first launched the Spectrum, they promised the user that, ultimately, the following features would be available for it: an RS-232 port, networking facilities, and a mass data storage system. They kept their word and these features are now all provided in one fairly small, black box that fits snugly onto the back of the computer. Anyone owning a Spectrum can now have access to these facilities that up to now have been available only on machines costing many times more than their own humble computer.

In essence, Sinclair have introduced three separate add-ons for the Spectrum. Interface 1 can be used independently and allows the user to communicate either through the network port (to other Spectrums) or through the RS-232 port (to a wide range of non-Sinclair equipment). As well as this there is an additional edge connector on the Interface 1 that allows the connection of the Sinclair Microdrive storage system. By 'daisy-chaining' each one onto the side of the previous one, up to eight Microdrives can be connected to the Spectrum, totalling over half a megabyte of on-line storage! Interface 2 is the third additional piece of hardware, and this will be dealt with in detail in Chapter 12. The Sinclair Microdrive is, in itself, a great boon to the Spectrum user. Each Microdrive cartridge can store up to approximately 90k (although this varies slightly) and data or programs can be accessed rapidly and reliably. Perhaps the greatest, and certainly the most convenient aspect of the Microdrive is that it is automatic in its operation. No more pressing of cassette recorder buttons, searching for the correct tape to put into the recorder or laborious labelling of each tape; the Microdrive takes care of all of these operations for you. With the help of this book you will find out how to use these new pieces of equipment, so read on!

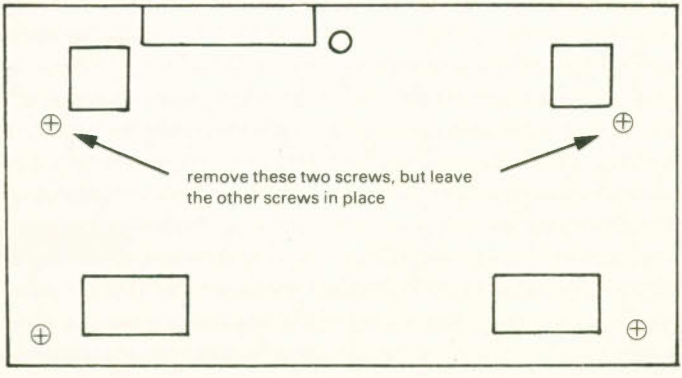
## Setting up the Interface 1

Whilst the Interface 1 can be used by itself without the Microdrive connected, the reverse is not the case. So, if your main interests lie in the realm of communications you need only purchase the Interface 1. However, if you wish to streamline your data storage facilities, you must have both the Interface 1 and at least one Microdrive unit.

Connecting the Interface 1 to your Spectrum could not be simpler. First, make sure that your computer is disconnected from the power supply. Then find the edge connector on the Interface and gently but firmly push it into place in the socket at the rear of the Spectrum where the ZX Printer normally fits.




You will notice that there are two screws in the Interface 1, these provide the option of connecting it up to the Spectrum permanently. If this is what you want to do then you simply remove the two corresponding screws at the rear of the Spectrum base and screw the interface firmly into position.



Even if you want to keep your Spectrum and interface permanently hooked together, you are likely to find that there will probably be odd times when you want to separate them, for example when using software that will not work with the Interface 1. If you have screwed the computer and interface together you will then have to grab a screwdriver to take them apart and you might find that what was initially a convenience has become a major inconvenience. Until you are sure how you are going to be using your new Spectrum add-ons, therefore, it is probably advisable that you use the temporary rather than the permanent method of connection initially.

Once the Interface 1 and the Microdrive have been connected you will be able to:

1. Send and receive data to and from any other compatible RS-232 device, (e.g. printer, terminal, another computer).
2. Talk with up to 63 other Spectrums (or QL's, in theory at least) *via* the network.
3. Store and retrieve data from a single Microdrive or multiple Microdrives.

**Interface 1 and its connections**

Once the interface is connected up you will find that the computer is tilted forward at a more convenient angle to facilitate rapid work on the keyboard. This angle also helps you to see the keyboard symbols more clearly as they have a shiny surface which may otherwise reflect your working light.

The connections for power, TV, Mic and Ear are accessible as before. The edge connector that was provided for other accessories is now in use with the Interface 1, but there is a duplicate connector on the rear of the interface so you are not denied this facility just because your Interface 1 is in use. All the accessories that could be used with the basic Spectrum will also work on the new edge connector, for example the ZX Printer.

On the rear edge of the Interface 1 you will find two unmarked 3 millimetre jack sockets and a 9 way 'D' line connector. The jack sockets are for use when networking; either one or both can be in use at the same time depending on whether you are on the end or in the middle of a network line. It will not do any damage to your computer if you should erroneously connect, say, a tape recorder outlet to these sockets, but of course you won't be loading the program as you thought.

The 9 way 'D' line socket is the RS-232 interface in/outlet, and it is quite likely that damage would occur if you could accidentally connect a joystick to this socket. Fortunately, Sinclair are one step ahead of you and have made the RS-232 port Connector the same 'sex' as the standard

joystick, thereby successfully deterring any casual interfacing between the two.

On the lefthand edge (looking from the front of the computer) there is a unique 14 way edge connector. Not all of these are used as you can see below. This is solely for connection of the short Microdrive cable and has no other function. If you want to attach extra Microdrives they can be 'piggy-backed' onto each other. The Microdrive closest to the computer is always drive number 1, the next is drive number 2 and so on, up to a maximum of eight, each one connecting to the next drive on the right.

*The Microdrive edge connector*

This consists of a double row of seven edge-type connections, of which five are unused and grounded and the rest are used as follows:

0 volts
+9 volts
read/write line
erase line
two bidirectional data lines
two control lines
write/protect line

*Inside the Interface 1*

If you were to venture inside the Interface 1 you would find a system of electronic logic very similar to that of the Spectrum itself, in that it has its own ULA (Uncommitted Logic Array), a ROM and various control and switching transistors. All this adds up to the heart of the extra facilities now at your command. Possibly the most interesting component is the ROM (Read Only Memory). From its name you can see that extra memory is provided when you use the interface 1, but it is only available to the Central Processor Unit. The ROM contains all the extra machine code needed to support the new instructions (for example CAT 1) that gives an error message if they were to be used without the interface connected.

Of course you don't have to know how the computer works in order to be able to use it to the full.

*To check that the interface is working*

The easiest way to check whether your Interface 1 is functioning correctly is to try out the new command CLS #. Perhaps one of the most useful features of the Spectrum is being able to change the INK, PAPER and BORDER colours very easily. However, when you want to LIST a program in which the INK and PAPER colours are very close to each other, for example, dark blue on black, or magenta on cyan, it is a lot of trouble having to change them separately to something more legible such as black on white. In fact it is not unusual for programmers to add a line such as:

```
1000 PAPER 7:INK 0:BORDER 7
```

to their programs so that the colours are reset to their default values on exit from the program.

Sinclair have made this procedure easier on the Interface 1 by introducing a new command: CLS#. This single command sets all the colours to their default values, thus speeding things up considerably. It is also a useful way of testing whether the Interface is connected up properly because it will generate an error message unless it is so.

### Streams and Channels

Before using the Interface 1 or any device associated with it, it is necessary to understand fully the use of streams and channels which is the method by which the Spectrum communicates with printers and any other peripherals. The data transmission travels along routes known as *streams*, and the peripherals or devices with which communication takes place are known as *channels*.

You probably already know that to output to the television screen you type:

PRINT

followed by either a string or a number. For example, if you enter:

PRINT 100;" Hello"

the message '100 Hello' will then appear on the screen. This is an example of using a channel, in this case the upper section of the TV screen. Other channels that you can send data to are:

a ZX Printer
a Microdrive file
another Spectrum on the network
a printer or other peripheral *via* the RS-232 interface

As well as using channels for the output of data you can also use them for inputting, for example receiving information from the keyboard.

In order to be able to organise and control the flow of all this data the computer is configured into 'streams'. The Spectrum can handle up to 16 streams at a time, numbered 0 to 15. Four of these are automatically allocated by the Spectrum when the power is switched on, and the user has the choice of any of the other 12. The table below shows the allocation of the reserved streams and associated channels at power on:

| STREAM NUMBER | CHANNEL | DEVICE |
|---|---|---|
| #0 | K | Input/output to lower screen/ keyboard |
| #1 | K | Input/output to lower screen/ keyboard |
| #2 | S | Output to upper screen only |
| #3 | P | Output to Sinclair printer only |

As you can see, the first two streams, 0 and 1, are assigned to the same channel. With those channels that permit both outputting and inputting of data, the commands to use are PRINT#, INKEY$# and INPUT#. Obviously an input is not possible from some of the channels, for example you cannot input from a printer, (P channel) and trying to do so will only generate an invalid I/O device message.

All the above channels are available on the standard Spectrum. In addition, when the Interface 1 is connected a number of other channels become available. These channels will be mentioned here so that you will have a complete overall picture, but the details of using them will be left to the relevant chapters.

| CHANNEL | DEVICE |
|---|---|
| M | Microdrive |
| N | network |
| T (text) | RS-232 |
| B (binary) | RS-232 (Useful in sending special control characters to some printers) |

Let us now continue with the PRINT example that we started at the beginning of this section. A PRINT statement may be followed by '#' and a number. This statement now means PRINT out to the stream of the specified number. For example, try typing:

```
PRINT #2; "Hello"
```

When you press ENTER the word 'Hello' will appear on the screen. This is no different from the normal PRINT statement so what is the use of the PRINT # statement? Let us modify the statement to:

```
PRINT #1; "Hello"
```

When you press ENTER you should see that the word 'Hello' will appear on the bottom line of the screen. This will then be very quickly replaced by OK. To prevent the OK message from erasing your message modify the line to read:

```
PRINT #1;"Hello":PAUSE 0
```

The word 'Hello' will now be visible on the bottom line of the screen and will remain there until you press a key.

As you can see, the command above PRINTS out to the area of the screen which you can normally only use for entering your program statements. Now that you know how to use PRINT #1; you will be able to output messages on this normally unused line, often referred to as the lower screen, within your programs. For example, the following subroutine could be used within a program to generate a pause:

```
1000 PRINT #1; "PRESS ANY KEY TO CONTINUE"
1010 IF INKEY $#1 = " " THEN GOTO 1010
1020 RETURN
```

If you have a ZX Printer attached to your Spectrum then try entering the following program:

```
10 PRINT #3;"HELLO"
20 FOR x=1 TO 50
30 PRINT #3;x;" ";
40 NEXT x
50 PRINT #3;"BYE-BYE"
```

When you RUN it, all the printing will be directed to the printer rather than to the screen. PRINT #3 has the same function as LPRINT.

*OPEN #*

No stream is permanently connected to a specific channel, not even those set up by the Spectrum at power on. To set up a stream and channel you need to use the statement:

OPEN #stream,"channel"

We know that the PRINT statement always outputs to stream number 2, which by default is the screen. However if we type:

OPEN #2,"P"

all PRINT statements will now be outputted to the ZX printer. This is very useful if you want to keep a record of any output. Likewise LPRINT and LLIST always output through stream number 3, so the statement:

OPEN #3,"K"

will cause all printer output to be re-directed to the lower screen. If you do not own a printer or simply do not want to use it in this case, OPENing stream 3 to either the lower or the upper screen will enable you to perform a check on what would have been sent to the printer had it been connected up. This course of action can save you a lot of wasted paper!

Of course you are not restricted only to streams 0 to 3. As already mentioned there are 16 streams available on the Spectrum. Therefore, there is nothing to stop you using the statement:

OPEN #5,"S"

and using PRINT #5 in any program rather than the normal PRINT.

One very handy feature offered by the use of streams and channels is that it allows you to choose where output should go within a program. As an example, the following few lines allow you to choose whether output should be directed to the screen or to the ZX printer.

```
  10 LET dev=2: REM 2 is the str
eam associated with the screen.
  20 INPUT " Screen or Printer (
S/P) ?";d$
  30 IF d$="P" OR d$="p" THEN LE
T dev=3: REM 3 is the stream ass
ociated with the printer.
  40 PRINT #dev;"Variable or str
ing"
```

All PRINT statements in the rest of the program should be of the PRINT #dev format so that output can go out to either the screen or the printer, depending on your choice.

Although the Spectrum offers a total of 16 different streams there are times when you will want to use a particular stream and channel. If you try to OPEN a stream that is already in use an error message *stream already open* will be generated. Streams 0,2 & 3 will be OPENed automatically on power up so be careful not to reOPEN them, as results are unpredictable to say the least!

*CLOSE#*

When you have finished, and want to end communications with the stream currently in use, you must use the CLOSE# command. The format is:

CLOSE #stream

When you CLOSE streams 0 to 3 they will return to their default channels whilst other streams cannot be used again until they are reOPENed for communication. If you try to use a stream that is not OPEN you will get the error message O *invalid stream*.

*CLEAR #*

If you have a large number of streams OPEN and you want to CLOSE all of them it would be a very long-winded process to CLOSE # each of them individually. Thankfully, the command CLEAR # has been added with the

Interface 1. In one operation this will CLOSE all streams that are currently OPEN, streams 0 to 3 will be returned to their original values and streams 4 to 15 will be forgotten. There is only one problem with using CLEAR # and this concerns any data that is left in the stream when it is CLEARed. When you use CLOSE # any data currently in the stream will be sent to the relevant channel *before* the stream is CLOSEd. However when you use CLEAR # any data still in the stream is simply destroyed, so you need to be very careful when using this command to ensure that you do not suffer the loss of some of your important data.

**MOVE**

The MOVE statement allows the user to transfer data from one source to another. The source can be specified either as a stream or a particular device ie. channel. The format is:

MOVE #source stream TO #destination stream

Either of the streams can be replaced by a channel. In this case the format for a channel rather than a stream would be:

MOVE #device,number,name

Depending on the device being used the number and name may be optional.

Let us have a look at some examples of MOVE # in use.

MOVE #10 TO #2

This will send any data received by stream number 10 to stream number 2. If the channel associated with stream number 2 hasn't been changed, all data received by stream 10 will be printed on the screen. The statement:

MOVE "t" TO "m";1;"RS-232"

will send all the data received from the RS-232 port to Microdrive number 1 and save it with the filename 'RS-232'.

Another useful application for MOVE is to copy files:

MOVE "m";1;"FILE1" TO "m";1;"FILE2"

This will copy FILE1 as FILE2 on the same Microdrive cartridge, for example as a backup copy. Copying can be carried out between multiple Microdrives simply by changing the drive numbers ";1;" in the above statement.

MOVE "m";1;"FILE1" TO "m";2;"FILE2"

Similarly the contents of a Microdrive cartridge can be displayed directly:

MOVE "m";1;"FILE1" TO #2

this displays FILE1 directly on the upper screen.

# 3 The Microdrive

The main storage device associated with the Spectrum is still the household cassette recorder. Though the cassette interface of the Spectrum is reliable and fast compared to that of other micros, many owners want an even faster way of storine their programs. Most machines have disk drive options and in fact many machines need disk drives to operate at all. However, although disk drives offer a very fast way of LOADing and SAVEing programs the cost is very high, in some cases more than the cost of the computer.

Within Interface 1 Sinclair have included an interface for what Sinclair call 'Microdrives'. Not only do Microdrives offer you facilities similar to those of a disk drive but they are also comparatively cheap. As well as moving away from the traditional disk drive Sinclair have also designed their own medium for storage rather than the usual floppy disk. The new medium is called a 'Microdrive cartridge'.

## The Microdrive cartridge

So what is the Microdrive cartridge? Basically the cartridges are very similar to the cassettes already used by Spectrum owners. The casing of the cartridge is much smaller than a normal cassette and the tape is much thinner and narrower. Probably the main disadvantage with a cassette is that you have to rewind the tape every time you wish to read a file from the beginning. To solve this problem the Microdrive tape is a continuous loop. This means that the tape only has to move in one direction to find a file. However the continuous loop doesn't remove all of the problems associated with cassette tape, like what to do if the section of tape containing your program has just gone past the read/write head of the Microdrive. You cannot rewind the tape and it must all be moved past the head before it comes back to your program. However when compared with the normal cassette the Microdrive and cartridge are much faster. Typically any program can be found and loaded within 3–4 seconds.

*Care of the Microdrive cartridge*

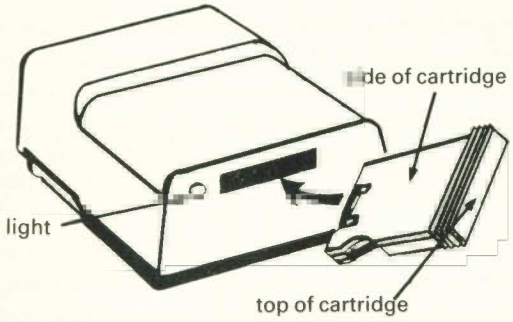
You probably know that you have to be careful how you handle cassette tapes:

- don't touch the tape with your fingers.
- never leave them out of their cassette boxes.
- never put them near to a magnet – don't forget TV's and radios contain magnets.

Microdrive cartridges are more fussy than the humble cassette. Not only must the above guidelines still be followed but you must also:

- never remove a cartridge from the Microdrive while reading or writing (the red light on the front of the Microdrive will indicate when the drive is active)
- don't apply power to, or remove power from the Spectrum while a cartridge is in the Microdrive.

Microdrive cartridges, unlike normal cassettes, can only be used one way round. Both the Microdrive and cartridges are manufactured in such a way that the cartridge can only be inserted into the drive the correct way round. So if you are having problems inserting a cartridge don't try to force it into place, make sure that you are trying to insert it correctly. The figure below shows how the cartridge should be inserted.




**The Microdrive in use**

When you purchase any commercial software on a Microdrive cartridge, accessing the programs contained on it is usually a simple process of LOADing them into the Spectrum and away you go. However, when you first buy a blank cartridge for using with your own programs this is not the case. Before you can actually use a new cartridge to SAVE any programs or data, you must first prepare the cartridge by *formatting* it. It may appear strange at first that a blank cartridge cannot be used immediately, especially if you are used to using cassettes, but if you think about the differences between the Microdrive and a standard cassette recorder, the necessity of the formatting process will become obvious. When you SAVE a program upon a cassette the computer starts SAVEing it as soon as the command is entered and you press the RECORD/PLAY keys on the

recorder. If there is already a program or other data on that section of tape then it will be irretrievably lost and the new program will be recorded over the top of it. When a cassette is being used the computer has no way of knowing what is on a particular tape or whether there is any free space available for more data. This makes the use of cassettes a very slow and tediously manual process with a great deal of operator intervention.

With the introduction of the Microdrive, Sinclair have taken the opportunity to automate operations such as LOADing and SAVEing, making the system more like that of a disk drive. However, if the operating system is to perform its function properly it must be able to know at all times the allocation of free space on the cartridge. To do this the loop of tape must first be formatted, that is, divided up magnetically into a series of *sectors*, each with its own set of identification marks. Once this has been carried out the operating system can easily detect the location of any particular sector. By keeping track of the sectors that contain data, the Microdrive can, quite automatically, SAVE more data onto the cartridge onto free areas of tape without corrupting what is already there.

The process of preparing a cartridge is started by using the command FORMAT as follows:

```
FORMAT "m";DN;"name"
```

where DN is the number of the drive that the cartridge is in and *name* is the identification name that you choose for the cartridge. The drive number must be in the range 1 to 8 and the cartridge name can be no longer than 10 characters. For example:

```
FORMAT"m";1;"GAMES"
```

will prepare the cartridge in drive 1 for use, assigning it the name 'GAMES'. As we shall see shortly, the cartridge name is therafter used and displayed whenever the catalogue is requested.

During the FORMATting process, every section of the tape is checked to make sure that it can be read from and written to. If any sections are no good then the computer will mark them as unusable. Then the cartridge name and the amount of storage that is available is written to the tape.

The actual amount of storage capacity on a Microdrive cartridge is usually between 80K and 90K. The 'K' stands for kilobytes and is basically a measure of the number of characters that can be stored. 1K is 1024 characters. So a cartridge can store around 90,000 characters.

Once a cartridge has been FORMATted you can use it to store your own files. Every file that you save onto a cartridge must have a completely different name of between 1 and 10 characters. Any type of file may be saved onto cartridge: programs and data can be mixed on the same tape

without any problems so long as file names are kept unique.

Lets try FORMATting a cartridge. Insert a BLANK microdrive cartridge into drive number 1 as shown in the earlier drawing. Then key into your Spectrum:

```
FORMAT "m";1;"GAMES"
```

When you press ENTER, the red light on the front of the microdrive will come on to show that it is in use and you will hear the tape moving. FORMATting a cartridge will take about half a minute.

Before you FORMAT a cartridge make sure that it is the one you want to use as all information on the tape will be lost and cannot be retrieved after. This can be used to clean a cartridge completely for re-use without having to delete files individually.

### Saving your information

*Saving*

The command used to SAVE a program onto Microdrive takes a similar form to that of the cassette SAVE command and is used in the following syntax:

```
SAVE *"m";drive number;"filename"
```

Of course you can still use all the optional parameters that are available when using a cassette tape, like:

```
SAVE *"m";2;"maccode" CODE 30000,35000
```

which will SAVE the contents of memory locations 30000 to 35000 onto a cartridge under the name "maccode" on Microdrive number 2. Furthermore,

```
SAVE *"m";1;"array" DATA C()
```

will save the array 'C' on Microdrive number 1 under the name "array".

```
SAVE *"M";1;"array" DATA a$()
```

Will SAVE the character array a$ under the name "array"; and

```
SAVE *"m";1;"picture" SCREEN$
```

will SAVE the screen memory with the file name "picture".

*Autorun*

When you SAVE onto tape you can SAVE a program so that it will start to RUN from a specified line number, the same is also possible with the Microdrive. However this autorun feature has been expanded.

If you turn on your Spectrum and type RUN and press ENTER, the

Spectrum will try to access a file on Microdrive number 1. The file that is being searched for is "run". Therefore if you wish to LOAD and RUN the same file every time you use a specific cartridge then SAVE the file under this name. Don't forget to SAVE the program with a line number if you wish the program to autorun once loaded:

SAVE *"m";1;"run" LINE 10

Lets try SAVEing a short program onto the cartridge which we have just FORMATted. Enter into your Spectrum the following:

```
10 FOR x=1 TO 100
20 PRINT "HELLO OUT THERE"
30 NEXT x
PRINT "THE END"
```

Now let's SAVE it onto a cartridge so that it will autorun. Make sure that your cartridge is in drive 1 and type:

SAVE *"m";1;"run" LINE 10

The red light on the front of Microdrive 1 should come on and you should hear the tape moving around. The program will now be SAVEd onto the Microdrive cartridge and the light will go out.

*VERIFY*

When a program has been SAVEd onto cassette tape, it is possible to check that it has been SAVEd correctly by using the command:

VERIFY "filename"

this will check the SAVEd file against the one in memory making sure that they are exactly the same. However, because the cassette interface on the Spectrum is so reliable the VERIFY command is usually forgotten.

It is also possible to VERIFY a program that has been SAVEd onto a Microdrive tape by using the command:

VERIFY *"m";drive number;"filename"

Lets check that the program which we have just SAVEd has been stored correctly. Enter on your Spectrum:

VERIFY *"m";1;"run"

The Spectrum will then check the file that is held in memory against the one that you have just stored onto cartridge. You will see the OK message if all is well.

Because Microdrive cartridges are not as reliable as cassettes it is a good idea to VERIFY EVERY program that you SAVE. It is also good practice to

SAVE the same program onto Microdrive more than once. This can not be done through BASIC: if you SAVE the file "test" onto cartridge and then try to SAVE "test" again you will get an error. However if you type:

```
POKE 23791,number of copies
```

before you SAVE the program, the specified number of copies will be SAVEd onto the cartridge under the same name. Saving multiple copies of programs has a number of advantages. First, if you accidentally ERASE one of the files then you will still have the other copies, and secondly, when LOADing the file the Microdrive tape will not have to travel as far before the named file is found.

*CAT*

We have already discussed how the cartridge tape is magnetically divided into sectors so that the operating system is able automatically to keep track of the contents of the cartridge and the whereabouts of data upon it. Since the operating system is keeping track of all the files on the cartridge, it also keeps a list of the filenames on record and you are able to call up this *catalogue* onto the screen using the command CAT. When you use this command the names of the cartridge, all the files upon it and the amount of free space still remaining will be PRINTed to the screen. This is achieved by keying in the following:

```
CAT DN
```

where DN is the drive number. There is an extension to this command which allows you to direct the output to a channel other than the screen, for example:

```
CAT #ST,DN
```

where ST is the stream along which output is to be sent and DN is still the drive number. When using this form of the command you must previously have OPENed the stream to the device to which you wish to direct the CATalogue data.

If you read the CATalogue for the cartridge that we have just FORMATted you will see from the screen that not only does it contain the cartridge name and a list of the files upon it (which will probably be only one at this stage, 'run') but also at the bottom of the list a number which is the amount of free space on the cartridge in kilobytes (1024 bytes). The initial value for this figure, when the cartridge is freshly FORMATted and as yet holds no files, will vary from cartridge to cartridge. This is because, as part of the FORMATting process, the operating system checks the quality of the

tape coating of each sector and thus its ability to hold data. Any sector that fails this test will be electronically marked as unavailable to the system. Whilst this process is worthwhile since it reduces the likelihood of data becoming corrupted, it does reduce the amount of storage space on a cartridge, sometimes quite drastically. Usually, the amount of free space on a newly FORMATted cartridge is between 80 and 90 Kbytes but if you are lucky it may be more than 90.

*LOAD*

The command for LOADing a program from Microdrive uses the same format as SAVE:

```
LOAD*"m";DN;"filename"
```

and, as with the SAVE command, all the usual extensions such as CODE or DATA can be used.

Try LOADing in the example program that you SAVEd earlier. Because this program has been SAVEd using the optional load and run variant, there are two ways in which you can LOAD it. You can either use the standard form of the command:

```
LOAD*"m";1;"run"
```

or else if you have just plugged your Spectrum in you can just type RUN (and ENTER), as long as the cartridge is in place in the Microdrive.

*MERGE*

As with a cassette–based program, it is possible to MERGE a program stored on a Microdrive with the one currently in memory. Let's write a simple program, SAVE it and then MERGE it with a new program. Enter the following on your Spectrum:

```
10 REM this is program one
20 FOR x=1 TO 100
30 PRINT x;" ";
40 NEXT x
```

Now save the program to Microdrive 1 by typing the following:

```
SAVE *"m";1;"PROGONE"
```

This will SAVE the first of our programs onto the cartridge. Now enter NEW to clear the first program from memory and then type in the second:

```
 100 REM this is program two
 110 PRINT "HELLO I'M PROGRAM TW
O"
 120 GO TO 10
```

Now if you type:

MERGE *"m";1;"PROGONE"

the program that you have just SAVEd will be added on to the one in memory. LIST the program and you should see the following:

```
  10 REM this is program one
  20 FOR x=1 TO 100
  30 PRINT x;" ";
  40 NEXT x
 100 REM this is program two
 110 PRINT "HELLO I'M PROGRAM TW
O"
 120 GO TO 10
```

*ERASE*

When you want to ERASE a program that is SAVEd on a cassette tape all you have to do is record a new program over it. However with the Microdrive there would be now way to tell which files could be deleted. The ERASE command allows us to get around this problem. ERASE tells the Interface 1 to delete the named file from the Microdrive. To use this command you must type:

ERASE"m";drive number;"filename"

Once you press ENTER the border will flash and the named file will be deleted.

However, before you press ENTER you should always make sure that the file that you are about to wipe out is the correct one. If you have ERASEd the wrong program then there is no way to get it back. For example, you can delete the first program that we saved from the cartridge by typing the following:

ERASE "m";1;"run"

If you now call for a CATlogue you will see that the program is no longer on the cartridge.

The Microdrive operates much faster than a normal cassette recorder. Arguably the best type of program to demonstrate the increase in speed would be one which LOADs a screen picture. Below is a program that will

produce a multicoloured display on the screen and then SAVE it as a SCREEN$:

```
10 REM multicolour screen
20 FOR c=1 TO 22
30 LET p=INT (RND*7)
40 PRINT PAPER p;"
                  "
50 NEXT c
60 REM now save the screen
70 SAVE *"m";1;"picture"SCREEN
$
```

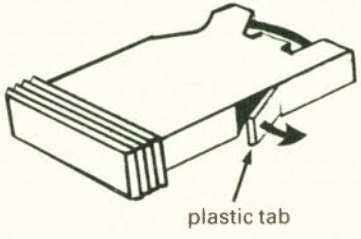
Now delete the program by typing NEW, and LOAD the program back into memory by typing:

LOAD *"m";1;"picture" CODE

It's a lot faster than loading a screen from cassette, I'm sure you'll agree.

*Program protection*

If you have stored a program which you particularly wish to keep it is a good idea to protect it against accidental erasure. With a normal cassette you break off the small plastic tabs on the rear of the casing. Microdrive cartridges, just like cassettes, have a write-protect tab. Once this tab is removed the Spectrum will no longer allow you to write to that cartridge. Of course if you wish to write to the cartridge at a later date you will have to cover the position where the tab should be with sticky tape. The figure below shows where the tab is positioned.




*Backing up your cartridges*

Unfortunately, Microdrive cartridges are not always as reliable as normal cassettes. By following the guidelines above it should be possible to use your cartridges for quite a long time. Even if you treat your cartridges with extreme care, however, it is still advisable to keep a backup of *all your programs* on a normal cassette. This is good computing practice in any case, and is the best way to guard against the accidental loss of valuable data.

# 4 File handling and storage

When you use a cassette with the Spectrum you can only SAVE programs which you have written in BASIC, programs which are written in machine code, and arrays and variables. With the Interface 1 connected to the Spectrum the programmer can use a new type of file on the microdrive: the data file.

The type of data file that Microdrives use is called a sequential file. All this means is that data is stored one item after another, for example:

```
1 3 4 33 567 22 44 55 66 88 234 456
```

If you wish to read the last item of data in the file, for example the number 456, then you must also read in the whole of the file up to that number. In the case above this would mean reading all of the numbers up to the end.

In an earlier chapter we discussed streams and channels. All the channels mentioned use sequential data since information is sent out to them one item at a time. However Microdrives are a little more clever than these, as not only can we send data to the drives but we can also read it back whenever we like.

To set up a data file on the Microdrive we use the OPEN # command and set up a stream to send data down to the Microdrive, just as we did when we set up streams to print out to the screen or printer. Because we need to tell the Microdrive which file to print to or read from extra information is needed in the OPEN # command, and the new format is:

```
OPEN # stream number;"m";drive number;"filename"
```

The stream number is one of your own choice; the 'm' indicates that we are using a Microdrive channel; drive number is the number of the Microdrive to which we wish to write and the filename is the name of the file we wish to use on the Microdrive tape, whether we are creating it for the first time or re-opening it for re-use.

### Writing a sequential file

Let's try and write something to a file. Type in the following program:

```
10 OPEN #5;"m";1;"FILEONE"
20 FOR a=1 TO 100
30 PRINT #5;a
40 NEXT a
```

and then RUN it, first making sure that you have a cartridge in drive 1. You should hear the Microdrive operate as it searches for the specified file. A file called "FILEONE" should not already be on the cartridge if you want "FILEONE" to be a write file. If the file is not found then the file will be created. If the file does exist then it will be OPENed for reading, however as soon as you try to write data to it you will get a *writing to a read file* error. This way you are prevented from inadvertently writing to a non-existent file.

You may be surprised to know that in the latter case none of the data which should have been sent to the Microdrive ever got there. This is because when we inform the Spectrum that we are using the Microdrive an area is set up in memory which holds all the information that is required to access the drive filename and so on. This is the actual Microdrive channel. Within this channel a buffer is established which has 512 different locations. Whenever the program performs a PRINT # operation this is where all the data is stored. No data will be sent to the Microdrive until either:

(i) the buffer gets full, in which case it empties itself onto the cartridge, or

(ii) you tell the Spectrum that you have finished with that channel by CLOSEing the stream.

So to send the above data to the Microdrive we must enter:

```
CLOSE #5
```

You should now hear the drive start to move and the data should be written onto the cartridge. Not only is the data sent out to the Microdrive but the area of memory used by the channel is reclaimed when it is no longer needed.

So that we can see the buffer in operation let's modify the above program so that we send out much more data.

```
10 OPEN #5;"m";1;"FILETWO"
20 FOR x=1 TO 1000
30 PRINT #5;x
40 NEXT x
50 CLOSE #5
```

When you RUN this program you should hear the drive stopping and starting as data is sent to it. Each time the drive operates it means that the

512 character buffer is full and it is emptying itself. While the Spectrum is writing to the drive the program will stop running. This is very easy to show. First ERASE the file so that it will be opened as a write file; then enter

```
ERASE "m";1;"FILETWO"
```

Once the file has been ERASEd add the following line to the program above:

```
25 PRINT X,
```

Now when you RUN the program the numbers will be printed out to the screen, but when the cartridge is being written to, output to the screen is stopped.

### Reading a sequential file

As soon as you have CLOSEd a write file it can no longer be written to. In other words it has become a READ file. It is now possible to identify any such files whenever you do a CATalogue of that tape. You've found out how to send data out to Microdrive but how do we send it back in? That is, how do we read a sequential file?

Just as we use PRINT # to send data to the file we can use the statements INPUT # and INKEY$ # to read from the file. Of course before we can read the file it has to be OPENed as before:

```
OPEN # stream;"m";drive;"file"
```

As with a write file a channel area and buffer are created. The first time that you read from the file the buffer is filled, the drive reaccessed, and more data is read in once this buffer becomes empty. To read back the data from the file which we have just saved onto cartridge we could write a program such as:

```
10 OPEN #5;"m";1;"FILETWO"
20 INPUT #5,x
30 PRINT x
40 GO TO 20
```

Because the file "FILETWO" already exists the Spectrum will treat it as a read file. INPUT # will take a sequence of numbers that are stored in the channel buffer until it comes across a carriage return (the ENTER code). Therefore the above short program will print out the numbers between 1 and 1000.

The next short program will demonstrate this feature by writing a string to a write file and then reading it back in:

```
 10 OPEN #5;"m";1;"FILETHREE"
 20 PRINT #5;"HELLO"
 30 PRINT #5;"HOW ARE YOU ?"
 40 PRINT #5;"BYE-BYE"
 60 CLOSE #5
 70 REM now re-open file for re
ading
 80 OPEN #5;"m";1;"FILETHREE"
 90 INPUT #5;a$
100 PRINT a$
110 GO TO 90
```

The three strings will be saved on Microdrive as a sequential file by the first section of the program, then the file is closed. This means that when we OPEN the same file again it has become a read file. The data is now read back and printed out. It can be seen, when you RUN the program that an INPUT # will read in complete strings.

However, there are times when you may want to read in just single characters from a file. This is what we use the INKEY$ # statement for. When you use INKEY$ on a Spectrum it reads a single keypress from the keyboard. INKEY$ # does exactly the same for the Microdrive. It takes just the next character from the file buffer. Let's change lines 90 to 110 of the above program and enter in their place:

```
 90 LET a$=INKEY$#5
100 BEEP .1,10
110 PRINT a$;
120 GO TO 90
```

Before you RUN the program make sure that you have erased "FILETHREE" from the cartridge, so that it can be opened as a write file again in line 10.

When RUN you will hear a 'beep' and see each character printed in turn, including carriage returns. The beep has been inserted to show that only one character is read in at any one time.

Because using INPUT # reads in a whole line of data up to a carriage return you must be careful how you write and read your data. Type in the following program:

```
 10 OPEN #;5"m";1;"I/O TEST"
 20 PRINT #5;"HELLO";"HOW ARE YOU"
 30 CLOSE #5
```

If you were to read in the data saved in the file 'I/O TEST' you would expect the first INPUT # statement to read the "HELLO" and the second to read in the "HOW ARE YOU". Well let's try it:

```
  10 OPEN #5;"m";1;"I/O TEST"
  20 INPUT #5;a$
  30 PRINT a$
  40 PRINT #0;"first data read i
n"
  50 LET k$=INKEY$
  60 IF k$="" THEN GO TO 50
  70 PRINT "lets now read in the
 second part of the data"
  80 INPUT #5;a$
  90 PRINT a$
 100 CLOSE #5
```

Can you see the problem? When you read in what you thought would be "HELLO" the whole file was read in and printed. As previously mentioned the Spectrum reads up to the first carriage return (ENTER code) for each input. When you wrote the file onto cartridge the carriage return was not sent until after the YOU. Therefore the best way to PRINT # data is to put each output statement onto a separate line ensuring that a carriage return follows every item to be stored in the file.

### Finding the ends of files

When any of the programs above finished reading in all the data from Microdrive the programs terminated with an *end of file* message. When your Spectrum produces this error message not only is it printed at the bottom of the screen but the Spectrum actually stops at the line where you were trying to read in the item. Unfortunately this means that any program statements after the read would be missed out. In some programs this would not be a problem, but what could you do if you wished to read in a large file and use it within a program of your own. If the program 'crashed out' every time the end of the file was read your program wouldn't get far!

The following program demonstrates one way of getting around this problem.

```
  10 OPEN #5;"m";1;"EOFTEST"
  20 FOR X=1 TO 1000
  30 PRINT #5;X
  40 NEXT X
```

```
 50 PRINT #5;CHR$(0)
 60 CLOSE #5
 70 OPEN #6;"m";1;"EOFTEST"
 80 INPUT #6,A$
 90 IF A$=CHR$(0) THEN GO TO 120
100 PRINT A$;" ";
110 GO TO 80
120 PRINT "END OF FILE FOUND"
130 CLOSE #6
```

The method by which the end of file is detected is to output a character that will never be used in the data at the end of writing a file. When the file is read back in all that the program has to do is to check for this character and take appropriate action once it is found. In the program above the value 0 is used to mark the end of file (line 50). When the data is read back into the computer line 90 checks to see if it is a 0. If it is, the program continues execution from line 120. In this example this is the end of the program; however there is nothing preventing you from continuing your program and using the file of data within your own software.

### MOVE and sequential files

It has been mentioned previously that it is possible to MOVE data from one channel to another. If both these channels are on the Microdrive and one of them is a new file then the data in the old file will be written into the new one. The new file however still remains open and it is possible to add data to it. Below is a program to demonstrate this:

```
 10 REM write first file
 20 OPEN #5;"m";1;"one"
 30 FOR x=1 TO 20
 40 PRINT #5;"A"
 50 NEXT x
 60 CLOSE #5
100 REM now move data from one
file to another and add extra
data
110 OPEN #6;"m";1;"two"
120 MOVE "m";1;"one" TO #6
130 FOR x=1 TO 20
140 PRINT #6;"B"
150 NEXT x
160 CLOSE #6
```

```
 200 REM now read the new file
'two' in including extra data
 210 OPEN #7;"m";1;"two"
 220 FOR x=1 TO 40
 230 INPUT #7;y
 240 PRINT y
 250 NEXT x
 260 CLOSE #7
```

The program above writes twenty A characters to a file called "one". Then a file called "two" is created and the data from file "one" moved into it. Once all the data has been moved, 20 Bs are added to it. Finally, the last section of the program opens file "two" for reading and prints out the contents to the screen. You will find that this is 20 'A's and 20 'B's.

### The format of Microdrive data storage

The way in which data is stored onto a Microdrive cartridge is usually of no concern to the programmer. However when using machine code it may be necessary to know the format. We have included a discussion about storage here so that a complete picture of the Microdrive system is given.

Once formatted the Microdrive cartridge tape is split into a number of sectors. Each sector consists of a header and a data block. These sections of the tape can be broken down further into the following parts:

#### Header

12 bytes of data
The header – consisting of the following 15 bytes:
- A flag byte
- A sector number byte
- 2 unused bytes
- 10 bytes of the cartridge name
- Checksum byte

A gap then follows

#### Data block

12 bytes of data
15 bytes description of the record as follows:
- Flag byte
- Record number byte
- Record length – 2 bytes

Filename – 10 bytes
Checksum
The actual record – consisting of:
Data area of 512 bytes
Checksum byte

Another gap

Header, and so on

The twelve bytes of data at the start of each block are there so that the Microdrive system can identify the start of a block and so commence reading the correct area. There are ten '0's and two '255's in this block of twelve bytes.

### Microdrive channels

In an earlier part of the book we looked at the use of streams and channels. It should now be obvious that communication with any device is made using a channel. The Microdrive is no exception. Whenever communication with the Microdrive needs to take place a channel of 595 bytes is set up in the CHANS area of the Spectrum memory map. It is these bytes that the Microdrive uses to check whether the motors of a specific drive are switched on, to check which Microdrive you are accessing and so on. Interface 1 will automatically create a Microdrive channel whenever an action involving the Microdrive takes place and this channel will disappear once the required action has been taken. However the programmer can create a permanent Microdrive channel by using the OPEN # command. This channel will stay present until the user deletes it by using a CLOSE # or CLEAR # statement. A Microdrive channel consists of the following sections:

| Bytes | Contents |
|---|---|
| 0–1 | Address 8 |
| 2–3 | Address 8 |
| 4 | "M" – This has 80H added if the channel has been created by the system |
| 5–6 | Address of output routine for Microdrive |
| 7–8 | Address of input routine for Microdrive |
| 9–10 | 595 the length of the Microdrive channel |
| 11–12 | CHBYTE Counter for data area |
| 13 | CHREC Buffer number |
| 14–23 | CHNAME 10 characters of filename |
| 24 | CHFLAG Read/write flag |

25 CHDRIV Microdrive number
26–27 CHMAP Address of current map
28–39 12 bytes of data as discussed
40 HDFLAG Flag byte
41 HDNUMB Sector number
42–43 Unused
44–53 HDNAME Name of cartridge
54 HDCHK Checksum of previous 14 bytes
55–66 12 bytes of data
67 RECFLAG Flag byte
68 RECNUM Buffer number
69–70 RECLEN Length of buffer
71–80 RECNAM Characters of filename
81 DESCHK Checksum for previous 14 bytes
82–593 This is the data area used before transfer to cartridge
594 DCHK Checksum of data area.

Bytes 28–54 are used to form the header block on the tape. Bytes 55–594 are used to form the data block.

### The Microdrive map

Between the system variables and the channel information area within the Spectrum memory map there is an area reserved for use with Microdrive maps. Whenever access is made to a Microdrive not only is the channel area created but so is a map. When a cartridge is formatted it is split up into 256 sectors, although not all of these may be usable. The Spectrum needs to know which of these sectors can be used and which have already been used: this is what the Microdrive map is for. Each sector is represented in the map by a single bit (i.e. a 1 or a 0). If this bit is set to 0 then that sector can be used. However, if it is set to a 1 that tells the Spectrum that either this sector is unusable or already has data stored in it.

It is possible to examine the contents of a Microdrive map with a program such as the one below. You should make sure that the Microdrive map is at a known location by typing NEW before entering the program. After a NEW statement the Microdrive map can always be found starting at location 23792.

```
10 LET start=23792
20 OPEN #5;"m";1;"newfile"
30 FOR m=start TO start+31
40 PRINT PEEK m
50 NEXT m
```

Because there are 256 sectors and one bit is used to represent each sector there are a total of 32 bytes in each Microdrive map. This program prints out the values of each group of eight sectors. If you wish to see the state of each sector we must convert these numbers to their equivalent binary representation. By changing lines in the program, and adding more, as below, we can look at each individual sector.

```
  25 LET sector=0
  40 LET content=PEEK m
  50 FOR c=1 TO 8
  60 LET value=content/2<>INT (c
ontent/2)
  70 LET content=INT (n/2)
  80 PRINT "sector ";sector;" =
";value
  90 LET sector=sector+1
 100 NEXT c
 110 NEXT m
```

When you now RUN the program it will print out the sector number and a 1 if the sector is unusable or if it has data stored there and a 0 if it is empty. You will never need to manipulate the Microdrive maps and the above program is only for interest, but it is useful for showing you how data is stored.

Now that we have seen how information is passed between the Interface and the Spectrum it is probably a good time to explain how a program is stored. Knowing this can come in very useful when using machine code and you wish to write your own files or manipulate existing ones.

Firstly a Microdrive channel is created by the system, then all the sectors of the tape are examined and a Microdrive map is created as mentioned above. This procedure also checks to ensure that the filename for your given file does not already exist so that you don't accidentally overwrite an old file. Within the data area of the Microdrive channel there is now created a header of nine bytes. This header describes the program and is similar to the header that proceeds a program that is stored on tape. This header is made up as follows:

| Byte | Content |
|---|---|
| 1 | 0=BASIC program<br>1/2=array<br>3=machine code |
| 2–3 | Length of the block |
| 4–5 | Start address of the block |
| 6–7 | The length of the program |
| 8–9 | The start line number is used. |

The data that makes up the program is now transferred to the data area in the Microdrive channel. As soon as this contains 512 bytes the data area is emptied and the contents written to the cartridge.

When writing the data the header block signals that the next sector is found. This is then checked against the map to see if it can be used, and, if not the computer moves on to the next free sector. The 540 bytes from the work space in the channel area are then written to this sector, including the buffer contents. To show that this sector is now used the bit representing that sector is now set in the Microdrive map. Each time that this is repeated the value of RECNUM in the channel area is incremented by 1. When the final block of data is sent out to the drive the flag RECFLAG is set so that when reading in a program this last record will be marked 'end of file'.

# 5 Building a Microdrive DATABASE

Databases are probably one of the most widely used types of program. They are used in garages to keep stock of spare parts, offices to file names and addresses, and even in homes to keep track of hobbies, for example cataloguing your computer cassettes.

For those of you who have never come across a database the small introduction above should tell you that it is a way of storing a collection of related data, such as addresses, or details of a record collection. You could possibly think of a database as a kind of thumb-indexed book. You use these to store names, addresses, birthdays or indeed anything that you need to store in an organised fashion. Let's take this as an example: the book would be known in computer parlance as a 'file', then turning to a page of the book, any entry would be called a 'record', and each line of the entry is called a 'field'. Thus, a record may be something along the lines of the list below, the whole being a record and each line being a field:

| | |
|---|---|
| NAME | John Bull |
| ADDRESS1 | 21 High Street |
| ADDRESS2 | Anytown |
| ADDRESS3 | Anywhereshire |
| POST CODE | A12 3BC |
| TEL | 1234–56789 |
| BIRTHDAY | 25 September |

Alright, so you don't need a computer to store this data because you already keep it in your address book. But what would you do if you wanted to know which of your friends have birthdays this month? It would take a long time to check through a whole address book for the data that you require. This is where the advantages of the database come in. If you had your address book on a data base you could instruct the program to search the file for the names of people whose birthday is in, for example, March. Of course you don't only have to store names and addresses, I keep a record of all my computer tapes on database. The layout for my records is:

| | |
|---|---|
| NAME | Kill the nasties |
| MACHINE | Spectrum |
| MEMORY | 16K |
| FROM | ABC Software |
| PRICE | 5.50 |
| JOYSTICK | KEMP/SINC/CUR |

Usually the headings for a database can be changed so that they can be used for different applications.

The strength of a computer database as opposed to a manual filing system of any sort lies in its flexibility. Manual systems are by their nature sequential and ordered by one particular element, such as people's surnames or order numbers. Computer systems are versatile enough to enable the data to be manipulated by the user so that information can be retrieved according to a number of different criteria, rather than the one which has to be specified at the time the file is created. For example, if we create our manual address book in alphabetical order of people's names, that is the only element of the whole record, (name, address, telephone number), by which we can search. If the same information is put onto a computer database we can juggle the information round and search for addresses, telephone numbers or any other fields. With some sophisticated systems it is even possible to search across fields and produce, say, a list of all people living in London with the christian name Dave and owning grey cats. The practical advantages of such powerful and interactive systems are obvious.

There now follows a database program for the Spectrum hooked up to a Microdrive. This database allows for up to 15 characters to be used for each entry within a record, don't use any more or you will find that some of the data is missing. Of course the field length can be changed but this will be at the expense of total file storage space because you would be storing a lot of blank spaces in fields with little in them. To explain this each field is stored as, say 15 characters whether or not you have put in the maximum number.

```
 10 LET NC=0
 20   REM no of columns
 30 LET NR=0
 40   REM no of records
 50 LET R=0
 60     REM row
 70 LET C=0
 80     REM column
 85 LET CN=0
 90 LET I$=""
100 REM input
```

```
 110 LET K$=""
 120 REM key pressed
 130 LET N$=""
 140 REM file name
 150 DIM A$(100,11,15)
 160 LET A$(R+1,C+1)=""
 170 REM holds file
 180 DIM S$(1,15)
 190 REM holds search string
 200 LET F=0
 210 LET S=0
 220 POKE 23658,8
 230 REM set caps lock on
 240:
 250:
 260 BORDER 1
 270 PAPER 1
 280 INK 7
 290 GO TO 680
 300 REM options
 310 CLS
 320 PRINT AT 10,12; FLASH 1;"E
X I T"
 330 GO SUB 610
 340 REM y/n
 350 IF k$<>"Y" THEN GO TO 290
 360 CLS
 370 PRINT AT 10,12;"BYE-BYE"
 380 PAUSE 100
 390 STOP
 400 REM **END OF PROGRAM**
 410:
 420:
 430 REM ** INPUT STRING **
 440 LET i$=""
 450 IF INKEY$<>"" THEN GO TO 4
50
 460 LET k$=INKEY$
 470 IF k$="" THEN GO TO 460
 480 IF k$=CHR$ (13) AND i$<>""
THEN RETURN
 490 IF K$=CHR$ 13 THEN GO TO 4
50
```

```
 500 IF k$=CHR$ 12 AND i$<>"" TH
EN LET i$=i$(1 TO LEN i$-1): PR
INT CHR$ 8;" ";CHR$ 8;: GO TO 45
0
 510 LET i$=i$+k$
 520 PRINT k$;
 530 GO TO 450
 540:
 550:
 560 REM ** DRAW LINE **
 570 PRINT "----------------------
------------"
 580 RETURN
 590:
 600:
 610 REM ** YES OR NO **
 620 PRINT #0;AT 0,11; FLASH 1;"
Y"; FLASH 0;"es or ";FLASH 1; I
NVERSE 1;"N"; FLASH 0; INVERSE 0
;"o"
 630 LET k$= INKEY$
 640 IF k$="Y" OR k$="N" THEN R
ETURN
 650 GO TO 630
 660:
 670:
 680 REM ** OPTIONS **
 690 CLS
 700 PRINT AT 0,11; INVERSE 1;"M
AIN MENUE"'''
 710 GO SUB 560
 720 PRINT " CREATE NEW FILE
        1 "
 730 GO SUB 560
 740 PRINT " MANIPULATE FILE
        2 "
 750 GO SUB 560
 760 PRINT " SEARCH FOR RECORD
        3 "
 770 GO SUB 560
 780 PRINT " COLUMN SEARCH
        4 "
 790 GO SUB 560
```

```
 800 PRINT " LOAD FILE
         5 "
 810 GO SUB 560
 820 PRINT " SAVE FILE
         6 "
 830 GO SUB 560
 840 PRINT " EXIT PROGRAM
         7 "
 850 GO SUB 560
 860 PRINT #0;"INPUT OPTION ( 1
- 7 )"
 870 LET K$=INKEY$
 880 IF K$="" THEN GO TO 870
 890 IF K$<"1" OR K$>"7" THEN GO
 TO 870
 900 LET O=VAL K$
 910 IF O=7 THEN GO TO 310
 920 IF O=1 THEN GO TO 1010
 930 IF O=2 THEN GO TO 1610
 940 IF O=3 THEN GO TO 2960
 950 IF O=4 THEN GO TO 3360
 960 IF O=5 THEN GO TO 3670
 970 IF O=6 THEN GO TO 3970
 980 GO TO 680
 990:
1000:
1010 REM ** CREATE FILE **
1020 CLS
1030 PRINT AT 0,6;"C R E A T E
 F I L E"
1040 PRINT AT 10,10; FLASH 1;"AR
E YOU SURE"
1050 GO SUB 610
1060 IF K$="N" THEN GO TO 680
1070 CLS
1080 CLEAR
1090 DIM A$(100,11,15)
1100 DIM S$(1,15)
1110 LET F=0
1120 PRINT AT 0,8;"CREATE RECORD
"
1130 PRINT
1140 GO SUB 560
1150 PRINT "WHAT FILE NAME :";
```

```
1160 GO SUB 430
1170 LET N$=I$
1180 PRINT
1190 GO SUB 560
1200 PRINT "HOW MANY COLUMNS IN
A RECORD :";
1210 GO SUB 430
1220 LET NC=VAL I$
1230 PRINT
1240 IF NC<1 OR NC>10 THEN PRIN
T FLASH 1;"NO MORE THAN 10": GO
TO 1200
1250 CLS
1260 PRINT "FILE : ";N$
1270 GO SUB 560
1280 LET C=1
1290 PRINT "INPUT TITLE OF COL "
;C;" ";
1300 GO SUB 430
1310 LET A$(1,C)=I$
1320 PRINT
1330 GO SUB 560
1340 IF C<>NC THEN LET C=C+1: G
O TO 1290
1350:
1360:
1370 REM ** ENTER FILE DATA **
1380 IF F=1 THEN LET R=NR+1: LE
t NR=NR+1: GO TO 1420: REM ** AD
D REC **
1390 LET R=1
1400 LET NR=1
1410 REM ** ENTER DATA **
1420 CLS
1430 PRINT AT 0,10;"RECORD NUMBE
R ";R
1440 PRINT
1450 GO SUB 560
1460 PRINT "ENTER 'END' IN EACH
COLUMN TO"'"          TERMINATE F
ILE"
1470 GO SUB 560
1480 FOR C=1 TO NC
```

```
1490 PRINT "INPUT ";A$(1,C);" :
";
1500 GO SUB 430
1510 LET A$(R+1,C)=I$
1520 PRINT
1530 GO SUB 560
1540 NEXT C
1550 IF I$="END" THEN LET R=1:
LET NR=NR-1: GO TO 1610
1560 LET R=R+1
1570 LET NR=NR+1
1580 GO TO 1420
1590:
1600:
1610 REM ** MANIPULATE FILE **
1620 LET C=2
1625 LET R=1
1630 LET CN=0
1640 IF NR=0 THEN CLS : PRINT A
T 10,9; FLASH 1;"NO FILE EXISTS"
: FOR X=1 TO 100: NEXT X: GO TO
680
1650 CLS
1660 PRINT N$;AT 0,19;"RECORD NO
 ";R
1670 GO SUB 560
1680 PRINT A$(1,1),A$(1,C+CN)
1690 GO SUB 560
1700 PRINT A$(R+1,1),A$(R+1,C+CN
)
1710 GO SUB 560
1720 PRINT
1730 PRINT " 5 AND 8 TO  VIEW CO
LUMNS"
1740 PRINT " R       TO  VIEW RE
CORDS"
1750 PRINT " F       TO  VIEW FI
LE"
1760 PRINT " N       FOR NEXT RE
CORD"
1770 PRINT " L       FOR LAST RE
CORD"
1780 PRINT " M       TO  MODIFY
DATA"
```

```
1790 PRINT " K        TO  MODIFY
KEY FIELD"
1800 PRINT " H        TO  MODIFY
COL. HEADING"
1810 PRINT " A        TO  ADD REC
ORD"
1820 PRINT " C        TO  CHANGE
FILE NAME"
1830 PRINT " D        TO  DELETE
THIS REC."
1840 PRINT " Q        TO  QUIT TO
 MAIN MENU"
1850 LET K$=INKEY$
1860 IF K$="" THEN GO TO 1850
1870 IF K$="5" THEN LET CN=CN-1
1880 IF K$="8" THEN LET CN=CN+1
1890 IF K$="N" THEN LET R=R+1:
IF R>NR THEN LET R=NR: GO TO 16
50
1900 IF K$="L" THEN  LET R=R-1:
IF R<1 THEN  LET R=1: GO TO 1650
1910 IF K$="M" THEN  GO TO 2020
1920 IF K$="H" THEN  GO TO 2090
1930 IF K$="C" THEN  GO TO 2150
1940 IF K$="F" THEN  GO TO 2230
1950 IF K$="R" THEN  GO TO 2660
1960 IF K$="D" THEN  GO TO 2820
1970 IF K$="A" THEN  LET F=1: GO
 TO 1370: REM ADD REC
1980 IF K$="K" THEN  LET CN=0: L
ET C=1: GO TO 2020
1990 IF K$="Q" THEN  GO TO 680
2000 IF C+CN<2 OR C+CN>NC THEN
LET C=2: LET CN=0
2010 GO TO 1650
2020 REM ** MODIFY **
2030 GO SUB 560
2040 PRINT "NEW DATA FOR ";A$(1,
C+CN);
2050 GO SUB 430
2060 LET A$(R+1,C+CN)=I$
2070 LET C=2
2080 GO TO 1650
2090 REM ** CHANGE HEADING **
```

```
2100 GO SUB 560
2110 PRINT "CORRECT COL.HEADING
";
2120 GO SUB 430
2130 LET A$(1,C+CN)=I$
2140 GO TO 1650
2150 REM ** CHANGE FILE NAME **
2160 GO SUB 560
2170 PRINT "NEW FILE NAME ";
2180 GO SUB 430
2190 LET N$=I$
2200 GO TO 1650
2210:
2220:
2230 REM ** VIEW FILE **
2240 CLS
2250 PRINT AT 10,0;"SCREEN ,ZX
PRINTER ,RS232"
2260 LET K$=INKEY$
2270 IF K$="" THEN  GO TO 2260
2280 IF K$="Z" THEN  OPEN #2,"P"
2290 IF K$="R" THEN  FORMAT "T";
1200:OPEN #2,"T"
2300 IF K$<>"S" AND K$<>"Z" AND
K$<>"R" THEN  GO TO 2260
2310 CLS
2320 GO SUB 560
2330 PRINT
2340 PRINT "     FILE WILL SCROL
L DOWN"
2350 PRINT
2360 PRINT "      ANY KEY WILL
PAUSE"
2370 PRINT
2380 GO SUB 560
2390 PRINT
2400 IF K$="S" THEN  PAUSE 50
2410 CLS
2420 PRINT N$
2430 PRINT
2440 GO SUB 560
2450 FOR R=1 TO NR
2460 FOR C=1 TO NC
2470 PRINT A$(1,C),A$(R+1,C)
```

```
2480 IF C=1 THEN  PRINT
2490 IF K$="S" THEN  PAUSE 50
2500 NEXT C
2510 GO SUB 560
2520 LET T$=INKEY$
2530 IF INKEY$<>"" THEN  GO TO 2
590
2540 NEXT R
2550 IF K$="S" THEN  PAUSE 50
2560 LET R=NR
2570 LET C=2
2575 CLOSE #2
2580 GO TO 1650
2590 LET K$=INKEY$
2600 IF INKEY$<>""  THEN GO TO 2
590
2610 LET K$=INKEY$
2620 IF K$="" THEN  GO TO 2610
2630 GO TO 2540
2640:
2650:
2660 REM ** VIEW RECORD **
2670 CLS
2680 PRINT N$;AT 0,10;"RECORD NO
 ";R
2690 PRINT
2700 GO SUB 560
2730 FOR C=1 TO NC
2740 PRINT A$(1,C),A$(R+1,C+1)
2750 GO SUB 560
2760 NEXT C
2770 PRINT #0;"PRESS A KEY TO RE
TURN"
2780 LET K$=INKEY$
2790 IF K$="" THEN GO TO 2780
2800 LET C=2
2810 GO TO 1650
2820 REM ** DELETE RECORD **
2830 CLS
2840 PRINT AT 10,9; FLASH 1;"DEL
ETING RECORD"
2850 LET REC=R
2860 FOR R=REC TO NR
2870 FOR C=1 TO NC
```

```
2880 LET A$(R+1,C)=A$((R+2),C)
2890 NEXT C
2900 NEXT R
2910 LET NR=NR-1
2920 LET R=REC
2930 LET C=2
2940 PAUSE 50
2950 GO TO 1650
2960 REM ** SEARCH FOR RECORD **
2970 CLS
2980 IF NR=0 THEN  PRINT "NO FIL
E IN MEMORY": GO TO 690: REM RET
URN TO OPTIONS
2990 PRINT AT 3,8;"DO YOU WANT T
O"
3000 PRINT AT 7,0;"1  FIND CONTE
NT OF FIELD NUMBER     ONE"
3010 PRINT AT 11,0;"2  FIND A RE
CORD NUMBER"
3020 PRINT AT 14,0;"3  RETURN TO
MENU"
3030 LET K$=INKEY$
3040 IF K$="" THEN  GO TO 3030
3050 IF K$="3" THEN  GO TO 690
3060 IF K$="1" THEN  GO TO 3090
3070 IF K$="2" THEN  GO TO 3160
3080 GO TO 3030
3090 REM SEARCH FOR RECORD BY FI
ELD NUMBER ONE
3100 CLS
3110 PRINT AT 5,0;"ENTER THE CON
TENTS OF FIELD     NUMBER ONE TO
BE SEARCH FOR"
3120 PRINT AT 8,0;
3130 GO SUB 420
3140 REM INPUT NAME
3150 GO TO 3260
3160 REM SEARCH FOR A NUMBERED R
ECORD
3170 CLS
3180 PRINT AT 3,0;"THERE ARE ";N
R;AT 4,0;"RECORD IN THIS FILE"
3190 PRINT AT 7,0;"WHICH RECORD
NUMBER DO YOU WANT"
```

```
3200 PRINT AT 10,15;
3210 GO SUB 420
3220 REM  INPUT RECORD
3230 LET R=VAL I$
3240 IF R=0 OR R>NR THEN  GO TO
3160
3250 GO TO 1650
3260 REM SET UP SEARCH FOR FIELD
ONE
3270 FOR R=1 TO NR
3280 LET S$(1)=I$
3290 IF S$(1)=A$(R+1,1) THEN  GO
 TO 1650
3300 NEXT R
3310 CLS
3320 PRINT AT 10,0;"ENTRY NOT FO
UND"
3330 PAUSE 50
3340 GO TO 680
3350 FOR R=1 TO NR
3360 REM SEARCH FOR COLUMN DATA
3370 CLS
3380 LET E=0
3390 FOR R=1 TO NR
3395 CLS
3400 PRINT "WHAT IS THE COLUMN O
F INTEREST "
3410 PRINT AT 4,10;
3420 GO SUB 420
3430 LET S$(1)=I$
3440 LET C=1
3450 IF A$(1,C)=S$(1) THEN  GO T
O 3510
3460 LET C=C+1
3470 IF C<NC THEN  GO TO 3450
3480 CLS
3490 PRINT FLASH 1;"CONTENTS OF
COLUMN NOT FOUND": PAUSE 100
3500 GO TO 680
3510 CLS : PRINT "WHAT ";S$(1);'
"DO YOU WISH TO SEARCH FOR ?"
3520 PRINT AT 4,10;
3530 GO SUB 420
3540 LET H$=S$(1)
```

```
3550 LET S$(1)=I$
3560 CLS
3570 LET S=0
3575 FOR R=1 TO NR
3580 IF A$(R,C)=S$(1) THEN  PRIN
T "RECORD NUMBER ";R-1: LET S=1
3582 NEXT R
3585 IF S=1 THEN  GO TO 3630
3610 PRINT AT 10,0; FLASH 1;"FOU
ND NO SUCH DATA"
3630 PRINT #0; FLASH 1;"PRESS AN
Y KEY TO CONTINUE"
3640 LET K$=INKEY$
3650 IF K$="" THEN  GO TO 3640
3660 GO TO 680
3670 REM LOAD THE FILE
3680 CLS
3690 PRINT AT 4,10;"LOAD FILE"
3700 PRINT AT 8,9;"ARE YOU SURE"
3710 GO SUB 610
3720 IF K$<>"Y" THEN  GO TO 690
3730 CLS
3740 PRINT AT 6,5;"WHICH MICRODR
IVE (1-7)"
3750 LET K$=INKEY$
3760 IF K$<"1" OR K$>"7" THEN  G
O TO 3750
3770 LET N=VAL K$
3780 CLS
3790 CAT N
3800 PRINT AT 21,0;"FILE NAME";
3810 PRINT ;AT 21,10;"""
  """
3820 PRINT AT 21,11;
3830 GO SUB 430
3840 LET N$=I$
3850 OPEN #5;"M";N;N$
3860 INPUT #5;NR
3870 INPUT #5;NC
3880 FOR X=1 TO NR+1
3890 FOR Y=1 TO NC
3900 FOR Z=1 TO 15
3910 INPUT #5;A$(X,Y,Z)
3920 NEXT Z
```

```
3930 NEXT Y
3940 NEXT X
3950 CLOSE #5
3960 GO TO 690
3970 REM SAVE FILE
3980 CLS
3990 PRINT AT 4,10;"SAVE FILE"
4000 PRINT AT 8,9;"ARE YOU SURE"
4010 GO SUB 610
4020 IF K$<>"Y" THEN  GO TO 690
4030 CLS
4040 PRINT AT 6,5;"WHICH MICRODR
IVE (1-7)"
4050 LET K$=INKEY$
4060 IF K$<"1" OR K$>"5" THEN  G
O TO 4050
4070 LET N=VAL K$
4080 OPEN #5;"M";N;N$
4090 PRINT #5;NR
4100 PRINT #5;NC
4110 FOR X=1 TO NR+1
4120 FOR Y=1 TO NC
4130 FOR Z=1 TO 15
4140 PRINT #5;A$(X,Y,Z)
4150 NEXT Z
4160 NEXT Y
4170 NEXT X
4180 CLOSE #5
4190 GO TO 690
```

Once the "DATABASE" has been LOADed and RUN the display will show a menu of 7 options available to the user.

These are:

| | |
|---|---|
| CREATE NEW FILE | 1 |
| MANIPULATE FILE | 2 |
| SEARCH FOR RECORD | 3 |
| COLUMN SEARCH | 4 |
| LOAD FILE | 5 |
| SAVE FILE | 6 |
| EXIT PROGRAM | 7 |

You will then be prompted to reply with your choice from the following:

1. Create a new file. Following this option through you are able to create a file under whichever name you specify, then choose the number of columns you wish to allocate to each record. Remember to type END at the end of each column when you have entered all your data. At this point you will then automatically continue into option 2 where you are given the chance to check the file and make any alterations you feel necessary.

2. Manipulate a file. This option can only be chosen if you have already LOADed a file through option 5 or if you have just created a new file through option 1. This option is designed to enable you to alter any particular record or column, or alternatively to examine the complete file. You are able to move between alternate records at the touch of a certain key. This gives you the flexibility to do any or all of the following: examine the contents of the record all at the same time; look through the columns of the record to find a single piece of data or change the data/title of that field; delete a complete record; add a new one or even change the data of the key field for that particular record.

3. Search for a record. This option will allow you to find a particular record by means of the record number or the content of the first field of the file. Whichever you choose you will then be given the chance to manipulate the record via section 2.

4. Column search. This option is designed to produce the number of the record you wish to investigate. You will be required to specify the title of the column to be searched as there may be as many as 10.

5. LOAD a file. This section gives you the facility for LOADing an existing file from the Microdrive. You are asked to confirm that this is exactly what you wish to do because the LOADing process will overwrite any file already in memory. If you don't want to lose the last file you were working on you must remember to SAVE it before LOADing another file. You are able to specify which of 7 Microdrives the file is on and then the name of the file. The computer will then LOAD the file and return you to the main menu.

6. SAVE a file. This section also gives you a chance to change your mind if you want to. Once the computer has SAVEd your file it will return to the main menu.

7. Exit the program. Choosing this option will make you leave the program and any file that you have in memory will be lost, so be sure to SAVE it if it is important.

**Detailed description**

The program works as follows:

*Lines 10–210* sets up the arrays and strings necessary to run the database.
*230* sets CAPS LOCK on, so all input appears in upper case.
*260–290* sets up the screen colours.
*300–400* sets up the exit routines and displays.
*430–530* checks the input to a record to see that something has been entered, checks whether you want to delete a character and if so carries this out then checks that a RETURN is pressed at the end of a record.
*560–580* is a display subroutine.
*610–650* is a routine to display a flashing Yes/No option and check your answer, taking appropriate action.
*680–980* displays the main menu options and takes action according to your response.
*1010–1340* sets up the database file and allows you to title each record within the file.
*1370–1580* controls the entry of data to the file.
*1610–2200* sets up a secondary menu with options to change the data entered and so on.
*2230–2630* is the subroutine that controls the file viewing system and allows you to browse through the file.
*2660–2810* handles the individual record viewing system.
*2820–2950* allows the deletion of an individual record.
*2960–3660* this routine sets up a search for a particular record depending on the area of interest, that is to say, records can be scanned for:

i) contents of a particular field, number 1 to 10.
ii) a particular record number.
iii) the contents of a particular column.

*3670–3960* loads the data file to memory for processing or scanning.
*3970–4190* saves the data file to a selected Microdrive.

This program is meant as a demonstration and gives guidelines as to how useful a database can be. It uses the Microdrive as a mass storage device, dumping and retrieving data as necessary. The program is complete but slightly restrictive, however it does point the way for your own program structure, and can be developed further as your needs and experience also develop.

# 6 Interfacing

The term 'interfacing' is used to describe the connecting up of a peripheral to a computer for data transmission between the two. In this sense, the cassette recorder that you have been using with the Spectrum to store and retrieve programs can be said to be interfaced with the Spectrum itself, through the cassette lead. In the same way, the TV can be thought of as being interfaced with the computer via the TV lead. Although both these pieces of equipment are strictly speaking peripheral to the Spectrum, they are so commonly used and so essential to the proper running of the machine that they are rarely thought of as being 'interfaced', rather as being simply 'connected' to the Spectrum. The connections are incorporated into the body of the computer and the machine itself takes care of all the necessary programming to permit the use of these items. For instance, sending information to the video screen is accomplished simply by using the BASIC command PRINT while SAVE automatically sends a program to the cassette recorder. In fact, the processes involved are so simplified by the computer that we more or less take for granted that the TV and cassette recorder *must* be connected to make a complete, albeit rudimentary, computer system.

The Sinclair ZX Printer was specifically built for connection to the Spectrum and so it is designed to be as easy as possible to use with it. This is not the case with all computer-associated equipment. For instance, you will not be able to connect any other printer to the standard Spectrum because there is neither a connecting socket for a standard printer nor supporting software built-in. On the other hand, you can interface anything else to your Spectrum providing that you can buy or construct the necessary hardware interface, connecting cable and software. Indeed, this has come to be the commonly accepted definition of interfacing: connecting a computer to non-specific equipment using a specialised hardware connection. The definition is becoming increasingly blurred as the relentless march of technology delivers to the markets of the world computers that with each passing year have an increasing number of I/O ports built in as standard. Indeed, the NEC PC-8201A portable computer upon which most of this book was written has no less than six I/O ports, including one for a bar-code reader!

There are several methods of interfacing in current use but they can be

divided into two main categories depending upon the method of data transmission, *serial* or *parallel*. To understand the reason for this division you need to know that computers store data within their memory as a series of electrical charges, each representing two states: 0 (off/ no charge) or 1 (on/ positive charge). A series of on/off charges can be combined to form an electrical equivalent of a binary (base 2) number. Each individual on/off element is known as a BIT (an abbreviation of Binary digIT) and larger numbers are represented by grouping BITs together. Most home computers use a standardized grouping of eight BITs, which are known collectively as a BYTE (a corruption of by-eight). All information is stored within the computer as a collection of bytes.

Interface systems are categorised according to the method that is used to transmit each byte of data: either in parallel or serially.

## The parallel interface

An interface is of the 'parallel' type if the eight BITs of each BYTE are transmitted simultaneously. To permit this, a separate wire or line is used for each BIT, a total of eight lines for transmitting a BYTE of data.

The two most common parallel interface standards are Centronics and IEEE-488. The Centronics port was developed by the company of that name which was among the first to manufacture computer printers. The method they used was so flexible and reliable that it has now become an industry standard interface. Eight wires are used for the transmission of data plus a single wire to control the timing, so that everything is co-ordinated. An additional tenth line allows the receiving peripheral to signal to the host computer that it is 'busy' and cannot accept any more data. These last two lines constitute the control part of the interface and, between them, they ensure that the data is sent properly and does not become scrambled or corrupted in the process. Known as *handshaking*, the ability to transmit timing and control signals is essential to the smooth running of any interfacing system. The Centronics standard, having been developed by a printer manufacturer, offers only a rudimentary form of interfacing, eminently suitable for transmissions involving a printer but not providing sufficient control for communication between two computers or other such sophisticated equipment.

The other common parallel standard is the IEEE-488 interface. This system employs additional lines which allow communication between the computer and any one of several peripherals that can be connected simultaneously. Peripherals that use the IEEE-488 standard must be intelligent in the sense that they must be able to recognise when a transmission is directed at them and ignore those that are not. Whilst the IEEE-488 standard is commonly used on large computers its implementation on home computers is very rare, with the notable exception of Commodore

computers which use a version of it for the connections to their peripherals.

### The serial interface

The alternative method of data transmission is serial transmission. As the name suggests, each BYTE is transmitted as a *series* of BITs along a single wire, hence serial transmission. Serial transmission by one means or another, is very common in home computers. For instance, the cassette connections of all home computers utilise serial transmission of data and Sinclair's innovative NET does also. The *de facto* industry-wide serial standard is the RS-232 interface, which we shall look at now because this is one of the extra facilities afforded to the Spectrum by the connection of the Interface 1.

The RS-232 interface has become the standard for serial transmission and receipt of data for two reasons. Firstly, the standard is a very flexible one which allows a wide choice of format in which the data can be presented and the speed with which it is transmitted. Secondly, there is quite a wide choice in the complexity with which the interface is actually implemented. It may, at first, appear strange that a so-called standard method of communication should have such a wide variation but, in fact, this is the very reason that the RS-232 interface is so popular (and increasingly so): because it allows communication between virtually any two pieces of equipment, so long as they both have some form of RS-232 port.

In its simplest form, the RS-232 interface can be used with just two wires: one to carry the data transmission and the other to act as a common earth between the two machines. RS-232 is a two-way system so a third wire can then be added that is used to receive data, if the peripheral itself is capable of transmitting. A printer, for instance, need only be used with a two-wire connection, whereas communications between two computers will require a minimum of three wires. As we shall see later, other wires can also be added which act as controls for the interface and make the communication faster, easier and more reliable. But first of all let us look at how the data is actually transmitted.

We have already seen that a single BIT can represent either a 0 or a 1. The RS-232 uses +12 volts to represent a BIT value of 1 and −12 volts for a BIT value of 0, while a series of BITs is transmitted quite simply as a series of changes of voltage. Working on these principles this simple system could work very well until a single BIT was misread at the receiving end when the data following would become corrupted by the single error. Indeed, the problem would be particularly difficult if a BIT were not misread but missed altogether, in which case all the following BITs would be one out and the result would be total confusion. This potential problem

is overcome by the practice of surrounding each complete byte with one *start BIT*, (which signals that a byte is about to be transmitted), and one or two *stop BITs* which signal the end of the byte. In this way, should a byte become corrupted the following bytes of data will not be affected because the system will treat each byte individually, looking for the start and stop bits which delimit each byte. As an added measure to detect possible errors there is also a check-sum BIT known as the *parity BIT*. (See the end of this chapter for a full explanation of all the terms associated with the RS-232 interface). In the simple, three-line interface the parity BIT has no importance but in the more complex versions of RS-232 a parity error can be used to request the re-transmission of the corrupted byte, thereby making the interface very much more accurate.

The next task required for successful communication is the synchronization of the speed of transmission with the speed of receipt. With the Centronics interface this is achieved by using a separate line to transmit a continuous series of timing pulses. This method is not employed with the RS-232 interface but, instead, the speed of communication is set at both ends to be one of up to nine standard settings (the same for both machines, of course!). The speed is measured in BITs per second, more usually known as the *Baud* rate after the Belgian engineer *Baudot* who, in 1860, invented the teletype machine which uses this method of communications.

## Connections and conventions

The following lists, whilst not necessarily definitive, describe and detail the various connections that can be made with the RS-232 interface and the meanings of the various terms involved in establishing a protocol.

*Connections*

**GND** – The protective ground line. This is used to establish a common earth between both machines and must be connected.

**TxD** – Transmit data. This is used to transmit the data from your computer. This line should be connected to the RxD connection on the second computer or peripheral.

**RxD** – Receive data. This is used to receive the data into your computer. This line should be connected to the TxD connection on the second computer or peripheral.

**RTS** – Request to send. This is one of many control lines and is used by your computer to notify the receiving peripheral that some data is about to be sent. Unlike the first three connections in this list, this line and those that follow are optional. If implemented, RTS should be connected to DSR on the receiving peripheral.

**CTS** – Clear to send. This line is used to signal to the peripheral that the computer is ready to receive data from it. If implemented, CTS should be connected to DTR on the peripheral.

**DSR** – Data set ready. This line is used by the peripheral to indicate that it wants to send some data. If implemented, DSR should be connected to RTS on the peripheral.
**GND** – Signal ground. A second ground between the two machines that establishes a common zero voltage for the data signals.
**DCD** – Data carrier detect. This line is used to indicate that there is a peripheral attached to the end of the cable.
**DTR** – Data carrier ready. This line is used by the peripheral to indicate that it is ready to receive data. If implemented, DTR should be connected to CTS on the peripheral.

### Conventions

**Baud rate** – A measurement of the speed of transmission of data, in BITs per second. The standard Baud rates are 75, 110, 300, 600, 1200, 2400, 4800, 9600 and 19200. Note that not all RS-232 interfaces support all these speeds, the higher rates sometimes being left out because of the increased possibility of corruption of data.
**Parity** – This is an optional error-checking BIT selected by the user to conform with the requirements of the receiving peripheral. There can be: no parity sent, an even number parity or an odd number parity. Even number parity implies that the sum total of all the BITs in a byte plus the parity BIT must be an even number. Thus, if the total for the byte (ignoring the parity BIT) is odd (ie. 10010010 which has three BITs set to 1) then the parity BIT will be set to a 1, making the total an even number (in this case, 4). On the other hand, if the total of set BITs in the byte is already an even number then the parity BIT is reset to zero. If, during transmission, any BIT within the group is corrupted then the total on reception will be an odd number and an error will be indicated. Odd number parity works in the same way except that the total for each byte plus the parity BIT is set to be an odd number. The use of the parity BIT is optional and is not usually necessary when using short cables and slow Baud rates, although both machines must use the same convention.
**Word length** – Some RS-232 interfaces allow the user to select the number of BITs per byte that are to be transmitted or received: 5, 6, 7 or the full 8. If less than 8 BITs are used then the higher BITs are not taken into account.
**Start BIT** – This BIT is always sent and it indicates to the receiving peripheral that a byte of data is about to be transmitted.
**Stop BIT** – This BIT indicates that the data byte has finished. Some RS-232 interfaces allow a choice in the number of stop BITs that are transmitted: 1, 1.5 or 2.
**Protocol** – This defines the requirements of a particular computer or peripheral. The required protocol for the Spectrum with Interface 1 is that

parity should be off, eight BITs of data are sent and one stop BIT is used. These, therefore, are the settings that you should use for any piece of equipment or peripheral that is connected to the other end of the Spectrum RS-232 cable.

# 7 Using the Spectrum RS-232 interface

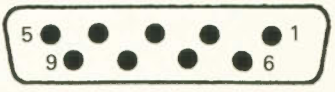
### Connecting peripherals to the RS-232 interface

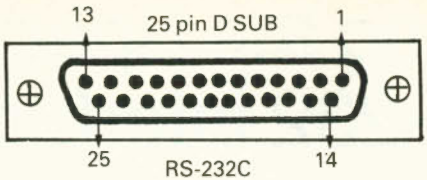
In the last chapter we looked at the connections and conventions of the various industry-standard interfaces. By contrast, in this chapter we shall describe and detail the more practical side of interfacing: the actual communication between various devices using the RS-232 port that is built into the Interface 1.

Before your Spectrum can talk to the outside world you will have to beg, borrow or more likely buy a cable. The proper RS-232 cable is available from Sinclair but it is a simple task, and much cheaper, to make one yourself. You will need a 9 pin D-plug for the Spectrum end of the cable, a length of 5 core cable, (about 2-3 metres is convenient), and also a plug for the second computer. Although the RS-232 interface is supposed to be a standardised affair, most manufacturers make their own choice as to which type of socket should feature on their particular computer so you will have to look at the manual of the chosen computer to find out exactly which plug is needed. Most computers use a 25-way Amphenol plug but home computers vary considerably. The BBC Micro, for example, uses a 5 pin DIN plug whilst the Sinclair QL uses a standard GPO type jack and, of course, the Spectrum has a 9-way connector. The pin connections for the Spectrum are shown below, along with those for the standard 25-way socket.

The RS-232 socket is wired as follows:

1. No connection
2. TX data (input)
3. RX data (output)
4. DTR (input) this should be high when ready
5. CTS (output) this should be high when ready
6. n.c.
7. Ground (pull down)
8. n.c.
9. +9v (pull up)






| Pin number | Signal name | Remarks |
|---|---|---|
| 1 | GND | Protective ground |
| 2 | TxD | Transmit data |
| 3 | RxD | Receive data |
| 4 | RTS | Request to send |
| 5 | CTS | Transmission authorized |
| 6 | DSR | Data set relay |
| 7 | GND | Signal ground |
| 8 | DCD | Data carrier detect |
| to | | |
| 20 | DTR | Data carrier ready |
| 22 | RD | Bell detect |
| 25 | . . . | |

When you have the cable wired up correctly you can then hook everything together and start communicating. The first thing that you will probably notice, especially if you try to transmit or receive without any preparation, is that nothing at all happens, or if anything does happen then it will be something odd and unplanned! This is usually the result of not setting the protocol between the two devices correctly. The Spectrum can run at the following baud rates: 50, 110, 300, 600, 1200, 2400, 4800, 9600 and 19200. You must choose a rate that is compatible with the second device. What is more, if you are to communicate under the control of a BASIC program, you will be very unlikely to achieve satisfactory results at rates greater than 110–300 (50 baud is best) because the program runs too slowly.

The Spectrum will always transmit data using the full eight bits plus one stop bit, so this setting should be used for the other device also. The Spectrum performs no parity checking, nor does it generate a parity bit so the setting should be adjusted to 'no parity' on the receiving device. Finally, most printers have a switch to control what is known as the auto line-feed sequence. When this is turned on the printer will automatically perform a line-feed (CHR$(10)) whenever the carriage-return character (CHR$(13)) is received. As we shall see shortly, the Spectrum has in fact two RS-232 channels that can be used either independently or together. One of these channels, known as the *text* or 't' channel, automatically sends the line-feed whenever it sends a carriage-return so there is no need for the printer to generate its own line-feed. By way of contrast, the *binary* or 'b' channel sends only the carriage-return so the printer would, in this case,

have to generate its own line-feed so as to advance the paper. It is very easy to tell which setting you should be using because the print out will be either correct, or double spaced (auto line-feed should be off but is actually on) or all on one line (the switch should be on but it is off). Once you have sorted out these preliminary obstacles, and they only need to be sorted out once, you are at last ready to start communicating. You may feel by this point that it is easier and quicker to communicate by writing a letter and posting it but perseverance pays its own rewards!

### "t" and "b" channels

The software for the RS-232 interface of the Interface 1 has been especially written to allow the easy use, wherever possible, of the communication channels that are available. Two channels have been incorporated and, whilst both can be in use simultaneously, they are each designed to perform a specific task more easily than would otherwise be possible. The "t" or text channel is principally used for transmitting or receiving text, especially program listings. The second channel is the "b" or binary channel, which allows a more general form of communication for particular applications and also where the use of the "t" channel is considered inappropriate.

*The "t" channel*

The "t" channel is normally used to send program listings and simple data or text. It does not transmit bytes directly but first of all 'manipulates' them. Values between 0 and 31 are ignored completely. The only exception to this is the carriage return (CHR$(13)). When this character is encountered the Spectrum will send it, followed by a line feed (CHR$(10)) so as to advance the paper.

The standard ASCII coded data, from 32 to 127, is sent down the line completely unaltered. This is done so as to provide compatibility with other standardised peripherals and computers.

The Sinclair Spectrum has a set of customized graphics characters built into its character set. The characters from 128 to 143 are pre-defined block graphics and those from 144 to 164 are user-definable. Whilst the ZX Printer is able to print these characters, most other printers are not equipped to do so. For this reason, the "t" channel replaces the characters within the range 128 to 164 with a question mark to indicate that a character is missing.

Keywords within a program are tokenized before being placed into memory, and as you can see from a quick perusal of Appendix A in the Sinclair user manual, the values from 165 to 255 are used for this purpose. When a program is listed to the screen all these keyword tokens are converted back into the full form of the words before being printed. The

same process takes place when the "t" channel is used so that, for instance, the word FORMAT will be sent to the RS-232 port whenever the byte 208 is encountered. Obviously, this makes the printing of program listings on an RS-232 printer much more simple.

Before you can use the 't' channel, you must first set the transmission speed to the required Baud rate. On first powering up the Spectrum sets a default value of 9600 Baud, so if you want to use this rate you need do nothing further. Other rates can be set by specifying them within a FORMAT statement:

```
FORMAT "t";baud rate
```

If you try to set a rate which is not recognised by the Interface 1 as one of the standards the system will automatically select the next lowest Baud rate. For example:

```
FORMAT "t";250
```

will set the rate at 110 Baud, not 250 Baud. It is always worthwhile experimenting with the Baud rates, whenever possible, so that you can find the highest usable rate and thus reduce transmission time to the minimum that is compatible with reliable communications. As a general rule, you will find that with program listings, SAVEing and LOADing can be carried out at a considerably faster speed than communication from within a BASIC program: it is the slow running speed of the program that is the limiting factor, not the RS-232 port itself.

Once the Baud rate has been set, you must then OPEN a channel to the RS-232 port by entering:

```
OPEN #3,"t"
```

If you wish to LIST a program to an RS-232 printer you should enter these commands in direct mode, otherwise they can be entered as program lines. Finally, to send a program listing to the printer, use:

```
LLIST
```
or
```
LIST #3
```

and, if there is a program in memory it will be output to the printer and the border will flash whilst the transmission is in progress.

Text can also be sent to the RS-232 port using the LPRINT or PRINT #3 commands.They too can be used interchangeably and either in direct mode or within a program. As an example, try transmitting your own

message using LPRINT in direct mode. Don't forget that you must set the Baud rate and OPEN a channel first . Note that LPRINT has the same syntax and conventions as the simpler form PRINT. You can use the print modifiers, the comma, semi-colon and apostrophe with LPRINT but you cannot LPRINT AT or LPRINT PAPER, INK or BORDER since these have no relevance to a printer.

An example of using the "t" channel to work with an RS-232 interfaced printer is shown in the next short program.

```
  10 FORMAT "t";1200: REM baud r
ate to suit your printer
  20 OPEN #5;"t"
  30 FOR x=1 TO 255
  40 PRINT #5;CHR$ x;
  50 NEXT x
  60 CLOSE #5
```

This program will send out to the printer the entire ASCII character set through the "t" channel. The control characters, with the exception of 13, (the carriage return), will, however, be ignored.

If your Spectrum is connected to a terminal or a second computer you will be in a position to receive as well as transmit data through the "t" channel. See the section later in this chapter for more details on how to achieve this two-way communication.

### The "b" channel

The "b" channel should be used whenever you wish to send your bytes of data in an unaltered form. The "b" channel, or binary channel, will always send the full eight BITs of each byte, unlike the "t" channel which only sends the lowest seven. Also, when using the INPUT# and INKEY$# statements, the "b" channel will accept and use 8 BIT data which the "t" channel does not do.

The "b" channel is ideal for sending control codes to activate the extra facilities available on a printer. The Epson range of printers, for example, offers a wide range of printing options ,any of which can be selected by sending an 'escape' sequence to the printer. The next short program demonstrates how to control the printer and shows a few of the facilities available. For a complete list of control codes you should refer to the manual of your printer.

```
  10 REM ********************
  20 REM * "b" CHANNEL DEMO *
  30 REM ********************
  40 FORMAT "b";300
  50 OPEN #3;"b"
  60 PRINT #3;CHR$ 27;CHR$ 40 :
REM RESET PRINTER
  70 PRINT #3;"THIS IS NORMAL PR
INTING"
  80 PRINT #3;CHR$ 27;"E";: REM
EMPHASISED PRINT
  90 PRINT #3;"DARK PRINT FOR EM
PHASIS"
 100 PRINT #3;CHR$ 27;"W";CHR$ 1
 110 PRINT #3;"DOUBLE WIDTH CHAR
ACTERS"
 120 PRINT #3;CHR$ 27;"W";CHR$ 0
 130 PRINT #3;"BACK TO NORMAL"
 140 PRINT #3;CHR$ 27;"M"
 150 PRINT #3;"ELITE CHARACTERS"
 160 PRINT #3;"AND THERE IS PLEN
TY MORE CHOICE"
 170 PRINT #3;"SEE YOUR PRINTER
MANUAL FOR DETAILS"
 180 CLOSE #3
 190 STOP
```

You may find that the program listed above does not work properly with your printer. If the paper is not advancing with each line of print then you should reset the dipswitch that controls the auto-line feed so that a line feed is carried out when a carriage return is received. The 'b' channel does not generate a carriage return/line feed combination, only a carriage return.

If the program used previously to send all the ASCII character set to an RS-232 interfaced printer is amended for the 'b' channel by changing 't' in lines 10 and 20 to 'b', when RUN it will probably cause the printer to print odd characters, beep or even stop working. This is because when using the 'b' or binary channel all the control codes are being sent unchanged, so the control functions are being carried out. Not very useful, perhaps, but the amended program does demonstrate the full range of possibilities available.

It is possible to have both the 't' and 'b' channels open at the same time and to send the relevant data to the appropriate channel, that is, to obtain text transfer via the 't' channel and control codes via the 'b' channel. This

can be particularly useful because the 't' channel sends a carriage return/line feed combination to advance the paper, whereas the 'b' does not. So, in the program listed above, you could PRINT all the text through the 't' channel, thereby generating line feeds, instead of altering the dipswitch on the printer.

*The use of SAVE and LOAD*

It is possible to send over the RS-232 link not only data but also any command that can be used with a channel or stream. For example, you might wish to transfer a program in memory to another computer, possibly by using a telephone link via a modem and you could use the SAVE command. SAVE and LOAD will only work through the "b" channel, never with the "t" channel.

To transfer a program through the RS-232 link enter the SAVE command in the form:

```
SAVE *"b"
```

The program that is held within the memory of the Spectrum will be sent to the receiving computer via the "b" channel. If you and a friend both have a suitable cable, or even modems, then you could send your programs to each other by one, the sender, using the above statement, and the other, the receiver, using:

```
LOAD *"b"
```

The BASIC program you send will then be LOADed, together with all the variables, through the RS-232 interfaces and modems (if used), to the second computer. Of course you can also send CODE and SCREEN$ over an RS-232 link using the following statements:

```
SAVE *"b" CODE Start address,End address
SAVE *"b" SCREEN$
```

Another extension that you can also use is:

```
SAVE *"b";LINE number
```

If you wished to VERIFY the data that had been sent to your Spectrum over an RS-232 link then you would have to arrange for the data to be sent to you again. Then you would have to type:

```
VERIFY *"b"
```

The data coming through the RS-232 port will now be checked against that in memory to make sure that it is the same.

*Simple test program for transmission through the RS-232 port*

First enter FORMAT"b";baud rate, and make sure it is the same at both ends of the system.

Now enter:

```
 5 REM figures
10 FOR n=1 TO 10
20 PRINT n,n*RND
30 NEXT n
```

Next type, in direct mode:

SAVE *"b"

As before, the second computer, waiting to receive the program, will already have set the baud rate (FORMAT) and will have entered, before you enter SAVE, the statement:

LOAD *"b"

### The RS-232 and the memory maps

We already know that whenever you OPEN a channel to a Microdrive then an area of memory is taken up in the channel area of the Spectrum memory map. The same applies to the RS-232 interface. Whenever you open an RS-232 channel 11 bytes are added to the channel area. These are:

| Bytes | Contents | |
|---|---|---|
| 1–2 | 8 | |
| 3–4 | 8 | |
| 5 | "t" | |
| 6–7 | Address of | : #0C3C for "t" |
| | output routine | : #0C5A for "b" |
| 8–9 | Address of | : #0B6F for "t" |
| | input routine | : #0B75 for "t" |
| 10–11 | Number of bytes | : #00B |

Note: numbers preceded by # are hexadecimal numbers, to the base sixteen. The use of hexadecimal notation is fully discussed in Chapter 10.

The baud rate is the same for any of the RS-232 channels that you have OPENed. When you FORMAT an RS-232 channel to set the baud rate the value is stored in location 5CC3H (23747). This address is one of the Interface 1 system variables. If you wish to set the baud rate yourself then this is done by using the formula:

VALUE = INT(3500000/(BAUD RATE*26))−2

The VALUE is stored in location 5CC3 Hex.

### 'Talking' to other computers

As has been previously mentioned, the Spectrum can communicate with any other device that has a standard RS-232 port. This ability unlocks the possibility of communicating not only with peripherals such as printers, but also with many other makes of computer, if they also have an RS-232 connection. In many ways this can be the most fascinating aspect of communications: you can, with suitable programming, allow an IBM mainframe to access the full power and potential of your Spectrum! Alternatively, you can interface your Spectrum to another computer so as to access its peripherals, disk-drive, printer, etc.

Half the fun of communicating between two computers is the self-discovery of the programming method necessary to achieve the final objective. So as not to diminish your pleasure we shall not provide you with a set of ready made routines that can be used with a variety of the most popular home computers. We will, however, get you started on this quest for greater knowledge by providing two demonstration programs that allow messages to be transmitted simultaneously between a Spectrum and a Sinclair QL. The first listing is for the Spectrum:

```
 10 REM **********************
 20 REM *   SPECTRUM TO QL   *
 30 REM *       RS-232       *
 40 REM *    DEMO PROGRAM    *
 50 REM **********************
 60 FORMAT "T";300
 70 PAPER 0: BORDER 0: INK 7
 80 CLS
100 REM ********************
105 REM *     TRANSMIT      *
110 REM ********************
120 OPEN #4;"T"
130 LET A$=INKEY$
140 IF A$="" THEN GO TO 200
150 IF A$<>"" THEN GO TO 150
160 PRINT INK 7;A$;: PRINT #4;A
$;
200 REM ********************
205 REM *     RECEIVE       *
210 REM ********************
220 LET B$=INKEY$#4
230 IF B$="" THEN GO TO 250
240 PRINT INK 6;B$;
250 GO TO 130
```

and this listing is the equivalent program for the Sinclair QL:

```
 10 REM **********************
 20 REM *     QL TO SPECTRUM *
 30 REM *        RS-232      *
 40 REM *     DEMO PROGRAM   *
 50 REM **********************
 60 BAUD 300
 70 OPEN #3,SER1OHC
100 REM ********************
105 REM *      TRANSMIT      *
110 REM ********************
120 A$=INKEY$
130 IF A$="" THEN GO TO 200
140 PRINT A$;: PRINT #3,A$;
200 REM ********************
205 REM *      RECEIVE       *
210 REM ********************
220 B$=INKEY$(#3)
230 IF B$="" THEN GO TO 250
240 PRINT B$;
250 GO TO 120
```

**Remote Networking**

Generally speaking the principal of remote networking is exactly the same as local networking with the Spectrum, with the exception that the RS-232 interface is required, and for truly remote networks, using a telephone or radio link, a *modem* is required.

Control statements are the same as when using the RS-232 Interface for local communications, but additional commands may be required for the modem.

*What is a modem?*

The name *modem* stands for MOdulator/DEModulator.

Basically a modem is an interfacing device which permits a computer or visual display unit to communicate with other computers or 'intelligent' peripheral units over a telephone or radio link, using audio tones instead of data pulses.

The use of audio tones (audio *MOD*ulation) allows the information to be sent over systems having a limited audio 'bandwidth'. Data pulses have electromagnetic components that cover a large frequency range. The faster the data rate (Baud rate), the higher the frequencies available. Signals can be produced that are between the audio frequencies we can hear, and

radio signals of frequency up to, and sometimes beyond, the broadcast FM band. This accounts for the buzz you may have heard on your radio when it is close to your computer. The modem tones are normally those complying to the international CCITT standards.

A modem converts serial data inputted to it from the computer (hence the need to use the Spectrum RS-232 interface), to two different audio frequencies, one representing 'mark', the other representing 'space'. These tones can then be transmitted down the telephone lines to the other end where another modem converts the tones back to the data signals that the receiving computer understands.

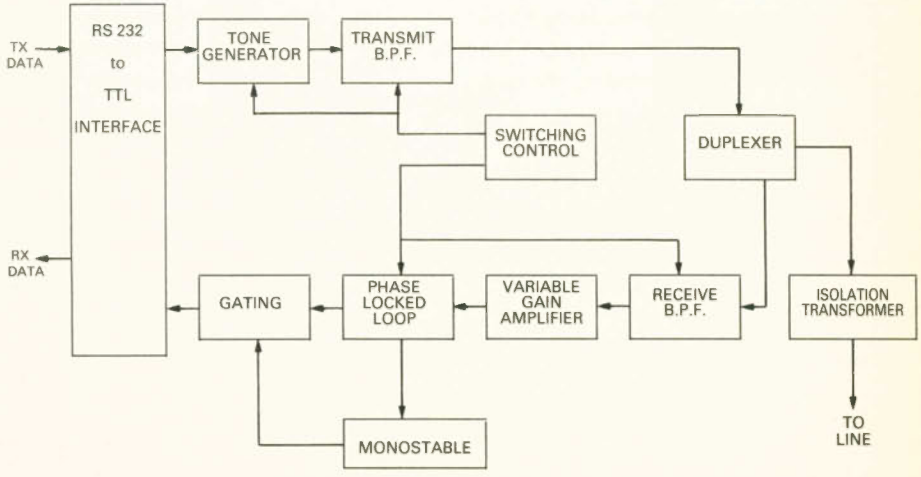
So that data can be sent in both directions, four different frequencies are used, two for each direction. In order that two modems can communicate, one must be switched to the 'originate mode', which transmits 980 Hz (cycles per second), and the other must be switched to the 'answer mode' and transmit at 1650 and 1850 Hz. Each modem receives the alternative pair of frequencies to those which it transmits.

The lower frequency is the mark condition in each case and it is usual for the terminal that makes the call to be switched to the originate mode.

To prevent interference between the two directions of communication, filters are used to pass only the required tones. Unfortunately although the use of these filters helps to remove 'noise' and other forms of interference from the signals, it also has the effect of delaying the data, by a different period for each frequency used.

This is known as 'intersymbol or interbit interference' and has the effect of 'jumbling' or 'smearing' the data signal which in turn damages the program information.

The overall performance of the modem therefore depends on the good design of these filters, and the correct use of Baud rates.



There are two methods of connecting a modem to the telephone system, *acoustic couplers* or *direct electrical connection*.

Acoustic coupling has the advantage of being electrically isolated and easy to connect. However, there is the problem of the coupler picking up local room noise, and the transmit tones will be heard in the telephone receiver considerably louder than the tones that you are trying to receive. The above block diagram shows the more satisfactory arrangement where the connection is made via a British Telecom approved transformer.

A modem can normally be used with any computer having an RS-232 Interface. Conversely it should be possible to use the Spectrum with its RS-232 Interface 1 with practically any modem.

There are various modems available on the open market, some with the added sophistication of a facility for automatically dialling the telephone number. But at the lower end of the price range, simple units, such as that produced by the Maplin organisation, can be obtained complete and tested at around £50 and are also available at lower cost in kit form.

The various operating techniques and command requirements will normally be supplied with the modem and are outside the general scope of this section.

The following tip and free viewdata telephone numbers may come in useful. When calling a British Telecom modem with automatic answer-in, the call will be answered after a few rings and then a 1650 Hz tone will be sent. Your modem should be switched to originate and a mark sent back within 10 seconds or else the modem at the far end will clear down and you will have to call again. Normally when your mark is received, a signing-on message is sent giving instructions on using the system.

Some free public Viewdata systems

| | | |
|---|---|---|
| ESSEX | C-View (Rochford District Council) | 0702 546373 |
| LONDON | CommunITel (Notting Dale ITeC) | 01–960 0327 |
| | Borough of Hackney | 01–985 3322 |
| | Borough of Islington | 01–226 6521 |
| HERTS | North Herts | 04626 77177 |

# 8 The network

According to *Chambers 20th Century Dictionary*, a network is "any structure in the form of a net: a system of lines, eg. railway lines, resembling a net: a system of units, as, buildings, agencies, groups of persons, constituting a widely spread organisation and having a common purpose: a system of stations connected for broadcasting the same program(me), radio, TV" and, in these modern times, computers also.

The choice of the word 'network' to describe the latest development in computer technology is, therefore, by no means accidental. Networks, and more especially Local Area Networks (LANs), are springing up all over the world in response to an increasing consumer demand for computers that communicate directly amongst themselves electronically and without any intervention from operators.

The LAN is developing rapidly within the communications industry as the price of mini-computers and microcomputers falls ( and their power and memory increases) to a point where the desktop computer is commonplace. It is only natural, where there may be several computers within the same office or company, for users to want them all to communicate with each other so that, in the not too distant future, each could become a work-station rather than a mere computer.

As discussed in the two preceding chapters, the RS-232 standard interface can be used to fulfil this communication requirement but it suffers from a few restrictions that the network system has been developed to circumvent. Firstly, by the very nature of a universal standard system, its generality can present some practical problems. Using the RS-232, for example,you have to specify your option from a choice of seven or eight different baud rates, the number of data bits that can be transmitted, choice of the type of parity transmission, and so on: a bewildering number of possible combinations. On the other hand, a networking system is specific to a particular make of computer and so has just one transmission format, making life easier for all concerned because no choice is necessary. Similarly, there is generally software built into the network that permits easy transmission of data, programs and so forth, another feature that is missing from the basic RS-232 system. The third, and most significant feature of a good networking system is its versatility and built-in capacity for straightforward transmission to all the computers on a network or just

to one, specified, terminal. This feature enables the network to be used with confidentiality, tranferring important data only between authorised recipients of the information. In an office environment this is obviously very important; it would not do to allow everybody free access to the company's mainframe!

The development of LANs by office computer manufacturers such as IBM, ICL and DEC, has been rapid over the last two or three years but the home micro sector of the industry has shied away from this new aspect of computing. This is perhaps unsurprising, given the differing requirements between business and the hobbyist. Home computers do, after all, tend to be used individually, not in an environment where local communications would be used to any great advantage, so why put the facility onto the computer? The BBC Micro, with its heavy bias towards education, was one of the first micros to have a dedicated LANs system designed and built for it; the ECONET is ideal for use within a classroom environment. It is, however, a very expensive system, some might even say over-priced for the facilities available.

So, it has been left to Sir Clive, so often the innovator in the home computer section of the industry, to develop and market the first affordable networking system for a home computer. The NET that is available built into Interface 1 allows the Spectrum owner to communicate easily with up to 63 other Spectrums (if they also have Interface 1). Not being the type of person to rest on his laurels for long, Sir Clive is now also producing the world's first home computer with built-in networking facilities, the Sinclair QL.

The Spectrum network system is a very fast method of transferring data between computers. The fastest rate on the RS-232 interface is 19200 bits/sec and this is easily outpaced by the NET, which sends and receives at over 30000 bits/sec, faster even than most disk drives can transfer data! Combine this speed with the ease of use, ensured by the added commands available with Interface 1, and you have in your hands an eminently practical and responsive networking system. But what can you use the NET for?

The first and most obvious application is to transfer programs and/or data between participating computers, an unexiting but essential role for any NET to fulfil. More fun is the playing of 'multi-user' games, games that are played between several competitors, each on their own Spectrum with their own TV. There is an example of this type of game at the end of this chapter; a variation of the old schoolchild pen-and-paper game of Battleships. In this two-player computerized version, each player has on his screen the two relevant grid-maps of his own and his opponent's area of combat. Each move is transmitted to the opposing computer, which then checks for a hit and transmits the result back to the first computer. PAN/PCN Books also publish a listings book containing 25 multi-user

games for the Spectrum with Interface 1 and many of these demonstrate the flexibility and interest that can be created by imaginative use of the NET. One particularly good game in that book is a two computer variation on the old game of 'tele-tennis'. In this version, each player can only see their own half of the court and the ball crosses from one TV to the other and back again as the game progresses!

A more serious and practical application for the NET is the sharing of a single peripheral amongst several Spectrums. One machine, designated the *printer server*, can be connected directly to an expensive, high-quality printer to which other machines can also have access, transmitting their data via the printer server which acts as a buffer. Of course, it is not only a printer that can be accessed in this way; Microdrives, disk drives and any other peripheral equipment can also be connected to one Spectrum and thence accessed by others, with careful programming. It is even possible to connect a modem to one Spectrum, which in turn is connected up to 63 other Spectrums and for the whole network to communicate through the telephone system to 64 similarly-connected Spectrums anywhere in the world.However, the result of all of this may be nothing more than a modern Tower of Babel and a large telephone bill for parents!

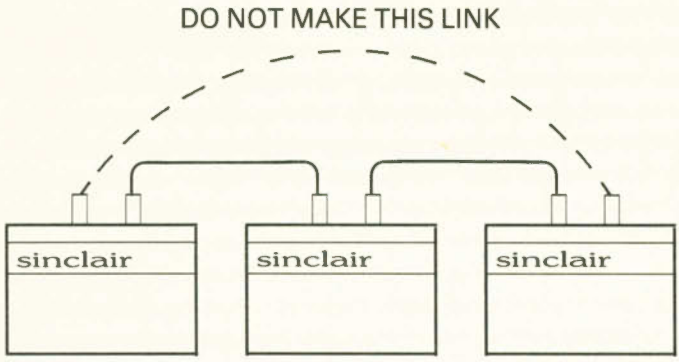
Another function of the NET, and one that is not often considered, is the facility to use the system to produce the equivalent of a multi-tasking computer. Each computer on the NET can be assigned to execute a particular part of a large program, with one machine acting as the 'controller' of the others so that processing can be carried out in a mere fraction of the time that it would take a single computer. I hope you don't mind if I leave the implementation of such a grandiose scheme to you but with 64 Spectrums connected together you will have access to 3 mega-bytes of RAM and 64 micro-processors; in theory at least, this set-up would be more powerful than most mainframe computers in use today and it is a little bit beyond the scope of this book. Before you start writing your mega-program, however, please read the rest of the book!

## Connecting up

When connecting Spectrums together on the NET, there are a few things that have to be noted to ensure smooth operation and good communications. Firstly, you should identify the sockets correctly. These are on the back of Interface 1: they are unmarked but otherwise look exactly the same as the sockets for the cassette recorder on the Spectrum itself. Whilst either or both of the NET sockets can be used, it is important not to try to use the cassette sockets! (They should really have been made a different size so as to avoid any confusion).

The second point to remember is that the Spectrum NET system is a 'line' network and not a 'ring' network. This means that all the partici-

pating computers must be connected in a line and *not* in a complete circle of connections. In practical terms this means that there must always be two computers, one at each end of the line, each having only one wire connected to to it:




Before we go any further, the last point to remember is that you should never turn a Spectrum that is on the NET on or off whilst communications are in progress, because this will cause the system to hang-up. You can have computers connected to the NET which are not turned on and you can, of course, turn a machine on or off when no communication is taking place.

### Setting station numbers

When the NET is connected you are ready to start communicating. As has already been mentioned, one of the features of the NET is the ability to communicate individually with a single computer, as well as to all machines as a group. To enable you to address a computer individually it must, of course, have its own identification or station number. To do this you must first of all decide amongst yourselves the station numbers for each machine on the NET. Up to 64 computers can be connected, so the range of stations is 1 to 64, station 0 being reserved for a special use, as we shall see later in this chapter. Having decided, each participant should type into his computer:

```
FORMAT "n"; ST
```

where ST is the chosen station number. When each machine is first turned on it is designated station 1 automatically, and the current station number is always held in location 23749. An alternative means of changing the station number would be:

```
POKE 23749, ST
```

and there is little to choose between the two methods. When all of the station numbers have been set up, the network is ready for action.

### Sending a program on the NET

Sending a program over the NET to another Spectrum is achieved by using a variant of the SAVE command. In this respect, it is very similar to a transmission over the RS-232 port or, indeed, to SAVEing a program to the cassette recorder, although, the exact syntax of the command varies with its different uses. For the NET, you should use:

```
SAVE "n";ST«ENTER»
```

where ST is the number of the station to which you want the program to be sent. When you enter this line, then press the ENTER key, you will notice that the only thing that happens is that the screen border turns black. Assuming that there is a program already in memory, it will not be sent over the NET until the recipient computer signals that it is ready and waiting to receive. So, the receiver should type:

```
LOAD "n";st«ENTER»
```

where, in this case, st is the station number of the transmitting Spectrum. Now that the receiving station is ready and listening, the program will be transferred and the screen border on both machines will flash in the same way as when LOADing or SAVEing a program from cassette. When the process is complete, the SAVEing can be VERIFYed by using the transmitting computer to SAVE the program again whilst at the same time issuing through the receiving machine the command:

```
VERIFY "n";st«ENTER»
```

At the end of the VERIFYcation a report will be issued by the receiving station: either VERIFY ERROR if something has gone wrong, or the cheery OK if everything proceeded according to plan.

All the usual additions to the SAVE and LOAD commands can be used with the NET. For instance, an array can be transmitted by using:

```
SAVE "n";ST DATA X()
```

where X is the array name. Similarly, you can send a program over the NET that will automatically RUN when received by typing:

```
SAVE "n";ST LINE
```

in almost exactly the same way as you would when using the cassette recorder or Microdrive. Code or screens can be SAVEd and LOADed and programs can be MERGEd with one already residing in memory by using the appropriate additions to the BASIC commands.

### Sending data over the NET

As we have seen, sending and receiving programs over the NET is a simple affair, no more difficult than performing the same functions with a Microdrive. The transmission of *data* over the NET is also similar to transmission using the Microdrive. The first task is to set up a stream along which the data will be able to flow. When you are using the NET, the stream number must be within the range 0-15 and the station number must be in the range 0-64. The stream is OPENed by using:

```
OPEN #s "n";ST
```

where s is the stream number and ST is the station number. This command sets up a 255 byte buffer within the Spectrum. When a NET stream is first OPENed, it can be either an input stream or an output stream but never both at the same time. You determine which it is to be by the first action that is performed to the stream: if you PRINT# to it, it will become a write file and if you INPUT# from it then it will become a read file. Just as with the Microdrive, the CLOSE# statement is vitally important to write files. If you fail to CLOSE a stream properly there is a great danger that not all your data will be sent because some may be left in the buffer!

Data is transmitted over the NET using the command PRINT# and it is received by using either INPUT# or INKEY$#. The syntax and usage of PRINT# is exactly the same as in the normal PRINT statement except, of course, that the output is directed to the NET buffer instead of to the screen. A carriage return (CHR$(13)) will be sent to the NET buffer unless a print modifier character (comma or semi-colon) is used to supress the carriage return. The INPUT# statement will read a series of characters from the NET, ending with a carriage return (CHR$(13)), in just the same way that an INPUT from the keyboard is terminated by pressing the ENTER key (also CHR$(13)). Similarly, INKEY$# will read a single character at a time from the NET and will return a null (CHR$(0)) if there is no character to be received.

It is important, when you are writing your own networking programs, that you carefully match the commands that you use to receive data with the type of data that has been transmitted and with the method of transmission itself. If you wish to transmit the data in the form of variables you should use the PRINT# statement without any modifier so that a

carriage return will be sent as a delimiter between each individual variable. This means that each variable is easily identified and delimited on transmission; the INPUT# statement should then be used to receive the data because it will detect the carriage return at the end of each variable transmitted. The use of INKEY$# in this situation would be inappropriate because this command draws no distinction between characters. If you were to use this command you would have to write your own additional code to detect the delimiting character, which would be inefficient and wasteful when there is already an in-built command that will perform the task for you.

On the other hand, if the data to be transmitted consists of a series of unrelated characters, with no determinate form or structure, then the program that is used to receive such data must be sufficiently flexible to handle all possible eventualities. In the case of this more general type of transmission, it is better to use INKEY$# than INPUT# simply because the former receives a single character rather than a whole string of them. Of course, once each character has been received the program must act upon it and this is entirely up to the program and the programmer.

### Demonstration programs

The pair of listings that follow are a simple demonstration of Networking between two Spectrums. The first is for the computer from which the initial transmission will emanate, whilst the second listing should be typed into the Spectrum that will be the initial receiver. Both programs are substantially the same, so let us look at the first to see how it works.

This program is divided into three modules. The first, lines 10 to 60, initialize the computer to become station number 1 and set the screen and border colours to black with white text. The second and third sections deal with the transmission and reception of data respectively. Line 120 OPENs the NET channel to station 2 and assigns stream 4 for transmission to that channel. In line 130 the keyboard is read. If no key is being pressed at the time then line 140 will keep the program looping back until a key is pressed. When this happens, line 150 will hold up the flow of the program until the key is released. There might appear, at first sight, to be no reason for the inclusion of line 150, but there are two very good reasons for it being there. Firstly, this section of the program is so small that it can be carried out very quickly. Without line 150 the routine would have been executed several times before you had time to take your finger off the key, so that for each single keypress you would be transmitting several characters! The second reason for the inclusion of line 150 is because of an apparent 'bug' in the shadow ROM of Interface 1. Under normal circumstances, as I am sure you are well aware, a program can be stopped by pressing the «CAPS SHIFT» and «BREAK SPACE» keys together. It

appears that, when using the NET for either transmission or reception of data, the «CAPS SHIFT» key is unnecessary and pressing the «SPACE» key by itself is sufficient to stop the program. This is patently an absurd situation to be presented with because it implies that you cannot type a space in your sentences if the resultant data is to be transmitted immediately. Thus line 150 is added to ensure that no key is pressed during transmission and the program will continue to RUN correctly despite the bug.

Line 160 PRINTs the character both to the screen (so that you can see what you are typing) and to the NET channel. Line 170 is the nucleus of this routine because it checks to see whether the character is the delimiting one or not. We have chosen the carriage return, CHR$(13), to denote that we have finished transmitting to station 2 and want to start receiving the answer from it. If the character used is CHR$(13) then the stream will be CLOSEd and the program will jump to line 200, the start of the reception routine. At this point you can add your own lines to detect any other character that you may choose and carry out various functions if you wish. For example, you could add a section that CLOSEs all streams and STOPs the program if a certain key is pressed, so that there is an orderly end to the program,(after informing the other station, of course). The final line, 180, simply sends the program back to read another keypress from the keyboard and repeat the whole process.

The receiving routine is, in many ways, very similar to the one that we have just examined so there is little point in going through it in detail. Stream 5 is used this time instead of stream 4 but this is mostly for clarity rather than being essential. Line 240 looks for a single character on the NET and the program will proceed if one is found. Line 270 again checks for the CHR$(13) character, which denotes the end of transmission from station 2 and CLOSEs the receiving stream if it is found. Replies from station 2 are PRINTed onto the screen of station 1 in yellow to help distinquish them from your own messages (which are in white).

If you next look at the second listing, for station 2, you will see that the differences are few and minor. In fact, all the changes relate to the assignment and use of the NET channel and the reasons for them should be obvious when you have read the rest of this chapter. The second Spectrum is initialized to station 2 rather than station 1, streams are OPENed to channel 1 rather than channel 2 and the program is initially directed to the receiving routine so that station 2 can capture the initial transmission of station 1. Apart from these minor differences the two programs are essentially the same, highlighting the simplicity that is one of the major benefits of a dedicated networking system.

Type the following program into station 1:

```
 10 REM ************************
 20 REM *NETWORK DEMONSTRATION*
 30 REM *       STATION 1       *
 40 REM ************************
 50 PAPER 0: BORDER 0: CLS
 60 FORMAT "N";1
100 REM ************************
105 REM *        TRANSMIT        *
110 REM ************************
120 OPEN #4;"N";2
130 LET A$=INKEY$
140 IF A$="" THEN GO TO 130
150 IF INKEY$<>"" THEN GO TO 15
0
160 PRINT INK 7;A$;: PRINT #4;A
$;
170 IF A$=CHR$ 13 THEN CLOSE #4
: GO TO 200
180 GO TO 130
200 REM ************************
210 REM *        RECEIVE         *
220 REM ************************
230 OPEN #5;"N";2
240 LET B$=INKEY$ #5
250 IF B$="" THEN GO TO 240
260 PRINT INK 6;B$;
270 IF B$=CHR$ 13 THEN CLOSE #5
: GO TO 120
280 GO TO 240
```

This program is for station 2:

```
 10 REM ************************
 20 REM *NETWORK DEMONSTRATION*
 30 REM *       STATION 2       *
 40 REM ************************
 50 PAPER 0: BORDER 0: CLS
 60 FORMAT "N";2
 70 GO TO 200
100 REM ************************
105 REM *        TRANSMIT        *
110 REM ************************
```

```
 120 OPEN #4;"N";1
 130 LET A$=INKEY$
 140 IF A$="" THEN GO TO 130
 150 IF INKEY$<>"" THEN GO TO 15
0
 160 PRINT INK 7;A$;: PRINT #4;A
$;
 170 IF A$=CHR$ 13 THEN CLOSE #4
: GO TO 200
 180 GO TO 130
 200 REM ************************
 210 REM *        RECEIVE        *
 220 REM ************************
 230 OPEN #5;"N";1
 240 LET B$=INKEY$ #5
 250 IF B$="" THEN GO TO 240
 260 PRINT INK 6;B$;
 270 IF B$=CHR$ 13 THEN CLOSE #5
: GO TO 120
 280 GO TO 240
```

### Broadcasting on the NET

As we have already seen, up to 64 Spectrums can be connected to the NET at any one time and each can be individually addressed by means of its station number, which must be within the range 1 to 64. There is a special purpose for the station number 0: it is used when you want to broadcast to all other stations simultaneously. Normal networking is carried out using handshaking. This means, as we have seen, that a transmitting station waits until the receiving station is ready before actually sending the information across the NET. The broadcast channel, designated as station 0, will not wait but will instead send its data immediately onto the whole NET. Any other stations that happen to be listening in at the time will pick up the broadcast, otherwise they will miss out because in this case they do not have to indicate that they are ready first.

The example in the manual that accompanies the Interface 1 describes a superficially good use for the broadcasting capabilities of the NET: one Spectrum is connected to a Microdrive, so that programs can be LOADed quickly, and the program is then broadcast over the NET to all the others. In the classroom situation this could be useful and the quoted example goes on to explain that all pupils should type:

```
LOAD *"n";0
```

so that their stations are ready and waiting for the teacher's broadcast which, incidentally, is achieved by typing:

```
SAVE *"n";0
```

This example, shows to very good effect that handshaking is essential for proper, accurate and reliable communication between computers.

Although the example does not specifically mention it, using the broadcast capabilities of the NET also involves the simultaneous use of an age old handshaking standard known as SPEECH. This form of communication has been developed over many thousands of years and, whilst it was not specifically intended to be used with the Spectrum it does have a very significant role to play in broadcasting.

As with most so-called standards, SPEECH is not, in fact, standardized at all; there are many different forms known as 'languages' which are then sometimes further subdivided into smaller units known as 'dialects'. This is not the place to enter into a detailed discussion about SPEECH and all its ramifications, instead we will concentrate on the version that has been more or less universally adopted in Great Britain and which is known as 'English'. This is a fairly complex version of SPEECH but is essentially no more than an extended form of BASIC, a language which we should all be familiar with by now! Whilst 'English' can be used for broadcasting completely by itself, it is only when it is combined with the NET that the full power of broadcasting is unleashed.

As an example let us develop the idea of the teacher/pupils relationship, so that by combining the NET with SPEECH (remember, a variant of BASIC!), the process will actually work successfully. The programming involved is fairly simple, even for those people who are not too familiar with 'English'. First, the teacher must say something, like:

> TEACHER: "Now, children, I want you all to type into your Spectrums exactly what I have written on the blackboard."

This is often followed by a period of confusion whilst the teacher tries to coerce everybody to do as they are told. After this, the teacher will say:

> TEACHER: "I am going to send you a program over the NET, so is everybody ready for it?"

To ensure that the 'broadcast' is going to be successful the teacher must now poll each of the pupils in turn to find out if they are ready. The correct form of reply to this questioning should, of course, be "YES" but there are several other possibilities, for example "Please sir, can I go to the toilet?" (I leave it up to you to think of some more). Eventually, however,

all the pupils will be ready for the broadcast and it will go ahead, the program will be transferred and the lesson will continue. (Undoubtedly, though, one or more of the computers will 'crash' and the whole, drawn-out process will have to be repeated!)

The point behind this slightly whimsical aside in an otherwise serious book is simply to point out the essential nature of handshaking. The broadcast channel will, in most cases, only work successfully when some form of outside co-ordination is applied to the operation and it is otherwise almost impossible to perform any useful task with this channel. When writing your own programs that use station 0 you could, for instance, poll each of the receiving stations individually to ensure that they are 'listening' before broadcasting simultaneously to them all. Even then you will, no doubt, notice lengthy delays whilst your station is waiting for each of the other stations to acknowledge that it is listening! On the whole, you will usually find a much better method than using station 0 for your communications.

### Networking with the QL

According to the manual and the initial advertising of the Sinclair QL it is possible to communicate between a QL and a Spectrum using the network connections of the two machines, with Interface 1 of course. It is indeed possible to connect the two machines together since they both have the same electrical and electronic standards for these ports. To get them actually to communicate, however, is an entirely different matter and has so far eluded the author and all his many associates within the computer sub-culture of London. As far as can be deduced at the time of going to press with this book, it is not actually possible to network between the Spectrum and QL computers (although it is believed that 'final' versions of the QL will include the facility so to do). This is a very unfortunate state of affairs because it would be a distinct advantage to be able to do so; if only to transfer data and programs from one to the other. For many people the natural progression will, no doubt, be from a Spectrum as their first computer onto the QL, as they become more proficient and familiar with the concept and practice of computing and wish to progress to a more powerful and versatile machine. Changing from one computer to another can often be a traumatic experience in the life of a hacker because all one's previous work, gleaned from extended periods burning the proverbial midnight oil, is suddenly rendered unusable for the simple and frustrating reason that the new computer is unable to read the cassettes or Microdrives created on the original machine. Frustration often leads to anger when it is realised that none of the data and text files that have been patiently keyed-in over the months can be transferred and they will all have to be entered again!

To the rescue comes the ubiquitous RS-232 interface. As we have already seen, this interface standard can be used to connect any two (or more) computers together and, fortunately, the QL has a built-in RS-232 port. By using this connection, together with the knowledge gleaned from the previous three chapters you should have no difficulty at all in communicating between your Spectrum and its more powerful sibling. You could, for example, quite easily write a pair of programs, one for each machine, that allow you to copy a Spectrum Microdrive cartridge onto a QL cartridge via the RS-232 port. Thus armed, the transference of your Spectrum programs and data should be a simple process, free of tears.

The last program in this chapter, *Battleships*, is a good example of the type of programming that should be brought into action when using the NET. At the same time, the game is fun to play. Type it in carefully and don't forget to save it onto a Microdrive or cassette before you RUN it, just in case something untoward happens and you risk losing the program.

```
  10 REM BATTLESHIPS
  15 PRINT "TYPE IN ON OTHER SPE
CTRUM": PRINT TAB 10;"LOAD*""N""
;1": SAVE *"N";1 LINE 20
  20 REM INITIALISE
  30 DIM A(10,10): LET ERRORFLAG
=0: POKE 23658,10: LET COUNT=0:
LET HITCOUNT=0: LET WIN=0: LET G
0=0: DIM v(100,2): LET sflag=0:
LET MCOUNT=0
  40 FOR G=0 TO 7: READ X: POKE
USR "A"+G,X: NEXT G
  50 FOR G=0 TO 7: READ X: POKE
USR "B"+G,X: NEXT G
  60 FOR G=0 TO 7: READ X: POKE
USR "C"+G,X: NEXT G
  70 FOR G=0 TO 7: READ X: POKE
USR "D"+G,X: NEXT G
  80 FOR G=0 TO 7: READ X: POKE
USR "E"+G,X: NEXT G
  90 FOR G=0 TO 7: READ X: POKE
USR "F"+G,X: NEXT G
 100 FOR G=0 TO 7: READ X: POKE
USR "G"+G,X: NEXT G
 110 FOR G=0 TO 7: READ X: POKE
USR "H"+G,X: NEXT G
 120 FOR G=0 TO 7: READ X: POKE
USR "I"+G,X: NEXT G
```

```
 130 CLS : PRINT TAB 11;"BATTLES
HIPS": PRINT TAB 11;"BY TIM BANK
S": PRINT '"THIS IS A COMPUTER
NETWORK       VERSION OF THE TRAD
ITIONAL       PENCIL AND PAPER GA
ME.           EACH PLAYER HAS ONE
BATTLESHIP,   ONE CRUISER, TWO DE
STROYERS AND  TWO SUBMARINES.
              THE BATTLESHIP OCCU
PIES THREE    HORIZONTAL SQUARES,
 CRUISER AND  DESTROYER TWO SQUAR
ES AND THE    SUBMARINES ONE SQUA
RE EACH.      EACH SCREEN DISPLAY
S TWO 10 X 10GRIDS. THE PLAYER P
LACES HIS OR HER SHIPS ON THE LE
FT HAND GRID,WHILST THE RIGHT HA
ND GRID       RECORDS BOMBS DROPP
ED ON THE     OPPONENT'S SQUARES.

              Press any key to co
ntinue"
 140 PAUSE 0
 150 CLS : PRINT "HITS ARE DISPL
AYED IN RED.       THE WINNER IS
THE FIRST ONE TO   DESTROY ALL TH
E OPPONENT'S       FLEET.


                   Entries of shi
ps or bombs are    accomplished b
y entering the     vertical colum
n letter followed by the horizon
tal line number.

                   Press any key
to start": PAUSE 0
 200 REM SCREEN DISPLAY
 210 CLS : BORDER 1: PAPER 7: IN
K 0: CLS : PRINT TAB 11;"BATTLES
HIPS"
 220 PRINT AT 3,4;"ABCDEFGHIJ
  ABCDEFGHIJ"
```

```
 230 FOR B=1 TO 9: PRINT AT B+3,
3;B; PAPER 1;""; PAPER 7;AT B+3,
18;B; PAPER 1;""; PAPER 7: NEXT
B
 240 PRINT AT 13,2;"10"; PAPER 1
;""; PAPER 7;AT 13,17;"10"; PAPE
R 1;""; PAPER 7
 250 PRINT AT 15,3;"YOUR FLEET";
AT 15,18;"ENEMY FLEET"
 300 REM DISPLAY SHIPS
 310 INPUT "COORDINATES OF BATTL
ESHIP        COLUMN BY ROW ";A$
 320 GO SUB 1000
 330 IF Y>8 OR ERRORFLAG=1 THEN
 GO TO 310
 340 LET A(X,Y)=4: LET A(X,Y+1)=
4: LET A(X,Y+2)=4: PRINT AT X+3,
"
 350 INPUT "COORDINATES OF CRUIS
ER           COLUMN BY ROW ";A$
 360 GO SUB 1000
 365 IF Y>9 THEN GO TO 350
 370 IF A(X,Y)<>0 OR A(X,Y+1)<>0
 OR ERRORFLAG=1 THEN GO TO 350
 380 LET A(X,Y)=3: LET A(X,Y+1)=
3: PRINT AT X+3,Y+3;""
 390 INPUT "COORDINATES OF DESTR
OYER 1       COLUMN BY ROW";A$
 400 GO SUB 1000
 405 IF Y>9 THEN GO TO 390
 410 IF A(X,Y)<>0 OR A(X,Y+1)<>0
 OR ERRRORFLAG=1 THEN GO TO 390
 420 LET A(X,Y)=2: LET A(X,Y+1)=
2: PRINT AT X+3,Y+3;""
 430 INPUT "COORDINATES OF DESTR
OYER 2       COLUMN BY ROW";A$
 440 GO SUB 1000
 445 IF Y>9 THE GO TO 430
 450 IF A(X,Y)<>0 OR A(X,Y+1)<>0
 OR ERRORFLAG=1 THEN GO TO 390
 460 LET A(X,Y)=2: LET A(X,Y+1)=
2: PRINT AT X+3,Y+3;""
 470 INPUT "COORDINATES OF SUBMA
RINE 1       COLUMN BY ROW ";A$
```

```
 480 GO SUB 1000
 490 IF A(X,Y)<>0 OR ERRORFLAG=1
 THEN GO TO 470
 500 LET A(X,Y)=1: PRINT AT X+3,
Y+3;""
 510 INPUT "COORDINATES OF SUBMA
RINE 2       COLUMN BY ROW ";A$
 520 GO SUB 1000
 530 IF A(X,Y)<>0 OR ERRORFLAG=1
 THEN GO TO 470
 540 LET A(X,Y)=1: PRINT AT X+3,
Y+3;""
 600 INPUT "STATION NUMBER (1/2)
 ";ST
 610 FORMAT "N";ST
 620 LET ST1=3-ST: LET GO=ST
 630 IF GO=1 THEN  GO SUB 800: G
O TO 650
 640 IF GO=2 THEN  GO SUB 900: G
O TO 650
 650 IF WIN=0 THEN  LET GO=3-GO:
 GO TO 630
 660 IF WIN=1 THEN  PRINT #1;"CO
NGRATULATIONS YOU HAVE WON !": P
AUSE 0: INPUT "ANOTHER GAME (Y/N
)";S$: IF S$="Y" THEN  RUN 30
 670 PRINT #1;"     UNFORTUNATEL
Y YOU LOST !": PAUSE 0: INPUT "A
NOTHER GAME (Y/N) ";S$: IF S$="Y
" THEN  RUN 30
 800 LET sflag=0: INPUT "POSITIO
N TO BOMB ";A$: GO SUB 1000
 810 IF ERRORFLAG=1 THEN  GO TO
800
 814 FOR d=1 TO 100: IF x=v(1,1)
AND x=v(1,2) THEN  LET sflag=1
 815 NEXT d: IF sflag=1 THEN  GO
TO 800
 816 LET mcount=mcount+1: LET v(
mcount,1)=x LET v(mcount,2)=y
 820 OPEN #4;"N"+ST1: PRINT #4;X
: PRINT #4;Y: CLOSE #4: OPEN #4;
"N";ST1: INPUT #4;HIT: INPUT #4;
SHIP: CLOSE #4
```

```
 830 IF HIT=0 THEN  PRINT AT X+3
,Y+18;"X": RETURN
 840 IF SHIP=2 THEN  PRINT AT X+
3,Y+18; INK 2;"D"; INK 0
 845 IF SHIP=3 THEN  PRINT AT X+
3,Y+18; INK 2;"C"; INK 0
 850 IF SHIP=4 THEN  PRINT AT X+
3,Y+18; INK 2;"B"; INK 0
 860 IF SHIP=1 THEN  PRINT AT X+
3,Y+18; INK 2;"S"; INK 0
 870 LET COUNT=COUNT+1: IF COUNT
=11 THEN  LET WIN=1
 880 RETURN
 900 OPEN #4;"N";ST1: INPUT #4;X
: INPUT #4;Y: CLOSE #4
 910 IF A(X,Y)=0 THEN  LET HIT=0
: LET SHIP=0: PRINT AT X+3,Y+3;"
X": GO TO 940
 920 LET HIT=1: LET SHIP=A(X,Y):
 LET HITCOUNT=HITCOUNT+1: PRINT
AT X+3,Y+3; INK 2;"*"; INK 0: IF
 HITCOUNT=11 THEN  LET WIN=2
 940 OPEN #4;"N";ST1: PRINT #4;H
IT: PRINT #4;SHIP: CLOSE #4: RET
URN
1000 REM ERROR CHECK
1010 LET ERRORFLAG=0: IF CODE A$
(1)<64 OR CODE A$(1)>74 OR CODE
A$(2)<49 OR CODE A$(2)>57 THEN
LET ERRORFLAG=1: RETURN
1020 IF LEN A$>2 THEN  IF CODE A
$(3)<>48 THEN  LET ERRORFLAG=1:
RETURN
1030 IF LEN A$=3 THEN  LET X=10:
 GO TO 1050
1040 LET X=VAL A$(2)
1060 LET Y=(CODE A$(1))-64: RETU
RN
9000 DATA 255,129,129,129,129,12
9,129,255
9010 DATA 0,0,48,48,48,255,255,0
9020 DATA 0,0,7,7,7,127,255,0
9030 DATA 0,0,128,132,136,255,25
4,0
```

```
9040 DATA 0,0,128,132,136,255,25
4,0
9050 DATA 0,0,3,35,19,255,254,0
9060 DATA 0,0,0,30,24,255,254,0
9070 DATA 0,62,62,62,62,255,255,
0
9080 DATA 0,0,32,20,28,255,254,0
```

# 9 Machine code

**What is machine code?**

Most owners of the Sinclair Spectrum teach themselves BASIC so that they can write programs, but when it comes to machine code many avoid taking the plunge. Some of you may never have seen an accurate definition of the term 'machine code' so here is one to think about. Machine code is a low level language that the Central Processing Unit of a computer can understand directly. In the case of the Spectrum the Central Processing Unit (or CPU) is Z80 machine code. A low level language is one that the processor understands fairly easily and a high level language is one that humans can understand more readily. Some examples of high level languages are BASIC, Pascal, LISP, COBOL and Logo.

It may seem confusing that the Spectrum uses a Z80A microprocessor, yet Z80 chip machine code. The reason is simple, the Z80 chip has been improved a number of times since being introduced; there are now at least four versions, each being able to work faster than the last. These are: Z80, Z80A, Z80B and Z80C. All these versions of the Z80 use the same machine code.

It is beyond the scope of this book to teach machine code in depth and there are many good books already covering the subject adequately. However, this introduction is intended to give enough information for you to understand the concept of machine code programming sufficiently for you to be able to use the built-in routines and programs within the Spectrum.

**What are the advantages of machine code?**

*1 Speed*
Machine code is much faster than BASIC, about 20 to 200 times faster depending on the operation being performed!

*2 Flexibility*
It is possible to write programs that change the way BASIC works, add commands to BASIC or write fast animation routines, for example.

*3 Portability*
A Z80 program can be written on one machine and run on another, try

doing that with BASIC! This is the way that a lot of software houses write Spectrum machine code games. Two of the most popular systems for development are the TRS-80 Models 1 and 3, both of which have a Z80 chip and can support up to four disk drives.

*4 Compactness*
In many cases it is possible to produce Z80 programs that are much smaller than the equivalent BASIC program.

*5 Image*
It is great to be able to say to your friends that you program in machine code!

### What are the disadvantages of machine code?

*1 Input*
On most home computers there is no simple way of inputting machine code other than POKEing numbers into memory locations. For entering machine code on your Spectrum a monitor or assembler program would be very useful. These can be bought quite easily in tape format or you could buy a book, such as *Cracking the Code on the Sinclair ZX Spectrum*, also published by PAN/PCN, containing a monitor program that can be typed in.

*2 Error trapping*
There are no helpful error messages in Z80 machine code as there are in BASIC, so if something goes wrong the machine will probably just crash or lock up, leaving you to work out the cause unaided.

### The Z80 number representation system

At this point we should explain the method by which the Z80 stores and uses numbers.

The Z80 is known as an 8-bit processor. 8-bit means that it will work with 8 digit binary numbers (bytes). *Binary notation* is a method of representing numbers using only 0's and 1's as units and the reason that computers use binary is that it is much simpler for an electronic device to deal with. Within the machine, 0's and 1's represent the condition of semiconductor switches (0=off, 1=on). In the same way that the decimal system works to powers of 10, the binary system uses powers of 2. The following examples should illustrate how the two systems work.

185 decimal breaks down into:

```
5 *   1 =   5
8 *  10 =  80
1 * 100 = 100
          ---
          185
```

110 binary gives:

```
0 *   1 =   0
1 *   2 =   2
1 *   4 =   4
          ---
            6
```

The numbers above on the righthand side of the multiplication signs are the values of each digit moving from right to left. In decimal the next digit is worth 10 times the last, in binary it is just double. The maximum value that can be held in 8 binary digits therefore is:

11111111 binary which is equal to:

```
1 *   1 =   1
1 *   2 =   2
1 *   4 =   4
1 *   8 =   8
1 *  16 =  16
1 *  32 =  32
1 *  64 =  64
1 * 128 = 128
```

The maximum value that the Z80 can deal with directly would appear therefore to be 255. The Spectrum RAM is arranged so that it can store 8-bit values as well; try POKEing 16384 (top left corner of the screen) with any value outside the range 0 to 255 and you will get an error message. The screen will display the binary value as pixels: an individual pixel will be set if the corresponding bit is 1 or reset if the bit is 0.

In fact, the Z80 can actually deal with values up to 65535 because there are some 16-bit (2 byte) instructions built into it. BASIC can deal with even bigger numbers than this, because the floating point numbers used by the Spectrum are stored in five bytes. For more detail on this look at the end of Chapter 24 in your Spectrum manual.

Since binary numbers of up to 16 digits are long and difficult to remember we can use another number representation system called *hexa-*

*decimal*. This uses base 16 in the same way that decimal notation uses base 10 and binary notation uses base 2. Hexadecimal digits are 0 to 9 and A to F to represent the values 0 to 15 thus:

| Hexadecimal | Decimal |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | 10 |
| B | 11 |
| C | 12 |
| D | 13 |
| E | 14 |
| F | 15 |

Therefore AF hexadecimal is equal to:

```
15 * 1 =  15
10 * 16 = 160
          ---
          175 decimal
```

You might be wondering at this juncture, what is the point of using any number system other than decimal? The reason for using hexadecimal is that it represents more closely the bit pattern in an 8 or 16 bit binary number, and when you are using machine code this can be very important. Below is a 16 bit binary number split into blocks of four bits (or *nybbles* as they are called) to show how hexadecimal numbers relate to binary.

```
1010  1100  0010  1110  Binary       1010110000101110
  A     C     2     E   Hexadecimal  AC2E
 10    12     2    14   Decimal      44078
```

As you can see, each hexadecimal digit represents four binary digits but there is no such relationship between a decimal number and a binary number.

To allow negative numbers to be represented in binary a system called *two's complement* is used. Using two's complement an 8-bit number will consist of one bit for the sign (0=positive, 1=negative) and seven bits for the actual value. A 16-bit number will have one bit for the sign followed by fifteen bits for the value. Using the binary example above the new value of 1010110000101110 is minus 11310. The value is negative because the leftmost bit is set and the value is 11310 because that is the result of converting the other 15 digits to decimal.

In two's complement 8-bit values will be in the range 127 to −128 and 16-bit values will be 32767 to −32768.

**What are the main differences between machine code and BASIC?**

*1 Speed*

As mentioned earlier, Z80 machine code is understood directly by the Z80 microprocessor, while BASIC and other high level languages have to be either converted to machine code or interpreted when RUN. Spectrum BASIC is interpreted when RUN, and this is the main reason why it is so much slower than machine code. In addition to having to look up tokens in tables held in the ROM the BASIC interpreter also has to check the syntax of the current command, carry out the command, and find the next statement or line number. All these functions take a lot of time and can be eliminated by using machine code. It is possible to use a compiler to speed up the execution of a BASIC program by doing most of this work and creating a machine code program. Unfortunately, most compilers produce very inefficient machine code and only succeed in speeding up BASIC by a factor of about 10 to 100.

*2 Use of variables*

In Z80 machine code there are no string variables, numeric arrays or string arrays. Numbers can only be used directly if they are within the range 32767 to −32768. Memory locations must be set aside for storing data that would have been held in BASIC variables and more care is needed when dealing with data in machine code.

*3 Methods of calculation*

As mentioned earlier, the Z80 operates only in numbers, but that actually doesn't help much when we wish to do calculations in machine code. There are two main problems: the first is that the size of the largest number that the Z80 can work on at any one time is 16-bits (2 bytes) with a value of 32767 to −32768. So if you want to use any numbers outside this range a specific machine code routine will be required to break the numbers down into manipulable parts and store results separately. The second problem is that the Z80 can only add or subtract numbers. If multiplication or division are needed then, again, specific routines will be

required. Fortunately, the BASIC floating point point calculator is available to the Z80 programmer via ROM calls. The Spectrum floating point calculator has a lot of useful arithmetic and mathematical routines available including multiplication, division, SIN, COS and TAN. Unfortunately, there is not enough time here to go into details of how to use this calculator since that would take most of a chapter in itself.

**Storage – Where should a machine code program be stored?**

This is a problem that doesn't arise with BASIC programs since the BASIC editor controls where the BASIC program is to be stored. There are a number of locations where machine code can be stored and we give them here together with the advantages and disadvantages associated with each.

*1) Above RAMTOP*
The code starts at the first address after the number used in the last CLEAR command. There are very few problems involved here, since the main purpose of the CLEAR command is to provide room for machine code. The only warning you need is to watch out for the User Defined Graphics. The address of the start of the UDGs is held in the system variable UDG. As long as your code does not overwrite this address or any one after it there will be no problem.

*2) In the printer buffer*
If your machine code is only 256 bytes or less long then you might decide to store it in the printer buffer (23296 to 23551). However, if the printer is used or any of the commands CLEAR, NEW or RUN employed, the printer buffer will be cleared or corrupted and your machine code will be lost!

*3) In front of a BASIC program*
It is possible to move the BASIC pointers so that there is room between the system variables and the start of BASIC for machine code. This method is quite awkward to implement, however, especially when extra streams and channels are being used when the Interface 1 is attached and Microdrives may be used. Also, some of the Spectrum floppy disk interfaces and other mass storage devices use this area. Therefore, this area is *not* recommended for storing machine code unless you are sure you know what you are doing.

*4) In REM statements*
This was a popular place for machine code before Microdrives were commonly available since it reduces loading time by loading BASIC and machine code in one operation. Usually only the first line is used since it is easier to find its address. Now that Microdrives are in fairly common use this procedure has become less popular. To use this method it is recommended that the programmer write position-independent code. This is

machine code that will run correctly wherever it is placed in memory (usually machine code is fixed to run only from a particular address).

*5) In BASIC variables*
It is possible to LOAD a string array which contains a machine code program. This should also be position-independent code. There are no great advantages in using this method and it is not widely used.

*6) On screen*
By using the same colour for INK and PAPER you could have machine code stored temporarily on screen without anybody realising! The only real advantage of this is that you could write a program that calls your screen-based routine as a security check. If someone were to pirate your program and try to speed up the loading by removing the SCREEN$ and changing the BASIC loader to suit, they would have a non-working program at the end. You could even note that they were ripping you off and give them a suitably nasty message, then go into a never-ending loop similar to '10 GOTO 10' but in machine code!

**What are monitors and assemblers?**

Both of these are programs designed to assist in the writing and debugging of machine code programs.

A *monitor* is a program designed for debugging existing machine code rather than for writing it. Fairly standard features of monitors include:

*Hex dump* – display or print the values held in a section of memory. The values will be displayed in hexadecimal.

*Edit address* – change the contents of a memory address.

*Disassemble address* – convert the values found in memory, starting at the specific address, to assembler mnemonics. Assembler mnemonics are the machine code instructions as used in assembly language. This is dealt with in more detail when the assembler program is explained.

*Move address1, address2, length* – move the contents of a block of memory from one place to another. The move will be 'length' bytes from address1 to address2.

*Find b1, b2, b3 . . . bn* – search through memory for a specified sequence of bytes.

*Jump address* – execute the machine code program in memory, starting at the address given.

*Save/Load/Verify tape and/or Microdrive storage.*

*Reg* – display the contents of the Z80 registers and allow them to be changed. Registers will also be explained later.

An *assembler* is used to write the machine code in the first place. Assembly language is a more user–friendly way of writing machine code than having to remember a whole sequence of hexadecimal numbers and their functions. There are nearly 700 instructions in the Z80 processor, so remembering them in hexadecimal would be somewhat tedious! Every Z80 instruction has a mnemonic associated with it. These mnemonics are standard and were decided upon by Zilog who originally designed the Z80 microprocessor (that's where the 'Z' comes from). Below is a short list of some mnemonics with their associated hexadecimal values and functions. Where we have used XX this represents data bytes that may change depending on the specific use of the instruction.

| Mnemonic | Hex | Function |
|---|---|---|
| CALL ADDR | CD XX XX | Equivalent to GOSUB |
| JP ADDR | C3 XX XX | Equivalent to GO TO |
| LD A, XX | 3E XX | Equivalent to LET A=XX |
| LD (ADDR),A | 32 XX XX | Equivalent to POKE ADDR,A |
| LD A, (ADDR) | 3A XX XX | Equivalent to LET A=PEEK (ADDR) |

By using an assembler the programmer is relieved of the need to memorize 700 hex codes because the assembler does all the necessary conversion from assembly language to machine code. Assembly language is therefore a slightly higher level language than machine code. The programmer simply types in a list of assembly language instructions to create an assembly language program, then instead of RUNning it he or she must assemble it and save the resultant machine code. It is always advisable to save the assembly language list (usually called the *source code*) as well as the machine code (sometimes called the *object code*) when dealing with assemblers before running the machine code, just in case a crash occurs.

### What is the structure of the Z80 and its registers?

It has already been noted that the Z80 can store values in memory. In addition, it also has some internal storage. This is usually of a temporary nature and is for data that is currently being worked upon. It exists in the form of registers as shown in this diagram of the internal structure of the Z80 chip.

Bank 0 Bank 1

| A | F |
|---|---|
| B | C |
| D | D |
| H | L |

| A' | F' |
|---|---|
| B' | C' |
| D' | D' |
| H' | L' |

| IX |
|---|
| IY |
| PC |
| SP |

The registers are all referenced by a letter name. Those that appear on the same line as each other can be used as register pairs and hold 16-bit values. All single registers are 8-bit wide. There are special functions for some of these registers. The A register is known as the Accumulator and is used to receive nearly all 8-bit arithmetic results. The HL register pair works as a single 16-bit register for 16-bit arithmetic results, PC is the Program Counter and keeps track of the location of the next instruction for the processor to execute. This is very similar to the contents of the system variables NEWPPC and NSPPC when executing a BASIC program. IX and IY are used as pointers; the IY register is used by the ROM and must not have its contents changed if any ROM routine is going to be called. The other registers are mostly general purpose storage areas.

*The memory maps*

The memory map overleaf shows where the ROM resides and what follows it.

**Memory map of the Spectrum without the Interface 1 connected**

| DECIMAL | | HEXADECIMAL |
|---|---|---|
| 0 | | #0 |
| | SPECTRUM | |
| | ROM | |
| 16384 | | #4000 |
| | RAM | |
| 32768 | | #8000 |
| | RAM IN 48K SPECTRUM | |
| | NOTHING HERE IN 16K SPECTRUM | |
| 65535 | | #FFFF |

**Memory map after Interface 1 has been added**

| DECIMAL | | | HEXADECIMAL |
|---|---|---|---|
| 0 | | | 1/2 |
| | 1/2 | | |
| | SPECTRUM | INTERFACE 1 | |
| | ROM | ROM | |
| 8191 | | | #IFFF |
| | 1/2 | | |
| | SPECTRUM | | |
| | ROM | | |
| 16384 | | | #4000 |

16384 _____ #4000

RAM

32768 _____ #8000

RAM IN 48K SPECTRUM

NOTHING HERE IN
16K SPECTRUM

65535 _____ #FFFF

The RAM is not all available for the user, and the diagram below shows how it is broken up.

DECIMAL HEXADECIMAL

16384 _____ #4000

DISPLAY FILE

22528 _____ #5800

ATTRIBUTES

23296 _____ #5B00

PRINTER BUFFER

23552 _____ #5C00

SYSTEM VARIABLES

23734 _____ #5CB6

MICRODRIVE MAPS
NOT ON SPECTRUM
WITHOUT INTERFACE 1

_____

| | |
|---|---|
| CHANS | CHANNEL INFORMATION |
| | ENDS WITH #80 |
| PROG | BASIC PROGRAM |
| VARS | PROGRAM VARIABLES |
| | ENDS WITH #80 |
| E-LINE | COMMAND OR LINE BEING EDITED |
| | ENDS WITH A NEW LINE THEN #80 |
| WORKSP | INPUT DATA |
| | END WITH NL |
| | TEMP WORK SPACE |
| STKBOT | CALCULATOR STACK |
| STKEND | SPARE |
| | MACHINE STACK |

—

GOSUB STACK

—

RAM TOP #3E

—

UDG USER DEFINED GRAPHICS

—

P-RAMT END OF MEMORY.

—

All the pointers after 23734 will move around in memory depending on the number of variables, whether Microdrives are attached, how many streams are open, and so on.

The Spectrum ROM is designed to make life easier for the programmer by supplying access to a reasonable BASIC interpreter and editor. There are three main functions that the ROM controls, they are:

i) program editing
ii) program execution
iii) communication with external devices (tape, keyboard, speaker and TV)

The BASIC is interpreted in a similar manner to the following list.

```
RUN     GOTO NEXT
NEXT    READ THE NEXT COMMAND
        IF NO COMMAND IS FOUND THEN GOTO END
        FIND THE COMMAND IN THE COMMAND TABLE
        IF NOT FOUND THEN GOTO ERROR
        CALL ROM ROUTINE FOR THE COMMAND
        FIND THE NEXT COMMAND
        GOTO NEXT
ERROR   DISPLAY ERROR MESSAGE
        GOTO END2
END     DISPLAY OK MESSAGE
END2    STOP
```

Since there is 16K of machine code sitting in the ROM waiting to be called by the BASIC system, we might as well make use of it from machine

code. We can try a simple ROM call by POKEing in four bytes, then calling them. Here are the assembly language and machine code listings.

| Hex | Mnemonic | Decimal |
|---|---|---|
| CD 6B 0D | CALL CLS | 205 107 13 |
| C9 | RET | 201 |

To test this try the following program.

```
 10 FOR I=1 TO 100
 20 POKE 23677,127
 30 POKE 23678,88
 40 LET X=INT(RND*250)-125
 50 LET Y=INT(RND*160)-80
 60 DRAW X,Y
 70 NEXT I
 80 POKE 23296,205
 90 POKE 23297,107
100 POKE 23298,13
110 POKE 23299,201
120 PRINT "PRESS A KEY TO CALL
MACHINE CODE"
130 PAUSE 1: PAUSE 0
140 LET Z=USR (23296)
```

This routine calls the CLS routine which starts at 0D6B hexadecimal in the Spectrum ROM. The USR function is really a GOSUB to machine code and the machine code RET is the equivalent of RETURN. The CALL is the Z80 GOSUB so the program above calls a subroutine that calls the CLS subroutine.

### Adding to Interface 1

When you add the Interface 1 you get the use of an extra 8K of ROM, known as the 'shadow' ROM because it uses the same addresses as the lowest 8K of the normal ROM. The shadow ROM contains routines to control the Microdrives, RS-232 and networking, as well as a facility for extending the BASIC interpreter to add new commands easily. The memory map will change when the Interface 1 is connected, as we have seen.

It is possible, though awkward, to call shadow ROM routines using the CALL instruction because the shadow ROM must be switched in, or *paged* in before it can be called. To make life easier for us poor machine code programmers Sinclair has included some hook codes. By careful use of these hook codes it is possible for the programmer to delve into the shadow ROM at will and make use of the routines to be found there. The hook codes are dealt with in some detail in Chapter 11.

**How does the Spectrum decide which ROM to use?**

Normally the Spectrum ROM is paged in, but when you try to enter a command on the Spectrum that the BASIC interpreter does not understand then an error condition is generated and the shadow ROM is paged in to handle the error. If the error is detected during line editing, that line will be shown with a flashing question mark. If the error occurs during a program RUN then an error message will be displayed.

All Spectrum error handling is controlled via RST 8. This means that CALLing address 0008 hex in the Spectrum ROM will generate an error. To show that this is so, try the line below.

```
POKE 23296, 207: POKE 23297,26: RAMDOMIZE USR 23296
```

The two bytes above are the machine code for RST 8 followed by byte 26 for an error code of 26. In assembly language it is:

```
RST 8
DEFB 26
```

RST 8 is just a very quick way of CALLing address 0008 hex and DEFB is the assembly language method of storing a byte of data. RST 8 is set up in the Spectrum so that it will use the following byte for an error code or hook code depending on the value of the byte. Using the example above try some values instead of 26 and you will find that you get a variety of error messages.

You should find that values of 255 (−1) to 26 give various error messages and values from 51 to 254 will give *Hook code error, 0*:1. All other values are valid hook codes and will be explained in the next chapter.

# 10 Hook codes

### What is a hook code?

A hook code is a byte that will, when placed after an RST 8 instruction, cause a shadow ROM call to be executed. The advantage of hook codes is that by using them the Microdrive commands can be directly accessed from machine code. It then becomes possible to call such functions as READ SECTOR, SAVE SECTOR, SEND RS-232 and RECEIVE NETWORK without having to write enormous machine code programs. The hook code 27, for example, is a 'wait for any key' subroutine. To demonstrate its use type in the following line:

```
POKE 23296,207:POKE 23297,27:POKE 23298,201:RANDOMIZE
USR 23296
```

The machine should be stopped at the moment, now press a key. It's as easy as that to call all of the hook codes, although most of them do have more complex functions.

The assembly language for the line above is:

```
RST 8
DEFB 27
RET
```

From here on in this book there will be several more assembly language listings and it might be advisable for you to buy an assembler if you want to enter some of the longer listings, for example, Devpac from Hisoft.

### General hook codes

#### CODE #1B – INKEY

*Keyboard input*

As previously mentioned, this hook code will wait for a key to be pressed on the Spectrum keyboard. Once this has been done, the code for the character which has been depressed is put into the A register. The A register can be thought of as a memory location actually inside the Z80, once you have the code of the character in the A register you can manipulate it in any way you wish – be it printing it out to the screen or simply

checking that a particular key has been pressed – by comparing it with another value. Calling the routine couldn't be simpler, all that is required is for the hook code of the routine (#1B) to be included after an RST#08 instruction.

*How to call INKEY*

RST #08
DEFB #1B

*Entry conditions*

N/A

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

The A register will hold the code of the last key pressed.

The example below shows a simple way of using this INKEY routine. The keyboard is checked for a key press and the relevant character is printed out using a routine in the Spectrum ROM which prints out to the current stream. In the program below we set this up to be stream number 2 which is the upper screen.

```
10        ORG  30000    ;star
   t of machine code at 30000
    decimal
20 CHOPEN EQU  #1601    ;rout
   ine in spectrum ROM
30 ;that sets up the current
   stream to the contents of
   the A register
40 KEYIN  EQU  #1B      ;hook
    code
50 START  LD   A,2      ;star
   t of actual machine code.
   Load the A 2
60        CALL CHOPEN   ;set
   up stream to upper screen,
    ie 2
70 LOOP   RST  #08      ;call
    up interface 1
80        DEFB KEYIN    ;KEYI
   N hook code is stored here

90        JP   LOOP     ;jump
    back to LOOP
```

There is no way out of the above routine and you will have to turn the power to your Spectrum off before you can do anything else.

**CODE #1C – PRINT2**

This hook code will output the character that is held in the A register using stream number 2, which is usually the upper screen. By using this command you do not have to select the output stream yourself as in the example above.

*How to call PRINT2*

Again this is an extremely simple exercise:

```
RST #08
DEFB #1C
```

*Entry conditions*

The A register should contain the code of the character to be printed.
The interrupts should be on.

*Registers used by routine*

AF,BC,DE,HL,AF',BC',DE'

*Exit conditions*

N/A

Let's print out some characters using the hook code above:

```
10         ORG  30000
20 PRINT2  EQU  #1C
30 START   LD   A,32      ;CODE
    FOR SPACE
40 LOOP    PUSH AF        ;STOR
   E THE A REGISTER
50         RST  #08       ;CALL
    THE PRINT
60         DEFB PRINT2    ;ROUT
   INE
70         POP  AF        ;REST
   ORE THE A REGISTER
80         INC  A         ;INCR
   EASE THE CHARACTER CODE BY
    1
90         CP   "["       ;IS I
   T GREATER THAN Z Ø
100        JP   NZ,LOOP   ;IF N
   OT THEN GO ROUND AGAIN,IE
   JUMP TO LOOP
110        RET  ;BACK TO CALLI
   NG ROUTINE
```

The program above will print out all of the characters with codes from space to Z.

The program that was used to demonstrate the INKEY hook code could be written using this hook code:

```
10         ORG   #30000
20 KEYIN   EQU   #1B
30 PRINT2  EQU   #1C
40 LOOP    RST   #08        ;CALL
    KEYIN
50         DEFB  KEYIN
60         RST   #08        ;CALL
    PRINT2
70         DEFB  PRINT2
80         JR    LOOP       ;JUMP
    RELATIVE TO LOOP
90 ;JR IS AN ALTERNATE FORM O
   F JP AND IS 1 BYTE SHORTER
```

**CODE #1F – PRINT3**

This hook code is almost identical to the one above except that it outputs the character in the A register via stream 3, usually the ZX Printer, rather than stream 2.

*How to call PRINT3*

As with the previous commands, this routine is called by the RST #08 instruction:

```
RST #08
DEFB #1F
```

*Entry conditions*

A register should hold the code of the character that is required to be printed.

Interrupts should be switched on.

*Registers used by routine*

AF,BC,DE,HL,AF',BC',DE'

*Exit conditions*

N/A

**CODE #20 – KEYTEST**

This hook code causes the keyboard to be examined and will set the zero flag to 0 if any key is being pressed.

*How to call KEYTEST*

RST #08
DEFB #20

*Entry conditions*

N/A

*Registers used by routine*

AF,BC,DE,HL

*Exit Conditions*

The zero flag will be reset (NZ) if a key was pressed.
If no key was pressed then the zero flag will be set (Z).

The following routine waits for a keypress before returning to BASIC:

```
10          ORG   30000
20 KEYT     EQU   #20
30 LOOP     RST   #08        ;CALL
    THE ROUTINE
40          DEFB KEYT        ;KEYT

50          JR    Z,LOOP     ;IF N
   O KEY PRESSED THEN GO BACK
    TO LOOP
60          RET
```

**CODE #31 – CRVARS**

This hook code will create all the system variables associated with the Interface 1. The first reference to the Microdrive will automatically set up the system variables and they will stay there until a NEW is performed. Therefore this hook code only needs to be used if you intend to load a program from tape which will use the Interface 1 before you make any reference to the interface.

Since the routine will only create the system variables if they don't already exist, it is probably a good idea to include a call to this hook code before using any machine code program that will access the Interface 1.

*How to call CRVARS*

RST #08
DEFB #31

*Entry conditions*

N/A

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

N/A

**CODE #32 – CALI1**

Sinclair claim that this routine is for their own future expansion. It allows you to call any routine inside the Interface 1 by using its address. The routine is called in the same way as the previous hook codes and the system variable HD–11, which is found at location #5CED, should hold the address of the routine to be called in the interface ROM.

*How to call CALI1*

RST #08
DEFB #32

*Entry conditions*

HS–11 (location #5CED) should contain the addresss of the routine to be called.

*Registers used by routine*

This depends upon the routine which you call.

*Exit conditions*

Depend upon routine used.

## Hook codes and the Microdrive

The following hook codes are those which you will need to use if you want to access a Microdrive from machine code.

**CODE #21 – MTRON**

This hook code calls the routine which will turn on the motor of the specified drive. The number of the drive to be started should be held in the A register. If the number is 0 then all the motors will be turned off. When you start a drive the interrupts will be disabled and you will have to enable them before you use any routine which requires them.

*How to call MTRON*

RST #08
DEFB #21

*Entry conditions*

The A register should hold the number of the drive to be turned on (1 – 8).
If A is 0 then all drives are turned off.

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

If A is not 0 then interrupts will be off.
If A is 0 then interrupts will be on.

Let's start the motor on Microdrive number 1.

```
10         ORG  30000
20 MTRON   EQU  #21
30 START   LD   A,1
40         RST  #08
50         DEFB MTRON
60         EI
70 ; ENABLE INTERRUPTS AS SPE
   CTRUM REQUIRES THIS FOR OP
   ERATION
80         RET
```

If the drive is not connected or there is no formatted cartridge inside it then you will get the message *Microdrive not present.*

**CODE #22 – CREM**

Before any access can be made to a Microdrive there must be a Microdrive channel in the CHANS area of the Spectrum's memory for it to use. This hook code will CREate a Microdrive channel for you and the start address of the channel will be returned to you in the IX register.

When using this code you must make sure that

1 system variable D–STR1 holds the drive number.
2 N–STR1 holds the length of the filename followed by the address of the filename in T–STR1

This hook code is the machine code equivalent of the OPEN command used in BASIC and performs similar actions.

Once CREM has been called, the drive whose number is stored in D–STR1 is accessed, and the cartridge is searched for the filename that was requested. If the filename is found then that file is opened for reading and the first record is loaded into memory. If that filename can't be found on the Microdrive tape then the new file is created and opened for writing.

*How to call CREM*

RST #08
DEFB #22

*Entry conditions*

Ensure that interrupts are on.
D–STR1 (#5CD6,7)=drive number
N–STR1 (#5CDA,B)=length of filename
T–STR1 (#5CDC,D)=address where filename is stored

*Registers used by routine*

AF,BC,DE,HL,BC',DE',HL',IX

*Exit conditions*

IX register will hold start address of the channel being used.
HL holds the displacement of the stream.
Interrupts will be on.
A 595 byte buffer area will have been created.
If there was no map for the drive one is created.

**CODE #23 – CLOSM**

This hook code performs the same task as the CLOSE command that you use from BASIC. The start address of the Microdrive channel to be closed should be held by the IX register pair. If this is currently a write file then any data in the buffer is sent before the file is closed and the channel area is reclaimed.

*How to call CLOSM*

RST #08
DEFB #23

*Entry conditions*

IX register pair should hold the address of the Microdrive channel to be closed

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be switched on.
The 595 byte buffer is reclaimed.
The map area is reclaimed if not used by another channel.

**CODE #24 – DELFIL**

If you wanted to ERASE a file from a Microdrive using BASIC then you would use the command:

ERASE *"m";n;"filename"

where 'n' is the drive number of the Microdrive on which the file "filename" is stored.

DELFIL is the hook code equivalent of this instruction and will remove a named file from a Microdrive, if it exists. If the file you are trying to erase doesn't exist then an error condition will occur.

*How to call DELFIL*

RST #08
DEFB #24

*Entry conditions*

D–STR1 should contain the drive number.
T–STR1 should hold the address of the start of the filename.
N–STR1 should hold the length of the filename

*Registers used by routine*

AF,BC,DE,HL,BC',DE',HL',IX

*Exit conditions*

Interrupts will be on.

The following program shows how you could delete the file called 'DELETE' from drive 2.

```
 10          ORG   30000
 20 DSTR1    EQU   #5CD6
 30 NSTR1    EQU   #5CDA
 40 TSTR1    EQU   #5CDC
 50 FILE     DEFM "DELETE"
 60 START    RST   #08
 70          DEFB #31
 80 ;ENSURE SYSTEM VARIABLES A
    RE THERE.
 90          EXX
100          PUSH HL
110          EXX
120 ;MAKE SURE THE VALU OF HL'
     HAS BEEN SAVED AS IT
130 ;WILL BE REQUIRED FOR NORM
    AL OPERATION OF THE
140 ;SPECTRUM AFTER RETURNING
    TO BASIC
150          LD    HL,FILE
160          LD    (TSTR1),HL
170          LD    HL,#0006
180          LD    (NSTR1),HL
190 ;MAKE SURE SYSTEM VARIABLE
    S HOLD ALL THE RELEVENT
200 ;ADDRESSES
210          LD    HL,#0002
220          LD    (DSTR1),HL
230          RST   #08
240          DEFB #24
250 ;CALL DELETE FILE
260          EXX
```

```
270 ;GET HL' BACK FOR NORMAL O
    PERATION
280         POP  HL
290         EXX
300         RET
```

**CODE #25 – SEQRD**

This hook code is used to read the next record of a named file. The data from the record is read into the data buffer for that Microdrive channel. BIT 1 of RECFLG is checked to see if it is the last one, and if so, an end of file message will occur.

*How to call SEQRD*

RTS #08
DEFB #25

*Entry conditions*

IX should hold the address of the Microdrive area.
CHDRIV should hold the drive number.
CHNAME should hold the filename.

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be off.
The motor will be on.

**CODE #26 – SEQWR**

This is the hook code that will allow you to write a file of your own to Microdrive. The record will start to be saved from the first free sector. When called, the motor of the Microdrive is turned on and then a test is made to check that the tape is not full. Once the record has been written to tape the relevant BIT in the Microdrive memory map is set to show that it has been used.

*How to call SEQWR*

RST #08
DEFB #26

*Entry conditions*

IX should be the start of the Microdrive area.
CHBYTE, CHREC, CHNAME and CHDRIVE should hold the relevant data as for other Microdrive accesses.

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be off.
Microdrive motor will be on.

**CODE #27 – RANRD**

As previously mentioned, the hook code #25 is used to read the next record. This code works in a similar fashion but will read the record whose number is specified in the variable CHREC. When this routine is called the system variable SECTOR is set to #04FB, which will allow the drive to make five revolutions before coming up with a *file not found* error message if the requested file doesn't exist. If the file is not a PRINT file then an error will also be generated.

*How to call RANRD*

RST #08
DEFB #27

*Entry conditions*

CHDRIV should hold the drive number.
CHREC should hold the record number.
CHNAME should hold the file name.
IX should point to the start of the Microdrive channel

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be off.
Drive motor will be on.

**CODE #28 – SECRD**

This hook code also reads in the record specified by the CHREC variable. If the sector number is found and it is not a PRINT file the carry flag is set and the data is forgotten. If it is found and it is a PRINT file then the carry flag is reset and the buffer loaded in. If the relevant sector is not found after one revolution, then the *file not found* error is generated.

*How to use SECRD*

RST #08
DEFB #28

*Entry conditions*

CHREC should hold the number of the record that is to be searched for.
IX should point to the Microdrive area.

*Registers used by routine*
AF,BC,DE,HL

*Exit conditions*
Interrupts will be off.
Motor will be on.

**CODE #29 – NEXSEC**
Hook code #29 is used to retrieve the next sector that passes the read head of the Microdrive. The number of the sector read will be stored in the system variable HDNUMB.

*How to call NEXSEC*
RST #08
DEFW #29

*Entry conditions*
CHDRIV should hold the drive number.
IX should point to the start of the Microdrive channel.

*Registers used by routine*
AF,BC,DE,HL

*Exit conditions*
Interrupts will be off.
Drive motor will be on.
HDNUMB will contain the sector number.
Data will be in the buffer if the checksum was okay, i.e. if the carry flag is reset and the file was a print file. If the file was not a print file then the carry flag will be set and the data will be lost from the buffer.

**CODE #2A – RNDW**
This hook code will write the data that is currently held in the Microdrive buffer to the sector whose number is stored in the system variable CHREC. The write-protect tab is checked before the sector is written.

*How to call RNDW*
RTS #08
DEFB #2A

*Entry conditions*
CHREC should hold the record number.
CHDRIV should be set to the drive number.
CHNAME should hold the name of the file to be written.
IX should point to the start of the Microdrive channel.

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be off.
Motor will be on.
Microdrive buffer will have been written to tape.

*Testing the write-protect tab*

Not all the Microdrive routines check that the write-protect tab has not been broken off the cartridge. Obviously it would be useful to be able to test this within your programs so that you don't ruin an important cartridge. The routine below shows how you can do this.

```
IN A,(#EF)
AND 1
```

If the write-protect tab has been broken off then the zero flag will be set to 1. That is, JP Z,OFF would jump to the label OFF if the tab had been broken off.

## RS-232 and hook codes

Before using the RS-232 port of Interface 1 you must have the extra system variables present and must have stored the correct baud rate in the system variable BAUD at location #5CC3. You can work out the value to go into BAUD from the formula:

```
BAUD=(3500000/(26*rate))-2
```

It is also worth noting that the system variable IOBORD at #5CC6 holds the number that corresponds to the colour that the border will flash during I/O (Input or Output) operations. If you don't want the border to flash then simply store the same number in IOBORD as the present border colour.

### CODE #1D - IN232

This is the subroutine that is called to get a character from the RS-232 link. The carry flag will be set if the routine reads a character from the port. If the SPACE key is pressed, then a *break into program* error message will occur.

*How to call IN232*

```
RST #08
DEFB #1D
```

*Entry conditions*

N/A

*Registers used by routine*
AF,BC,DE,HL

*Exit conditions*
The carry flag will be set if a character has been read.
Register A will contain code if the carry flag is set.
Interrupts will be on.

**CODE #1E – OUT232**
This code will cause the byte of data held in the A register to be transmitted down the RS-232 line at the speed given by BAUD. The character will only be sent when the DTR input signal is held high.

*How to use OUT232*
RST #08
DEFB #1E

*Entry conditions*
A register to hold character code.

*Registers used by routine*
AF,BC,DE,HL

*Exit conditions*
Interrupts on

**Other RS-232 functions**
The two hook codes mentioned above allow you to transfer data using the "b" channel. If you wish to send data using the "t" channel or to open an RS-232 channel, then you must use the hook code #32 to call the following addresses respectively:

232TIN=#0C3C
OP232=#0B13

Reviewing the earlier section on using hook code #32 reveals that you must load HD-11 with the address of the routine to be called.

*How to use 232TIN or OP232*
For example, the routine to open an RS-232 channel is:

```
10          ORG   30000
20 HD-11    EQU   #5CED
30          LD    HL,#0B13
40          LD    (HD-11),HL
50          RST   #08
```

```
60          DEFB #32
70 ;..... REST OF PROGRAM
80 ;
```

*Entry conditions*

Interrupts will be on.

If you wish this to be a "B" channel then L–STR1 (#5CD9) should contain a "B".

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

DE will hold the address of the start of the new CHANS area.

### Hook codes and the network

It is worth having a look at the way in which the network functions before looking at the hook codes that relate to it.

Data is transmitted between stations in blocks of a maximum size of 255 bytes. Only one computer can be 'talking' at any one time. That is, station 1 can't talk to station 2 whilst it is talking to station 3. The step by step representation of the paths of data flow below shows at a glance how data is moved around.

(1) Sender checks to see if network is busy.
(2) If yes go back to (1).
(3) Transmit the header containing information on the data and the receiver's and sender's station numbers.
(4) If not broadcasting wait for acknowledgement from receiver. This will be the receiver's number.
(5) Check variable NTLEN. If it is 0 then that is the end of data and transmission. If it is greater than 0 then send the relevant number of bytes.
(6) Wait for acknowledgement (unless broadcast). If no acknowledgement is received then go back to (5).

#### CODE #2D – NOPEN

This is the command that is used to set up an "N" channel for temporary use whilst the network is being used. This routine will create a 255 byte area at the end of CHANS so that the data can be stored.

Once you make this channel the current one by storing its address in CURCHL you can write to it simply by using the RST #10 instruction.

*How to use NOPEN*

```
RST #08
DEFB #2D
```

*Entry conditions*

D–STR1 must hold the destination station number.
NSTAT must hold your own station number.

*Registers used by routine*

AF,BC,DE,HL

*Exit conditions*

IX will be the start of the network area.
DE and CURCHL will be as above.

**CODE #2E – NCLOSE**

Use this code to close the network channel whose code start address is held in the IX register pair and CURCHL. If you have been sending, the data in the buffer is sent together with an End of File marker. The 255 byte area is reclaimed.

*How to use NCLOSE*

```
RST #08
DEFB #2E
```

*Entry conditions*

CURCHL should hold the start address of the "N" channel to be deleted.

*Registers used by routine*

AF,BC,DE,HL,IX

*Exit conditions*

Interrupts on.
The 255 byte area is reclaimed by the system.

**CODE #2F – NINP**

This is the hook code that is used to receive data being sent to your Spectrum by another user on the network.

This hook code has a bug in it and will not function correctly. It is possible that Sinclair will correct the routine and so notes are included here on how to use it.

*How to use NINP*

```
RST #08
DEFB #2F
```

*Entry conditions*

IX register pair to point to the start of the N area.

*Registers used by NINP*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be on and data will be in the network area.

**CODE #30**–NWRT

This is the routine that is used to write a packet of 255 bytes over the network.

*How to use NWRT*

```
RST #08
DEFB #30
```

*Entry conditions*

The A register is to hold 0 for ordinary data or a 1 if this is the last packet in the file.
The packet should be in the correct area of memory (i.e. the 'n' buffer)
IX should point the 'n' channel

*Registers used*

AF,BC,DE,HL

*Exit conditions*

Interrupts will be on.
A=the destination number.

# 11 Extending Spectrum BASIC

### The limitations of your Spectrum

One of the criticisms often levelled at the Spectrum, other than moans about its rubber keyboard, is its rather limited BASIC. For example, there are a number of commands not available that would have been useful, such as RENUMBER, DELETE, MOVE SPRITE, COMPILE, ASSEMBLE and so on. With the Interface 1, however, it is now possible to add commands quite easily by a process that is the equivalent of telling the Spectrum that what it previously regarded as errors are now valid commands.

### Which command should be added?

Before you try to add any commands to Spectrum BASIC you will need to know what sort of names you can give to the new statements.

Well, you could use the existing commands and just alter the syntax. For example we could extend the INK command to read,

```
INK a,b,c
```

This new INK command could turn INK to colour a, PAPER to colour b and BORDER to colour c. In other words we have replaced the following

```
INK a:PAPER b:BORDER c:CLS
```

with this new command.

The other way to add commands would be to add your own new words to Spectrum BASIC. You can use any words you want, like COLOUR or FRED or BANG.

There is, however, one small problem. Let's take the example of the new word COLOUR. As soon as you type the 'C' the Spectrum will print out the word CONTINUE, this is because the Spectrum is expecting a keyword. Therefore before we can type in our new word we need to get the Spectrum out of 'K' mode. This can be done by prefixing all the new commands with a character that will put the cursor into normal mode. Examples of characters that we could use to do this are '*' or '!'. It is

sensible always to use the same prefix for your new commands so that you don't become confused about the syntax of them.

Therefore our new word to change the colours could now become:

*COLOUR a,b,c

## Dealing with errors

Before you can add any new commands to Spectrum BASIC it is essential that you understand the way in which the Spectrum checks the syntax of anything that you type in and how it generates error messages.

Let's examine what happens when you type in a BASIC command. When you have entered a line of BASIC it is entered into the edit buffer and onto the bottom of the screen, that is, via stream number 0.

As soon as you press the ENTER key the Spectrum will check to see that the syntax of the line is correct. This is done by checking which command you are using, for example, INPUT, and then calling the routine that checks the syntax for that command.

If you have used the command correctly the statement is accepted and if it was the last statement on the line then that line is transferred to its position in the program area.

If the syntax is not correct, error routine is called. This is done by using the RST #08 instruction followed by a byte that shows the type of error. The correct error message is then generated and the error is highlighted in the input line.

When the program is RUN the command is again found in the ROM and this time the RUN routine is called.

If, an error is found when it is executed, the error routine will be called. If every thing is alright the next command is found and executed.

*After Interface 1 is added.*

Once Interface 1 has been added to the Spectrum there are a lot of new commands available as well as some of the old ones with altered syntax.

Now, if an error is found when the Spectrum checks the syntax of a command then a check is made for it in the Interface 1 ROM. If the command is not there or the syntax is wrong, a return is made to the error routine in the Interface 1 ROM. The address that is returned to is held in the system variable VECTOR at location #5CB7. The routine that this points to is called ERR-6 and is located at #01F0.

By changing the address held in VECTOR to point to your own routine it is possible to extend what the Spectrum will accept as valid syntax, by adding checks for any new commands you may want to use.

If the syntax is correct, you must leave, using the routine ST-END (#05B7)

at syntax-checking time, or END1 (#05C1) at RUN-time. Since ST-END and END1 are Interface 1 ROM addresses do not expect to find them in any Spectrum ROM disassembly books. When you are generating an error message use:

```
RST #20
DEFB error number
```

if you wish to generate a shadow ROM error. Error numbers are:

00 – Nonsense in BASIC
01 – Invalid stream
02 – Invalid device expression
03 – Invalid name
04 – Invalid drive number
05 – Invalid station number
06 – Missing name
07 – Missing station number
08 – Missing drive number
09 – Missing baud rate
0A – Header mismatch error
0B – Stream already open
0C – Writing to a read file
0D – Reading a write file
0E – Drive-write protected
0F – Microdrive full
10 – Microdrive not present
11 – File not found
12 – Hook code error
13 – Code error
14 – Merge error
15 – Verification has failed
16 – Wrong file type
FF – Program finished

Alternatively, if you wish to generate an error held in the Spectrum ROM you must use the routine ROMERR. This can be called when the Interface 1 ROM is paged in by using the command RST#28.

To call this routine make sure that the variable ERRNR holds the appropriate error number and then use RST #28.

### How to check the syntax

When the Spectrum jumps to your routine the A register will hold the code

of the character or keyword that it is trying to find a match for. It is a simple matter to compare this code with your word by means of the CP instruction. Obviously if your word is more than one character long you will need to check the whole word before assuming it is your command. A couple of routines exist that will get the next character in the string. These are called GET–CHAR and NEXT–CHAR.

*GET–CHAR*
This can be called by using

```
RST #10
DEFW #0018
```

The A register will then contain the current character.

*NEXT–CHAR*
This can be called by using

```
RST #10
DEFW #0020
```

The A register will then contain the next character in the line being checked, ignoring control codes and spaces.

### Calling routines held in the Spectrum ROM

RST #10 is a call that will allow you to call any routine that is in the Spectrum ROM whilst the Interface 1 ROM is paged in. The two bytes that follow are the address of the routine to be called. For example, the routine held at location #0D68 will clear the Spectrum screen. To call this while the Interface 1 ROM is switched in we would use:

```
RST #10
DEFW #0D6B
```

### Other useful machine code routines

Before you add any more commands to Spectrum BASIC, it is worthwhile knowing a few of the important machine code routines that already exist in the Spectrum ROM which you can call to make life much easier for yourself.

**#0D6B** – *Clear screen*
Calling this routine will clear the Spectrum screen.

**#0DFE** – *Scroll screen*
Calling this routine will scroll the whole screen up by one character position but will not alter the print position.

**RST #10** – *Print to a stream*
By using the command RST #10 whilst in the Spectrum ROM the character in the A register will be printed out to the current stream.

**#1601** – *Select stream*
Calling this routine will make the stream number in the A register the current channel and therefore allocate the stream to the specified device.

**#0DD9** – *Move print position*
This routine will move the print position. Bit 1 of FLAGS at #5C3B must be 0 otherwise the printer's print position will be moved. B register should hold the row: 24 is the top of the screen, 0 is the bottom. C register is the column: 33 is the left hand side, 1 is the right hand side.

**#1C82** – *EXPT-1NUM*
This checks that the next character in the edit buffer is the start of a numeric expression. If it is not then the error routine is called with the *Nonsense in BASIC* error message. If a valid numeric expression is found, its value is left on the calculator stack.

**#1E94** – *FND-INT1*
This routine will fetch the last value from the calculator stack and move it into the A register.

**#1E99** – *FND-INT2*
This routine will fetch the last value from the calculator stack and move it into the BC register pair.

In both of the routines above the carry flag will be set if an overflow condition occurs and the *Integer out of range error* message will be given.

**#1C8C** – *Read string (CLASS-0A)*
This routine will check to see if the next expression is a string. If it is not then the error routine will be called. The error given will be *Nonsense in BASIC*.

**#1A28** – *Print 16-bit*
This routine is used to print a four digit 16-bit number. The HL register pair should hold the location where the number is stored.

NB. this routine assumes that the number is stored MSB first and *not* LSB first as you would expect.

**#1F54** – *Test for Break*
This routine will test for the Break key. If SHIFT and BREAK are being pressed then the carry flag will be reset.

*#0C0A – Print message*

This will print out a message from a table. The A register should hold the entry number of the message in the table. The DE register pair holds the start address of the table.

Each message in the table should have the last character inverted (add #80 to it) so that the routine knows where the message ends and the next one starts. I have already detailed the method for adding new commands to the Spectrum. Below are a number of example programs that show this being done. All are well documented but if you do have problems I suggest that you read a book about Z80 machine code, and explore the subject further.

*Program 1 – MESSAGE*

This program demonstrates quite simply how a command can be added that will print out a message. The command that you must use in this example is:

```
*M
```

The message printed out is 'hello I'm a new command' but you could change it to anything that you want. The routine that is in the Spectrum ROM is used to print the message, so you must always remember to add #80 to the last letter so that the Spectrum will know when it reaches the end.

```
10 ; ROUTINE TO PRINT OUT A M
ESSAGE BY TYPING
20 ; *M
30 ; DON'T FORGET TO ALTER VE
  CTOR TO
40 ; POINT TO START OF THIS P
  ROG.
50 ;    S.COOK    '84
60          ORG   32000
70 CALBAS EQU   #10
80 GETCHR EQU   #0018
90 NXTCHR EQU   #0020
100 STEND  EQU   #05B7
110 ERR6   EQU   #01F0
120 END1   EQU   #05C1
130 CLS    EQU   #0D6B
140 SELSTR EQU   #1601
150 PRNTAT EQU   #0DD9
160 PRNMES EQU   #0C0A
```

```
170 ;
180         RST  CALBAS   ;GET
   FIRST CHARACTER
190         DEFW GETCHR
200         CP   "*"
210 ERROR   JP   NZ,ERR6  ;IF N
   OT "*" THEN DO AN ERROR
220         RST  CALBAS   ;GET
   NEXT CHR
230         DEFW NXTCHR
240         OR   #20      ;AND
   MAKE IT LOWER CASE
250         CP   "m"
260         JP   NZ,ERR6  ;IF N
   OT m THEN DO ERROR
270         RST  CALBAS
280         DEFW NXTCHR
290         CALL STEND    ;END
   HERE IN SYNTAX TIME
300 RUN     RST  CALBAS   ;CALL
    CLEAR SCREEN ROUTINE
310         DEFW CLS
320         LD   A,2      ;SET
   UP STREAM TO 2
330         RST  CALBAS
340         DEFW SELSTR
350         LD   B,24     ;SET
   UP PRINT POSITION
360         LD   C,33
370         RST  CALBAS
380         DEFW PRNTAT
390         LD   A,0      ;POIN
   T TO FIRST MESSAGE
400         LD   DE,MESS
410         RST  CALBAS   ;CALL
    PRINT MESSAGE ROUTINE
420         DEFW PRNMES
430         JP   END1
440 ; START OF MESSAGE TABLE
450 MESS    DEFB #80
460         DEFM "HELLO I'M A N
   EW COMMAN"
470         DEFB "D"+#80  ;LAST
   CHR IN STRING MUST BE +#80
```

*Program 2 – MESSAGE2*

This program expands on the program MESSAGE. Now you can add a parameter to the *M command causing the relevant message to be printed. I have included four messages numbered 0 to 3. To see message number two you would type:

```
*M2
```

Once you have figured out how this routine uses the EXPT1N routine you should have no problem writing your own machine code program and being able to pass parameters freely to it.

```
 10 ; STUART COOKE  '84
 20 ; ROUTINE TO PRINT OUT MES
   SAGE DEPENDING ON NUMBER
 30 ; TO CALL USE
 40 ;             *Mnumber
 50 ; WHERE number is 0 to 3
 60 ; THIS SHOWS HOW YOU CAN P
   ASS PARAMETERS
 70 ;
 80         ORG  32000
 90 CALBAS  EQU  #10
100 GETCHR  EQU  #0018
110 NXTCHR  EQU  #0020
120 STEND   EQU  #05B7
130 ERR6    EQU  #01F0
140 END1    EQU  #05C1
150 CLS     EQU  #0D6B
160 SELSTR  EQU  #1601
170 PRNTAT  EQU  #0DD9
180 PRNMES  EQU  #0C0A
190 EXP1N   EQU  #1C82
200 GET1N   EQU  #1E94
210         RST  CALBAS    ;READ
    IN FIRST CHARACTER
220         DEFW GETCHR
230         CP   "*"       ;IS I
   T "*"
240 ERROR   JP   NZ,ERR6   ;IF N
   OT THEN CALL ERROR
250
260         RST  CALBAS    ;NEXT
    CHARACTER
```

```
270         DEFW NXTCHR
280         OR   #20       ;MAKE
    SURE ITS LOWER CASE
290         CP   "m"       ;IS A
   N "m"
300         JP   NZ,ERROR ;IF
   NOT THEN ERROR
310         RST  CALBAS    ;NEXT
    CHARACTER
320         DEFW NXTCHR
330         RST  CALBAS    ;MAKE
   SURE ITS A NUMBER
340         DEFW EXP1N
350         CALL STEND     ;CALL
    SYNTAX END
360 RUN     RST  CALBAS    ;CLEA
   R SCREEN
370         DEFW CLS
380         LD   A,2       ;SELE
   CT STREAM 2
390         RST  CALBAS
400         DEFW SELSTR
410         LD   B,22      ;SET
   UP SCREEN POSITION
420         LD   C,33
430         RST  CALBAS
440         DEFW PRNTAT
450         RST  CALBAS    ;GET
   THE NUMBER OFF STACK
460         DEFW GET1N
470         LD   DE,MESS   ;PRIN
   TOUT MESSAGE NO. IN A
480         RST  CALBAS
490         DEFW PRNMES
500         JP   END1
510 ;START OF MESSAGE TABLE
520 MESS    DEFB #80       ;STAR
   T TABLE WITH #80
530         DEFM "HELLO I'M A N
   EW COMMAN"
540         DEFB "D"+#80   ;EVER
   Y MESSAGE TO BE INVERTED
550         DEFM "MY NAME IS NU
   MBER "
```

```
560         DEFB "1"+#80
570         DEFM "AND I'M COMMA
   ND "
580         DEFB "2"+#80
590         DEFM "NUMBER 3 IS M
   Y NAM"
600         DEFB "E"+#80
```

*Program 3 – COLOUR*

```
 10 ;NEW MICRODRIVE COMMAND
 20 ;(C) 1984 DAVID JONES
 30 ;
 40 ;SYNTAX :-
 50 ;  *COLOUR I,P,B
 60 ;
 70 ;WHERE I=INK
 80 ;       P=PAPER
 90 ;       B=BORDER
100 ;
110 GETCHR EQU  #0018
120 NXTCHR EQU  #0020
130 STEND  EQU  #05B7
140 END1   EQU  #05C1
150 EXPTIN EQU  #1C82
160 FNDINT EQU  #1E94
170 BORDER EQU  #2294
180 CLS    EQU  #0D6B
190 ATTRP  EQU  #5C8D
200 ;
210        ORG  30000
220 ;
230 START  RST  #10      ;CALL
     SPECTRUM ROM GETCHR
240        DEFW GETCHR
250        CP   "*"
260        JP   Z,NEWSYN ;JUMP
     IF * IS FOUND ELSE ERROR
270 ERROR  RST  #20      ;SHAD
    OW ROM ERROR CALL
280        DEFB 0        ;NONS
    ENSE IN BASIC
290 ;
```

```
300 ;
310 ERROR2 LD   (IY+0),#13
320 ;INVALID COLOUR ERROR
330        RST  #28      ;CALL
     ROM ERROR ROUTINE
340 ;
350 NEWSYN RST  #10      ;NEW
    SYNTAX IS CHECKED FIRST
360        DEFW NXTCHR
370        RES  5,A      ;FIX
    AS UPPER CASE
380        CP   "C"
390        JP   NZ,ERROR ;ERRO
    R IF NOT C
400        RST  #10
410        DEFW NXTCHR
420        RES  5,A
430        CP   "O"
440        JP   NZ,ERROR ;ERRO
    R IF NOT O
450        RST  #10
460        DEFW NXTCHR
470        RES  5,A
480        CP   "L"
490        JP   NZ,ERROR ;ERRO
    R IF NOT L
500        RST  #10
510        DEFW NXTCHR
520        RES  5,A
530        CP   "O"
540        JP   NZ,ERROR ;ERRO
    R IF NOT O
550        RST  #10
560        DEFW NXTCHR
570        RES  5,A
580        CP   "U"
590        JP   NZ,ERROR ;ERRO
    R IF NOT U
600        RST  #10
610        DEFW NXTCHR
620        RES  5,A
630        CP   "R"
640        JP   NZ,ERROR ;ERRO
    R IF NOT R
```

```
650 ;
660 ;THE COMMAND IS *COLOUR SO
     FAR
670 ;NOW TO CHECK THE PARAMETE
    RS
680 ;
690         RST  #10       ;GET
    INK
700         DEFW NXTCHR
710         RST  #10
720         DEFW EXPT1N
730         CP   ","
740         JP   NZ,ERROR
750         RST  #10       ;GET
    PAPER
760         DEFW NXTCHR
770         RST  #10
780         DEFW EXPT1N
790         CP   ","
800         JP   NZ,ERROR
810         RST  #10       ;GET
    BORDER
820         DEFW NXTCHR
830         RST  #10
840         DEFW EXPT1N
850 ;
860 ; SYNTAX IS ALL OK
870 ;
880         CALL STEND
890 ;RETURN TO BASIC IF CHECKI
    NG SYNTAX ONLY
900 ;
910 ;BORDER IS THE FIRST VALUE
920 ;ON THE CALCULATOR STACK
930 ;SINCE IT WAS THE LAST TO
940 ;BE PLACED THERE
950 ;
960 NEWRUN RST  #10       ;BORD
    ER
970         DEFW BORDER    ;CALL
     THE ROM BORDER ROUTINE FO
    R SIMPLICITY
980         RST  #10       ;PAPE
    R
```

```
 990         DEFW FNDINT
1000         CP   8          ;LIMI
     T PAPER Ø TO 7
1010         JP   NC,ERROR2
1020         OR   A          ;CLEA
     R CARRY FLAG
1030         RLCA
1040         RLCA
1050         RLCA ;A=8*PAPER
1060         LD   E,A
1070         PUSH DE         ;SAVE
      PAPER *8 IN THE E REGISTE
     R
1080         RST  #10        ;INK
1090         DEFW FNDINT
1100         CP   8          ;LIMI
     T INK Ø TO 7
1110         JP   NC,ERROR2
1120 ;
1130         POP  DE
1140         ADD  A,E        ;A=IN
     K+8*PAPER
1150         LD   (ATTRP),A
1160 ;PLACE THE NEW ATTRIBUTES
     IN THE SYSTEM VARIABLE ATT
     RP
1170         LD   HL,#5800 ;CLEA
     R THE ATTRIBUTES TO NEW IN
     K AND PAPER
1180         LD   DE,#5801 ;BRIG
     HT AND FLASH ARE SET TO ZE
     RO
1190         LD   BC,#2FF
1200 ;
1210         LD   (HL),A
1220         LDIR
1230         JP   END1       ;END
     OF NEW COMMAND
```

### A new Microdrive command

The command mentioned previously can now be listed as you have all the relevant information to be able to understand what is going on. The command is:

*COLOUR i,p,b

where i=INK, p=PAPER, and b=BORDER. i,p and b can only be in the range 0 to 7 since this new command is given simply to demonstrate the principle of adding commands. It is the same as the BASIC program below:

```
  10 LET i=2: LET p=4: LET b=6
  20 INK i: PAPER p: BORDER b: L
ET c=i+8*p
  30 FOR x=22528 TO 23295
  40 POKE x,c
  50 NEXT x
```

but is much faster and simpler to use. To make this routine work two POKEs are required:

POKE 23735,48 and POKE 23736,117

These POKEs will set the Microdrive system variable VECTOR to point to 30000. Don't forget to reserve high memory for the machine code by using CLEAR 29999.

In the assembly listing below the use of RES 5,A to FIX upper case will enable the *COLOUR command to be interpreted correctly in either upper or lower case, or even a mixture of both. *ColoUR i,p,b would be quite valid.

```
  10 ;NEW MICRODRIVE COMMAND
  20 ;(C) 1984 DAVID JONES
  30 ;
  40 ;SYNTAX :-
  50 ;   *COLOUR I,P,B
  60 ;
  70 ;WHERE I=INK
  80 ;       P=PAPER
  90 ;       B=BORDER
 100 ;
 110 GETCHR EQU  #0018
 120 NXTCHR EQU  #0020
```

```
130 STEND   EQU  #05B7
140 END1    EQU  #05C1
150 EXPTIN  EQU  #1C82
160 FNDINT  EQU  #1E94
170 BORDER  EQU  #2294
180 CLS     EQU  #0D6B
190 ATTRP   EQU  #5C8D
200 ;
210         ORG  30000
220 ;
230 START   RST  #10       ;CALL
     SPECTRUM ROM GETCHR
240         DEFW GETCHR
250         CP   "*"
260         JP   Z,NEWSYN ;JUMP
     IF * IS FOUND ELSE ERROR
270 ERROR   RST  #20       ;SHAD
    OW ROM ERROR CALL
280         DEFB 0         ;NONS
    ENSE IN BASIC
290 ;
300 ;
310 ERROR2  LD   (IY+0),#13
320 ;INVALID COLOUR ERROR
330         RST  #28       ;CALL
     ROM ERROR ROUTINE
340 ;
350 NEWSYN  RST  #10       ;NEW
    SYNTAX IS CHECKED FIRST
360         DEFW NXTCHR
370         RES  5,A       ;FIX
    AS UPPER CASE
380         CP   "C"
390         JP   NZ,ERROR ;ERRO
    R IF NOT C
400         RST  #10
410         DEFW NXTCHR
420         RES  5,A
430         CP   "O"
440         JP   NZ,ERROR ;ERRO
    R IF NOT O
450         RST  #10
460         DEFW NXTCHR
470         RES  5,A
```

```
480         CP    "L"
490         JP    NZ,ERROR ;ERRO
    R IF NOT L
500         RST   #10
510         DEFW  NXTCHR
520         RES   5,A
530         CP    "O"
540         JP    NZ,ERROR ;ERRO
    R IF NOT O
550         RST   #10
560         DEFW  NXTCHR
570         RES   5,A
580         CP    "U"
590         JP    NZ,ERROR ;ERRO
    R IF NOT U
600         RST   #10
610         DEFW  NXTCHR
620         RES   5,A
630         CP    "R"
640         JP    NZ,ERROR ;ERRO
    R IF NOT R
650 ;
660 ;THE COMMAND IS *COLOUR SO
     FAR
670 ;NOW TO CHECK THE PARAMETE
    RS
680 ;
690         RST   #10       ;GET
    INK
700         DEFW  NXTCHR
710         RST   #10
720         DEFW  EXPT1N
730         CP    ","
740         JP    NZ,ERROR
750         RST   #10       ;GET
    PAPER
760         DEFW  NXTCHR
770         RST   #10
780         DEFW  EXPT1N
790         CP    ","
800         JP    NZ,ERROR
810         RST   #10       ;GET
    BORDER
820         DEFW  NXTCHR
```

```
830         RST  #10
840         DEFW EXPT1N
850 ;
860 ; SYNTAX IS ALL OK
870 ;
880         CALL STEND
890 ;RETURN TO BASIC IF CHECKI
    NG SYNTAX ONLY
900 ;
910 ;BORDER IS THE FIRST VALUE
920 ;ON THE CALCULATOR STACK
930 ;SINCE IT WAS THE LAST TO
940 ;BE PLACED THERE
950 ;
960 NEWRUN RST  #10       ;BORD
    ER
970         DEFW BORDER   ;CALL
     THE ROM BORDER ROUTINE FO
    R SIMPLICITY
980         RST  #10      ;PAPE
    R
990         DEFW FNDINT
1000        CP   8        ;LIMI
    T PAPER 0 TO 7
1010        JP   NC,ERROR2
1020        OR   A        ;CLEA
    R CARRY FLAG
1030        RLCA
1040        RLCA
1050        RLCA ;A=8*PAPER
1060        LD   E,A
1070        PUSH DE       ;SAVE
     PAPER *8 IN THE E REGISTE
    R
1080        RST  #10      ;INK
1090        DEFW FNDINT
1100        CP   8        ;LIMI
    T INK 0 TO 7
1110        JP   NC,ERROR2
1120 ;
1130        POP  DE
1140        ADD  A,E      ;A=IN
    K+8*PAPER
1150        LD   (ATTRP),A
```

```
1160 ;PLACE THE NEW ATTRIBUTES
     IN THE SYSTEM VARIABLE ATT
     RP
1170 .       LD   HL,#5800 ;CLEA
     R THE ATTRIBUTES TO NEW IN
     K AND PAPER
1180         LD   DE,#5801 ;BRIG
     HT AND FLASH ARE SET TO ZE
     RO
1190         LD   BC,#2FF
1200 ;
1210         LD   (HL),A
1220         LDIR
1230         JP   END1     ;END
     OF NEW COMMAND
```

*Program 4 – NEWCOM*

LOADing or SAVEing a program onto Microdrive requires quite a number of keypresses. Wouldn't it be easier if we could simply type:

*L"1fred"

to LOAD the file fred from drive 1? Well this is exactly what this machine code program does; it alters the syntax for LOAD, SAVE, VERIFY and MERGE. Of course you can still use the old commands as well, as the table below shows.

| OLD SYNTAX | NEW SYNTAX |
|---|---|
| LOAD *"M";N;"FILE" | *L"NFILE" |
| SAVE *"M";N;"FILE" | *S"NFILE" |
| VERIFY *"M",N;"FILE" | *V"NFILE" |
| MERGE *"M";N;"FILE" | *M"NFILE" |

```
100 ;       STUAT COOKE     '84
110 ;
120 ; THIS MACHINE CODE PROGRA
   M WILL
130 ; CHANGE THE SYNTAX OF THE
    MICRODRIVE
140 ; COMMANDS
150 ; LOAD BECOMES *L"driveNAM
   E"
160 ; SAVE          *S
```

```
170 ; VERIFY        *V
180 ; MERGE         *M
190 ;
200 ; VECTOR (23735,6) MUST BE
    ALTERED
210 ; TO POINT TO THIS ROUTINE
220 ;
230 ; IY ALWAYS POINTS TO SYST
   EM VARIABLE
240 ; ERRNR AT 23610
250 ;
260         ORG  32000
270 ;
280 ; SET UP ALL SYSTEM ADDRES
   SES
290 ;
300 ERR6    EQU  #01F0
310 CALBAS  EQU  #10
320 GETCHR  EQU  #0018
330 NXTCHR  EQU  #0020
340 SAVER   EQU  #0836 ;INTERFA
   CE 1
350 ;SAVE   ROUTINE
360 LOADR   EQU  #08A5 ;INTERFA
   CE 1
370 ;LOAD ROUTINE
380 LSTR1   EQU  #5CD9
390 CLASSA  EQU  #1C8C
400 SYN     EQU  #18
410 SYNERR  EQU  #20
420 ENDSHA  EQU  #0723
430 STKFET  EQU  #2BF1
440 ERR     EQU  #064C
450 DSTR1   EQU  #5CD6     ;DRIV
   E NO
460 DSTR1H  EQU  #5CD7
470 NSTR1   EQU  #5CDA     ;LENG
   TH OF NAME
480 FSTR1   EQU  #5CDC     ;ADDR
   ESS OF NAME
490 ;
500 ; START OF SYNTAX CHECK
510 ;
```

```
520         CP    #5C       ;"*"-
   206
530 ERROR   JP    NZ,ERR6
540         RST   CALBAS
550         DEFW  NWTCHR
560         RES   5,A
570         CP    "L"
580         JR    Z,LOADM
590         CP    "S"
600         JR    Z,SAVEM
610         CP    "M"
620         JR    Z,MERGEM
630         CP    "V"
640         JR    Z,VERM
650         JR    ERROR
660 ;
670 ;SIGNAL LOAD
680 ;
690 LOADM   SET   4,(IY+124)
700         JR    LVM
710 ;
720 ;SIGNAL SAVE
730 ;
740 SAVEM   SET   5,(IY+124)
750         CALL  INTS
760         JP    SAVER
770 ;
780 ;SIGNAL MERGE
790 ;
800 MERGEM  SET   6,(IY+124)
810         JR    LVM
820 ;
830 ;SIGNAL VERIFY
840 ;
850 VERM    SET   7,(IY+124)
860 LVM     CALL  INTS
870         JP    LOADR
880 ;
890 ; INTerrogate the String
900 ;
910 INTS    LD    A,"M"
920         LD    (LSTR1),A     ;
   DEVICE ="M"
930         RST   CALBAS
```

```
 940         DEFW NXTCHR
 950         RST  CALBAS
 960         DEFW CLASSA    ;CHEC
    K FOR STRING
 970         RST  CALBAS
 980         DEFW GETCHR
 990         RST  SYN       ;WILL
     RETURN Z
1000 ;IF SYNTAX ONLY
1010         JP   Z,ENDSHA ;STOP
     HERE IF SYNTAX
1020 ;
1030         RST  CALBAS
1040         DEFW STKFET    ;WILL
     RETURN WITH
1050 ;BC   = LENGTH OF STRING
1060 ;DE   = START OF STRING
1070 ;       LD   HL,11     ;MAX
    LENGTH OF STRING
1080         AND  A
1090         SBC  HL,BC     ;CHEC
    K NOT GREATER
1100         JP   C,ERR     ;THAN
     11 CHARS
1110         LD   A,B
1120         OR   C
1130         JP   Z,ERR
1140         LD   A,(DE)
1150         SUB  #30       ;CHEC
    K DRIVE NO.
1160         CP   #01       ;NOT
    < 1
1170         JR   C,DRVERR
1180         CP   #09       ;AND
    NOT >8
1190         JR   NC,DRVERR
1200         LD   (DSTR1),A   ;
    SET UP NO.
1210         XOR  A
1220         LD   (DSTR1H),A  ;
    OF DRIVE
1230         DEC  BC
1240         INC  DE
```

```
1250          LD    (NSTR1),BC  ;
    NAME LEN.
1260          LD    (FSTR1),DE  ;
    NAME ADDR.
1270          RST   CALBAS
1280          DEFW  GETCHR
1290          JP    ENDSHA      ;
    BACK TO NORMAL
1300 ;
1310 ; GENERATE SHADOW ROM ERRO
    R
1320 ;
1330 DRVERR RST   SYNERR
1340          DEFB 4          ;ERRO
    R NO. 4
1350 ; = INVALID DRIVE
```

# 12 Jolly joysticks

With the introduction of the Interface 2, Sinclair Research has at last permitted the Spectrum to be used with standard joysticks, to give the avid games player the freedom of movement that is already so much appreciated by owners of more expensive, though not necessarily more advanced computers. As an added bonus, the Interface 2 contains not only two joystick connector ports but also a cartridge port that allows the use of ROM cartridges with the Spectrum for the first time.

The Interface 2 can never be considered as having the same level of sophistication as the Interface 1. Inside the large, black cartridge there is nothing more than a circuit board connected to the various ports. Between them these accommodate the connection of the ROM cartridge and the joysticks, plus a logic chip that is used to decode the joysticks' movements. To the rear of the Interface is a partial duplication of the standard rear-edge connections on the Spectrum. This connection can be used with the ZX Printer and it is, in fact, possible to have Interfaces 1 and 2 and the ZX Printer connected to the Spectrum simultaneously to provide a complete computer system.

### The cartridge port

The joystick ports will be of most immediate use and relevance to the BASIC programmer and we shall describe their use in a short while. The machine code programmer is also well served by the joystick ports and he will no doubt feel the adrenalin coursing through his veins when he savours the prospects of using the ROM cartridge port for his own nefarious purposes. First, however, let us examine the cartridge port. Although at first sight the cartridge port appears plain and unexciting, it is this hole, neatly covered by a hinged lid when not in use, that can provide the means of expanding and developing the Spectrum computer to your heart's content. For a start, you can now buy ROM-based games software to run on the Spectrum that loads instantly and, unlike cassette or Microdrive-based programs, will never become damaged or corrupted. There is already an impressive range of games cartridges available and the collection will no doubt grow rapidly. Each cartridge contains 16k of ROM memory so no game or any other program can be larger than this

size. This obvious limitation should not be underestimated because whilst it is possible, using a cassette or Microdrive-based program, to load several times this size into the 48k Spectrum, this whole area of RAM is denied to the ROM user for program storage (although variables and data can be placed into RAM by the program itself). Furthermore, this restriction is compounded by the fact that the ROM cartridge replaces the BASIC ROM in the Spectrum memory map. Thus the ROM program is unable to draw upon the 16k of BASIC and operating system routines that normally reside within that area of memory, and any routines that are required have to be entirely contained within the ROM of the cartridge. This restriction is not shared by the 'shadow' ROM that is within the Interface 1; this can be switched in and out in exactly the same way as we have seen previously (although you have to write a machine code routine actually to do the switching).

Whilst this replacement of the BASIC ROM with that in the cartridge can at times be a distinct disadvantage, in other circumstances it can be thought of as a benefit. If the cartridge contains an applications program, for instance a word-processing package, the use of the ROM space within memory for the actual program frees the entire RAM area for storage of large quantities of data (in this case text), thereby allowing use of a much more powerful software package. On the other hand, other languages, such as Logo, Pascal, Forth and so on, could be loaded from a ROM cartridge to replace the resident BASIC whilst still maintaining a large amount of RAM for User programs. It is even possible to 'blow' your own EPROMS and create your own cartridge software. To do this, however, you must be an extremely competent machine code programmer; you have no more access to BASIC so you can't continue to program in BASIC!

## The joystick ports

Whilst the advantages of the ROM cartridge slot are more assumed than real the same cannot be said for the two games joystick ports on the top of the Interface 2. With the addition of these ports to his system, the Spectrophile can easily use joysticks to replace or supplement the use of his keyboard in both BASIC and machine code programs.

Each port comprises what has become the standard Atari and Commodore type of connector: a 9-pin 'D' plug that permits the connection of a wide range of switch-type joysticks. You should note that although games paddles too can be connected to a joystick port, neither the Spectrum nor the Interface contain the right connections to allow proper use of paddles or analogue (or proportional) joysticks.

The implementation of the joystick ports could not be more simple, especially from BASIC. There are two commands available, INKEY$ and IN. Rather cleverly, Sinclair has made the joystick movements imitate the

pressing of particular keys on the keyboard. In effect, therefore, you can incorporate the use of joysticks within your own program without having to write any extra coding, (providing you make the correct choice of keys), because you can use the same set of INKEY$ statements to test for them both. For joystick 1, here are the number keys from 6 to 0 and their corresponding joystick movments:

| Key | Joystick Action |
|---|---|
| 6 | LEFT |
| 7 | RIGHT |
| 8 | DOWN |
| 9 | UP |
| 0 | FIRE |

Similarly, for joystick 2 the keys are as follows:

| Key | Joystick Action |
|---|---|
| 1 | LEFT |
| 2 | RIGHT |
| 3 | DOWN |
| 4 | UP |
| 5 | FIRE |

By using INKEY$, therefore, one is able to check for the appropriate keypress (simulated or real) and direct the flow of the program accordingly. To see how this works, type in the next small routine which will allow you to control the movements of a character on the screen by using either keys 1 to 4 or joystick 2:

```
  10 LET c$=CHR$ 138
  20 LET b$=CHR$ 32
  30 LET x=1: LET x1=1
  40 LET y=1: LET y1=1
  50 LET K$=INKEY$
  60 IF K$="" THEN GO TO 50
  70 IF K$="1" AND x>0 THEN LET
x=x-1
  80 IF K$="2" AND x<31 THEN LET
 X=X+1
  90 IF K$="3" AND y<20 THEN LET
 y=y+1
 100 IF K$="4" AND y>0 THEN LET
y=y-1
 110 PRINT AT y1,x1;b$
```

```
120 PRINT AT y,x;c$
130 LET x1=x: LET y1=y
140 GO TO 50
```

Looking at this type of routine you should be able to detect one of the limitations of mirroring keypresses for joystick controls: you can only achieve movement in one direction at a time. Pressing two keys on the keyboard simultaneously will not be picked up by the INKEY$ statement, neither will the synchronous pressing of the FIRE button together with any other directional movement on the joystick. The net result of such enthusiastic and concurrent reactions will probably be no more than death at the hands of an invading alien!

This particular problem can be overcome by using the IN statement from BASIC or by testing the relevant register from machine code. Each joystick port has a register assigned to it and all movements can be deciphered from its momentary contents. The register for joystick 1 is at memory location 61438 and that for joystick 2 is at 63486. By using the IN statement, (which in itself is no more than a different form of PEEK since it reads the contents of the specified byte of memory), we can assign the value returned to a variable and then start testing for particular joystick movements. Only the lowest five bytes of each register are used and they are assigned as follows:

| **Action** | **Stick 1** | **(Key)** | **Stick 2** | **(Key)** |
|---|---|---|---|---|
| | IN 61438 | | IN 63486 | |
| FIRE | Bit 0 | (0) | Bit 4 | (5) |
| UP | Bit 1 | (9) | Bit 3 | (4) |
| DOWN | Bit 2 | (8) | Bit 2 | (3) |
| RIGHT | Bit 3 | (7) | Bit 1 | (2) |
| LEFT | Bit 4 | (6) | Bit 0 | (1) |

The Spectrum computer does not include any commands within its BASIC language to allow the BIT-wise manipulation of numbers. That is to say, testing to check whether a particular BIT is set or unset is not possible, so quite an amount of manipulation is required before we can reach our objective of deciding in which direction the joystick has been moved. This manipulation is, in fact, quite slow and clumsy but the necessary effort is justified by the result. The following routine demonstrates this. In it we are only going to use the setting of a series of flags to show the movement of the joystick, the values of which are then printed to the screen.

Incidentally, it is worth remembering that BIT-wise manipulation can be performed relatively easily using Z80 machine code, the native language of the Sinclair Spectrum.

```
  10 CLS
  20 LET up=0: LET down=0
  30 LET left=0: LET right=0
  40 LET fire=0
  50 LET move=255-IN 61438
  60 IF move>127 THEN LET move=m
ove-128
  70 IF move>63 THEN LET move=mo
ve-64
  80 IF move>31 THEN LET move=mo
ve-32
  90 IF move>15 THEN LET move=mo
ve-16: LET left=1
 100 IF move>7 THEN LET move=mov
e-8: LET right=1
 110 IF move>3 THEN LET move=mov
e-4: LET down=1
 120 IF move>1 THEN LET move=mov
e-2: LET up=1
 130 IF move=1 THEN LET fire=1
 140 PRINT left;right;down;up;fi
re
 150 GO TO 20
```

When you RUN this routine, you should notice that pressing any combination of the five keys 6 to 0 will be detected and shown in the relevant flag. Similarly, any of the possible joystick movement combinations will also be flagged. You can, with suitable modification, use this routine within your own programs. It is probably best to turn the whole thing into a subroutine, by changing line 150 to:

```
150 RETURN
```

and then calling the subroutine whenever you wish to read the joystick movement and/or the FIRE button. The next program illustrates this technique and has the further merit of being a utility that will help you to design more attractive screen displays for interesting backdrops for your games and title displays, or possibly more serious uses. The direction and movement of drawing can be controlled by a joystick plugged into the port and the finished screen can be saved onto a cassette or Microdrive.

```
  10 REM JOYSTICK SKETCH PROGRAM
  20 REM S.COOKE , AUG 84
  25 :
  30 REM SET UP INITIAL DEFAULTS
  40 BORDER 7
  50 PAPER 7
  60 INK 0
  70 CLS
  80 LET XCORD=120
  90 LET YCORD=88
  95 LET BCOL=7
 100 LET PCOL=7
 110 LET ICOL=0
 120 LET DRAW=0
 130 LET M$=" X=     Y=     P=    I
=   DRAW=     "
1000 REM MAIN LOOP
1010 GO SUB 3000
1020 IF DRAW=1 THEN  PLOT  PAPER
 PCOL; INK ICOL;XCORD,YCORD
1030 GO SUB 7000
1040 IF RIGHT=1 THEN  LET XCORD=
XCORD+1: IF XCORD>255 THEN  LET
XCORD=255
1050 IF LEFT=1 THEN  LET XCORD=X
CORD-1: IF XCORD<0 THEN  LET XCO
RD=0
1060 IF UP=1 THEN  LET YCORD=YCO
RD+1: IF YCORD>175 THEN  LET YCO
RD=175
1070 IF DOWN=1 THEN  LET YCORD=Y
CORD-1: IF YCORD<0 THEN  LET YCO
RD=0
1080 IF FIRE=1 AND DRAW=0 THEN
LET DRAW=1: PAUSE 30: GO TO 1100
1090 IF FIRE=1 AND DRAW=1 THEN
LET DRAW=0: PAUSE 30
1100 GO SUB 4000
1200 GO TO 1000
3000 REM PRINT STATUS LINE ON
3010 REM BOTTOM LINE OF SCREEN
3020 PRINT #0; INK 9;AT 1,0;M$;
```

```
3030 PRINT #0; INK 9;AT 1,3;XCOR
D;AT 1,9;YCORD;AT 1,15;PCOL;AT 1
,20;ICOL;AT 1,28;DRAW
3040 RETURN
4000 REM CHECK FOR KEYPRESSES
4010 LET K$=INKEY$
4020 IF K$="C" THEN  GO SUB 5000
4030 IF K$="B" THEN  GO SUB 5500
4040 IF K$="I" THEN  GO SUB 6000
4050 IF K$="P" THEN  GO SUB 6500
4060 IF K$="L" THEN  GO SUB 6700
4070 IF K$="S" THEN  GO SUB 6800
4080 IF K$="M" THEN  GO SUB 4500
4100 RETURN
4500 REM NEW X,Y COORDS
4510 INPUT "ENTER NEW X CO-ORD "
;X
4520 IF X<0 OR X>255 THEN  RETUR
N
4530 LET XCORD=X
4540 INPUT "ENTER NEW Y CO-ORD "
;Y
4550 IF Y<0 OR Y>175 THEN  RETUR
N
4560 LET YCORD=Y
4570 GO SUB 9000
4580 RETURN
5000 REM CIRCLE
5010 INPUT "ENTER DIAMETER OF CI
RCLE  ";C
5020 CIRCLE XCORD,YCORD,C
5030 GO SUB 9000
5040 RETURN
5500 REM ALTER THE BORDER
5510 INPUT "ENTER BORDER COLOUR
(0-7)  ";BCOL
5520 GO SUB 9000
5540 RETURN
6000 REM ALTER INK COLOUR
6010 INPUT "ENTER NEW INK COLOUR
 (0-8)  ";ICOL
6015 INK ICOL
6020 GO SUB 9000
6030 RETURN
```

```
6500 REM CHANGE PAPER
6510 INPUT "ENTER NEW PAPER COLO
UR (0-8)  ";PCOL
6520 PAPER PCOL
6530 GO SUB 9000
6540 RETURN
6700 REM LOAD A SCREEN
6710  INPUT "ENTER FILENAME ";N$
6720 INPUT "      "; FLASH 1;"T"
; FLASH 0;"APE  OR "; FLASH 1;"M
"; FLASH 0;"ICRODRIVE";T$
6730 IF T$="T" THEN  LOAD N$
6740 IF T$="M" THEN  LOAD *"M";1
;N$
6750 RETURN
6800 REM SAVE TO A FILE
6810 INPUT "WHAT FILE NAME  ";N$
6820 INPUT "      "; FLASH 1;"T"
; FLASH 0;"APE  OR "; FLASH 1;"M
"; FLASH 0;"ICRODRIVE";T$
6830 IF T$="T" THEN  SAVE N$
6840 IF T$="M" THEN  SAVE *"M";1
;N$
6850 RETURN
7000 REM SUBROUTINE TO READ
7010 REM DIRECTION OF JOYSTICK
7015:
7020 REM DIRECTION RETURNED BY:
7030 REM UP,DOWN,LEFT,RIGHT,FIRE
7040 REM BEING 1 IF DIRECTION
7050 REM IS BEING PUSHED
7060 REM 0 IN VARIABLE IF NOT
7070 LET UP=0
7090 LET DOWN=0
7100 LET LEFT=0
7110 LET RIGHT=0
7120 LET FIRE=0
7130 LET X=255-IN 61438
7140 IF X>127 THEN  LET X=X-128
7150 IF X>63 THEN  LET X=X-64
7160 IF X>31 THEN  LET X=X-32
7170 IF X>15 THEN  LET X=X-16: L
ET LEFT=1
```

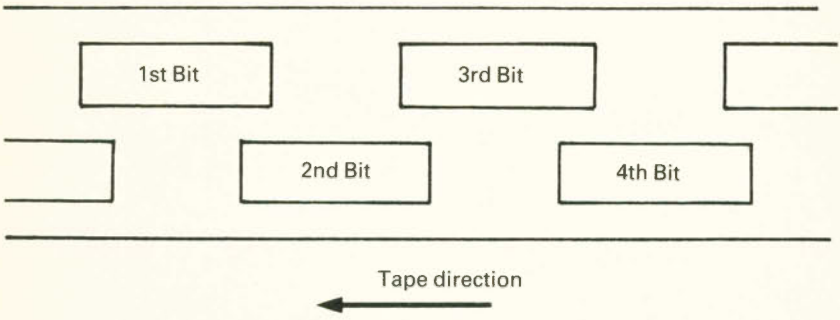
```
7180 IF X>7 THEN  LET X=X-8: LET
 RIGHT=1
7190 IF X>3 THEN  LET X=X-4: LET
 DOWN=1
7200 IF X>1 THEN  LET X=X-2: LET
 UP=1
7210 IF X=1 THEN  LET FIRE=1
7220 RETURN
9000 REM WIPE WRONG COLOUR FROM
9010 REM BOTTOM OF SCREEN
9020 BORDER BCOL
9030 PRINT #0;AT 0,0; INK BCOL;"

"
9040 RETURN
```

# Appendix A
# Microdrive cartridge tape format

The Microdrive stores data in a complex pattern of blocks of code upon the tape cartridge, independent of whether it is pure data or a program file. The system uses two tracks with data bits being stored alternately on overlapping strips of each track. This means that each data bit can physically occupy a longer length of tape than if a single track were used for a given tape speed thereby increasing the reliability of the device. Thus the read/write head of the Microdrive has to have double tracks, rather similar to the record/replay head of a stereophonic cassette player. This makes the Microdrive more complex, but the gain in operating speed is considerable and worthwhile.

The following diagram shows how the data bits are organised on the tape:




Each block of data starts with an identification block of 12 bytes. This consists of ten '0' bytes and two '255' bytes and allows the computer to identify the start of a block of data with high accuracy.

The operation of FORMATting divides the tape into sectors, each of which is divided up in the following way:

**a) Header Block:**

12 bytes of identification

A sector header of 15 bytes, divided into:
- A flag byte
- A sector number byte
- Two unused bytes
- Ten bytes for the cartridge name
- a checksum byte.

**b) A gap of blank tape**

**c) Data block:**

12 bytes of identification

A record identifier of 15 bytes, divided into:
- A flag byte
- A record number byte
- Two bytes for the record length
- Ten bytes for the file name
- A checksum byte

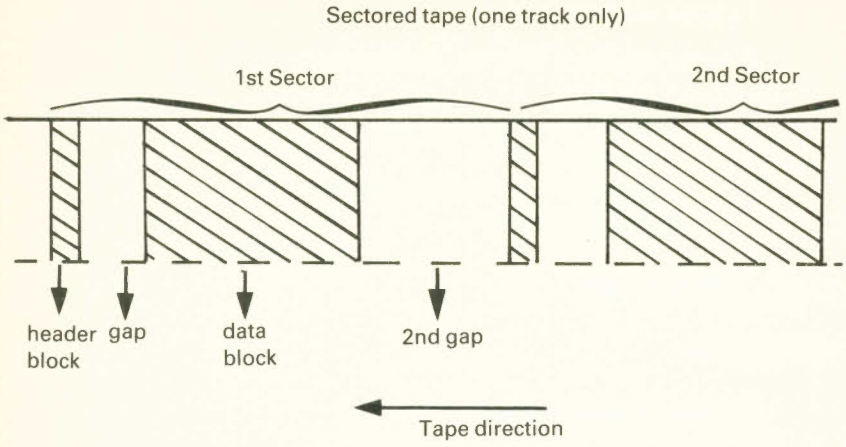
A record
- A 512 byte data area
- A single checksum byte

**d) Another gap of blank tape**
Then the start of the next sector and so on.

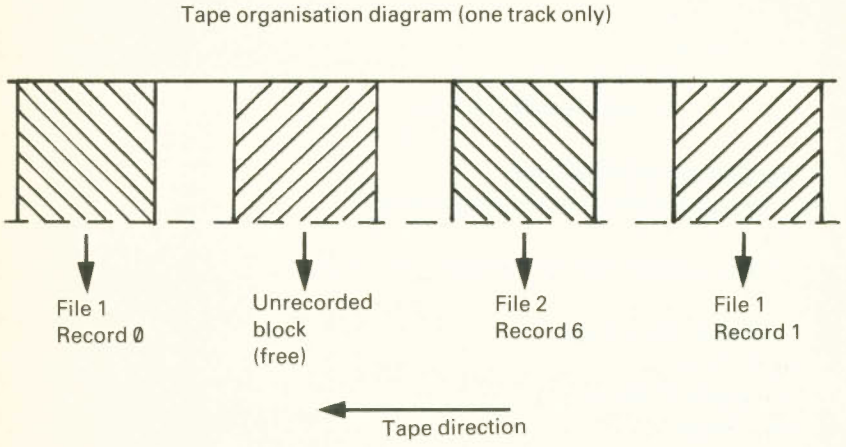
The approximate timing for each of the above are:

Header block 1.25 ms
Gap 3.75 ms
Data block 25.00 ms
Gap 7.00 ms

The diagram below shows how the tape sectors are organised:




Not all of the individual sectors will necessarily be in use at any one time, unless the cartridge is full. The diagram below shows how a tape track may be organised:



# Appendix B
# Microdrive memory map

In the previous Appendix you will have seen that the Microdrive organises the data that is stored on the tape within the cartridge. Within the system the Microdrive has to know which sectors of the cartridge are free to write to. To do this the system sets up what is called a memory map within RAM, and in this case it is called the Microdrive memory map. The program below allows you to display on screen the contents of the Microdrive Memory Map for the current cartridge and how it changes as you store more data on that cartridge. You will find that you will never need to manipulate the memory map contents, but it is interesting to examine it.

Start by FORMATting a blank cartridge, using FORMAT"m";1;"map

Use NEW to clear any existing maps and channels.

```
10 OPEN #4;"m";1;"newfile"
20 FOR a=23792 TO 23823
25 REM this is the map area
for newfile
30 LET n=PEEK a
35 REM this reads each byte
40 FOR b=1 TO 8
45 REM this means eight bits t
o a byte
50 PRINT n/2<>INT (n/2);
55 REM checks each bit as 0 or
 1
60 LET n=INT (n/2)
70 NEXT b
80 NEXT a
90 CLEAR #
```

On RUNning the routine listed above you will obtain a screen display similar to the following:

```
1000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000001000
0000000000000000000000000000000000
0000000000000000000000000000111111
1111111111111111111111111111111111
1111111111111111111111111111111111
```

In the above memory map for the Microdrive cartridge that is currently in use, the sectors are display in ascending order from sector 0 to sector 255. For this cartridge sector 0 is non-existent as are 72 and 73, presumably because of the splice necessary to produce the continuous loop of tape. Sectors 193–255 are also non-existent. This means that there are 190 sectors each of 512 bytes that can be used, or 95K of storage on this cartridge.

The following display was obtained after a 6.5K program was stored on this cartridge:

```
1000000000000000010000000000000000
0000000000000000010000000000000000
0000000000000000000000000000000000
0100000000000000000000000000001000
0000000000000000010000000000000000
0000000000000000010000000000111111
1111111111111111111111111111111111
1111111111111111111111111111111111
```

Note that the sectors are not adjacent to each other. This is due to the 'overlapping' of the sectors from the two tracks on the tape: there is insufficient time for the Microdrive system to fetch the next 512 bytes of data from the channel area and put them into the next sector, so a sector further along the tape will have to be used.

# Appendix C
# The Spectrum edge connector

The expansion port of the ZX Spectrum can be adapted to connect to many other devices than Interfaces 1 and 2, or the ZX Printer. For those willing to experiment, the connections for the edge connector are given below. The convention for the list is to position the Spectrum so as to look at the back of the machine with the keyboard uppermost, then from right to left the connections are numbered 1 to 28 (5 must be the slot).

| Upper row | Number | Lower row |
|---|---|---|
| A15 | 1 | A14 |
| A13 | 2 | A12 |
| D7 | 3 | +5 volts |
| NC | 4 | +9 volts |
| Slot | 5 | Slot |
| D0 | 6 | 0 volts |
| D1 | 7 | 0 volts |
| D2 | 8 | CLK |
| D6 | 9 | A0 |
| D5 | 10 | A1 |
| D3 | 11 | A2 |
| D4 | 12 | A3 |
| INT | 13 | IORQGE |
| NMI | 14 | 0 volts |
| HALT | 15 | VIDEO |
| MREQ | 16 | y |
| IORQ | 17 | v |
| RD | 18 | u |
| WR | 19 | BUSRQ |
| −5 volts | 20 | RESET |
| WAIT | 21 | A7 |
| +12 volts | 22 | A6 |
| +12 volts unreg | 23 | A5 |
| MI | 24 | A4 |
| RFSH | 25 | ROMCS |
| A8 | 26 | BUSACK |
| A10 | 27 | A9 |
| NC | 28 | A11 |

NC=No connection

The Video y,v,u are not normally connected but may be used with a monitor display unit to obtain better graphics resolution.

# Appendix D
# The Spectrum: Simple faults with hardware

The following is a simple self-help section containing the first things to try before you have to resort to sending the machine back to Sinclair for repair.

**No display**

(1) check the connections from the Spectrum to the television.
(2) is everything switched on? Check that there is power to the TV, to the Spectrum, and Spectrum power connector. Is the correct channel selected.
(3) Press and hold down the ENTER key and listen for faint clicks from the computer. If nothing is heard then recheck the power leads.
(4) the TV may be slightly detuned. It should be on or near channel 36.
(5) if there is still no success, start checking the mains fuses to the TV and Spectrum. Do not replace the fuse with a higher rated one.
(6) if you have or can borrow a voltmeter, check the +9 volt line on the edge connector (bottom edge on the right of the slot). If the voltage is zero, recheck the power connections.
(7) always check the TV on a broadcast channel to ensure it is working properly.
(8) if all else fails write to the Sinclair repairs department, explaining the fault and enclosing all of the computer system.

**No colour on the display**

(1) make sure that you have set up the colours; try the INK, BORDER or PAPER commands.
(2) check that the TV's colour and tint controls are set correctly, using a broadcast channel.
(3) check again that the TV is correctly tuned to the Spectrum. This is essential to obtain the correct colours.
(4) if you are still having no joy, it is probably one of two things. Either the colour crystal in the Spectrum is off-frequency or the TV in use is one of the odd few that are incompatible with the Spectrum. Check the second reason by borrowing a different TV made by a

different manufacturer; if this doesn't correct the fault, pack the computer system off to Sinclair.

### Random graphics displayed

When all of the power connections are correct and a display of random graphics of assorted colours occurs with no copyright message on the lower screen on power up, and no external units have been connected (such as Interface 1), then it is definitely time to contact Sinclair. If the fault occurs with a non-standard peripheral connected it may be due to overloading of the power supply circuits, so disconnect the unit and try the test with the base unit only. The standard Sinclair power unit can support the computer plus Interface 1 & 2 and the ZX Printer so it should have enough capacity to work other units as long as some are disconnected first. Alternatively, supply the external unit from a separate power unit.

Long leads to external circuits can also cause problems. If it is impossible to shorten the lead then some form of buffer circuit will be needed.

It is not advisable to go delving into the Spectrum unless you are very adept with a fine soldering iron and, remember, opening the case will invalidate the guarantee.

### Persistant Interface 1 error messages.

It sometimes happens that seemingly illogical error messages crop up even if the Interface 1 is solidly screwed to the computer. This could be due to oxides or grease on the edge connector. It can usually be cured by dismounting the interface and reconnecting it (always do this with the power off). Again, if no joy then it's back to Sinclair with the whole computer system.

# Appendix E
# The extended BASIC commands

Interface 1 extends the BASIC commands of the Spectrum computer; the new commands are listed below in alphabetical order:

| | |
|---|---|
| CAT n | Gives a list of the files contained on the cartridge in the Microdrive identifed by the number 'n' in the range 1 to 8. The list is headed by the cartridge name that was entered when the cartridge was last FORMATed. The files are organised in alphabetic order (except files that start with the character '0', which are hidden files so don't forget their names). The list is followed by the total (in kilobytes) of remaining storage free for use. |
| CAT #a;b | This sends the catalogue of the cartridge in Microdrive 'b' to stream'a'. |
| CLOSE #stream | This disconnects any channel from the stream specified. If any data is left in the buffer it is either transmitted (to a printer or onto the network or RS-232) or recorded (on the Microdrive) before the channel is CLOSEd. |
| ERASE "m";c;"filename" | ERASEs the specified file from the cartridge in Microdrive 'c' |
| FORMAT "m";c;"heading" | Prepares a blank cartridge, or wipes an old cartridge, located in Microdrive 'c' for use. The name "heading" is assigned to the cartridge and will appear as a heading when the CAT# instruction is used. |

FORMAT "n";x

Sets up the network station number x.

FORMAT "t";y
FORMAT "b";y

Used to set the baud rate for the RS-232 interface to 'y' (this must be chosen to suit the remote device but also must be one of the Spectrum standard baud rates: 50, 110, 300, 600, 1200, 2400, 4800, 9600 or 19200)

INKEY$ #stream

When used with the network or the RS-232 interface, this instruction returns a single character as a string if a character is available, and returns a 'null' string ("") if no character is available from the stream.

INPUT #stream;variable

The stream must have been previously OPENed to an INPUT channel. This command will INPUT data from the specified stream, up to and including a carriage return (CHR$(13)), and store it in the variable.

LOAD *channel options

This instruction LOADS a program, data or code from the channel specified. Only channels "b","n" or "m" may be used. All the options that are available with LOAD can be used with LOAD *.

MERGE *channel options

As for LOAD * above except that it does not delete old program lines or variable values unless the same line numbers or variable names appear in the new program that is to be MERGEd with the one in memory.

MOVE source TO destination

The source and destination can be any of the stream numbers or channels. The computer MOVES the data until it finds an end of file marker. This can only happen if the source is either a Microdrive or the network. If the source or destination is a channel then it is OPENed before use and CLOSEd after use.

OPEN # stream;channel

This allows the specified channel and stream to be connected and permits BASIC to input or output to that channel. The specified stream must not be already OPEN.

| | |
|---|---|
| PRINT #stream . . . | This outputs a PRINT sequence (. . . .) to the specified stream. The stream must previously have been OPENed. |
| SAVE *channel options | This SAVEs any program, or code to the specified channel but only 'b','n' or 'm' may be used. All the options available with SAVE may be used with SAVE *. |
| VERIFY * channel options | As with LOAD above except that any data is compared with the data already in the computer memory, to check that a LOAD has been carried out successfully. |

# Appendix F
# The extended error message system

Having attached Interface 1 to your Spectrum your program may produce a whole new set of error reports that do not appear in the basic Sinclair manual. These reports are generated not by Spectrum BASIC, although the BASIC error check routine is carried out, but by the Interface 1 as a backup to the main routine in the computer. These reports are followed by the line number and statement at which the program stopped.

These reports are now listed in alphabetic order:

Code error

The code block that you have tried to LOAD is too large to be stored in the destination area.

Drive write protected

The system has recognised that the cartridge in the Microdrive specified is 'write' protected, it has had the tab removed from the side of the cartridge and is thus 'read' only. If you want to write to this cartridge then cover the tab area with a small piece of sticky tape.

File not found

Either you have tried to LOAD a file that does not exist (a typing error maybe, or the cartridge is the wrong one) or else part of the file cannot be found. This may happen if the file has not been CLOSEd properly or if the file was damaged, perhaps by leaving the cartridge in the Microdrive when you switched the power off or on.

Inva
lid device expression
This report occurs when you have specified a channel other than 's', 'k', 'p', 'm', 'n', 't' or 'b'. Alternatively, the same report will occur if the wrong delimiter has been used with channels 's', 'k' or 'p'. The delimiter should be a comma.

Invalid drive number

You can specify Microdrive numbers from 1 to 8, and if you have n (less than 8) Microdrives then you can only use "m";1;, "m";2; and so on up to "m";n;

Invalid name

Either you have specified a file name that is longer than ten characters, or an empty string.

Invalid station number

When using the network you specified a station number outside the range 0–63.

Invalid stream number

When specifying a stream number you have used a number outside the range 0 to 15.

MERGE error

The system will only allow you to MERGE programs: you have either tried to MERGE data or code, or have tried to MERGE a program with an autorun statement like SAVE . . . LINE . . .

Microdrive full

The Microdrive cartridge was either full or did not contain enough space to accommodate the file that you tried to SAVE. You can get round this by erasing unwanted files to recover enough free space to SAVE your file, but you must remember to ERASE the file that you tried to write. This must be

done because, as there was not enough space, the file could not be CLOSEd. Erasing this file will take about 30 seconds because the Microdrive checks the cartridge several times.

Microdrive not present

This can occur for one of four reasons:
(1) a Microdrive is not attached
(2) the Microdrive does not contain a cartridge
(3) the Microdrive contains a cartridge that has not been formatted
(4) a dirty edge connector may be generating the error.

Missing baud rate

You have missed out the baud rate on FORMATting an RS-232 port.

Missing drive number

The computer will complain if it cannot identify where you want to put some data.

Missing name

The computer will complain if you forget the filename.

Missing station number

This happens on the network when a station number is missed.

Program finished

Usually an editing error. The program was asked to execute a line beyond its last line, for example to GOTO a line number outside the program. It may also occur if RUN is used within a program.

Reading a write file

The file you have tried to read does not exist yet or is a file that was OPENed for writing but was not CLOSEd properly.

Stream already open

A stream may only be OPENed if it was CLOSEd properly the last time it was used.

Verification has failed

The file that was SAVEd does not agree with the file that is still in central memory.

Writing to a 'read' file

The file name that you have just used already exists. The existing stored file should be erased or you will have to use a new file name to save the file in memory.

Wrong file type

This occurs if you tried to:

(1) INPUT or MOVE a SAVEd program file.

(2) LOAD, VERIFY or MERGE a data file.

(3) LOAD a CODE or DATA file as a progrm or vice versa.

If you were using INPUT then try LOAD. Similarly if you were using LOAD then you should use CODE or DATA options or use INPUT.

# Appendix G
# The network channel

On OPENing a stream to the network, a channel is created in the CHANS area of the main memory of the Spectrum. This area is normally addressed by the IX register in the central processor unit. This area is 276 bytes long and contains a 255 byte buffer.

The area is organised as follows:

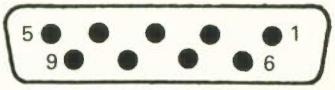
| Byte | Nomenclature | Contents |
|---|---|---|
| 0 | | Address 8 |
| 2 | | Address 8 |
| 4 | | 'N' |
| 5 | | Address of subroutine for output in the ROM |
| 7 | | Address of subroutine for input in the ROM |
| 9 | | Address 276 |
| 11 | NCIRIS | The destination station number |
| 12 | NCSELF | The home station number |
| 13 | NCNUMB | The block number |
| 15 | NCTYPE | The packet type code . . . 0 data, 1 End of file (EOF) |
| 16 | NCOBL | Number of bytes in the data block |
| 17 | NCDCS | The data block checksum |
| 18 | NCHCS | The header checksum |
| 19 | NCCUR | The position of the last character taken from the buffer |
| 20 | NCIBL | The total number of bytes in the input buffer |
| 21 | NCB | The 255 byte data buffer |

# Appendix H
# The RS-232 connections

The 9-pin 'D' line socket on the back of Interface 1 is the RS232 input/ output port and should not be confused with a joystick socket (if attempts are made to connect a joystick here then permanent damage may occur).

The RS-232 socket is wired as follows:

1. No connection
2. TX data (input)
3. RX data (output)
4. DTR (input) this should be high when ready
5. CTS (output) this should be high when ready
6. n.c.
7. Ground (pull down)
8. n.c.
9. +9v (pull up)




The following is a table of the pin connections for this socket:

(1) No Connection (NC)
(2) TX data
(3) RX data
(4) DTR
(5) CTS
(6) NC
(7) Ground (GND)
(8) NC
(9) +9 volts

The 'standard' for RS-232 connections is the 25-pin version of the same type of connector as above and the pin connections are also standardised as below:

( 2) TX data
( 3) RX data
( 5) CTS
( 6) +9 volts (normally DSR)
( 7) Ground
(20) DTR

# Appendix I
# An alternative to the Interface 1

The Interface 1 is not the only device that offers expansion and storage facilities to the Sinclair Spectrum. The Wafadrive from Rotronics offers the user of a Spectrum in one unit facilities that are similar to:

1 Interface 1
2 Microdrives
1 Centronics interface (Kempston etc.)

No networking facility is provided by the Wafadrive but for the single user a Centronics interface will probably prove more useful than a network that may not be used.

As with the Microdrive, a very thin continuous loop of tape is used as the storage medium. However, unlike the cartridges for the Microdrive the ones for the Wafadrive, called wafers, have not been especially designed for the Spectrum. They were originally designed, together with their drives, for use in business and industrial applications. This fact means that they are very robust, likely to prove highly reliable and should have a longer life and be more readily and cheaply available than Microdrive cartridges.

The wafers for the dual drive Wafadrive are supplied in three sizes: 16K, 64K and 128K. In use the Wafadrive is slightly slower than a Microdrive, one reason why different sizes of tape are sold. The 16K size would probably be used mostly for program development as it has a slowest access time of around 6 seconds. The 128K wafer has a slowest access time of about 45 seconds and therefore would probably be more suitable for data storage or even just for backup storage.

Even though LOADing and SAVEing are a little slower than with a Microdrive, getting a catalogue of a tape is much faster. As already mentioned, the Wafadrive has two drives; when the operating system is called in an area of memory is put aside for storage of the directory of each drive. Once the directory is LOADed into RAM the Wafadrive only has to check that the wafer hasn't been changed and then dump out the contents from RAM, which is a lot quicker than the same process on the Microdrive.

Another feature of the Wafadrive is that its operating system has to be loaded into RAM. This is not as difficult as it sounds. All you have to do is type:

NEW *

then you will be ready to use the Wafadrive. However, until you do this the Spectrum will behave as if the Wafadrive is not plugged in. This means that you can RUN all of your software without having to unplug the Wafadrive. Remember that with the Interface 1 present you may sometimes have to unscrew the two screws on the bottom of the interface and remove it from the Spectrum in order to be able to RUN your favourite games.

The syntax of the commands that are available for use with the Wafadrive are different from those used with the Interface 1. All the Interface 1 commands are there, (apart from the network ones), and in many cases they have been expanded. There are even some new ones, which we shall briefly look at here.

To LOAD a file called PROG1 from the Microdrive you would have to enter the command:

LOAD *"m";1;"PROG1"

the Wafadrive equivalent is

LOAD *"PROG1"

The reason that no drive specifier has been added to the instruction is that the Wafadrive will always assume you are using the default drive, unless you specify otherwise. The drives are labelled A and B and to change the default drive all that you need to do is type:

CAT #"b:"

The default drive will now be B.

As mentioned, you can also specify the drive number by typing the number together with the file name, e.g.

LOAD *"B:fred"

Streams and channels are also used on the Wafadrive, although the extended channels used with the Interface 1 have been changed:

channel R is the RS-232 input/output port
channel C is the Centronics port.

No channel specifier is required for use with the Wafadrives as the OPEN# command has been expanded to allow for them. With the Interface 1, if you

wished to direct an LPRINT to the printer you would type:

OPEN#3;"T" (or "B")

and you would then have to set the baud rate by typing:

FORMAT "T";1200

You can perform exactly the same function using the R channel, (there is no T channel so make sure you don't send any weird codes!) and format its speed. Alternatively you can use the Centronics port, which is probably much more useful as most printers are supplied with a Centronics interface. So, to LPRINT to the C channel you would type:

OPEN#*3,"C"

and all LPRINTing would be directed to where you had specified.

We mentioned that the syntax for OPENing a file to drive had been expanded. With the Interface 1 you would have to specify the channel you were writing to as M, thus:

OPEN #4;"M";1;"DATA1"

The syntax to do this on the Wafadrive is:

OPEN #*4,"A:DATA1"

This is probably a little more confusing than the Interface 1 version. Also the CLOSE# command should *always* be replaced with CLOSE#*

There now follows a list of all of the commands that can be used with the Wafadrive. This list could prove useful if you ever wish to convert programs between the Microdrive and the Wafadrive. Note that the only way of transferring programs between the two interfaces would be using cassette or the RS-232 port. You cannot have both a Wafadrive and an Interface 1 connected to your Spectrum at the same time.

| COMMAND | FUNCTION |
|---|---|
| NEW | clears operating system and program |
| NEW# | clears BASIC program |
| NEW* | initialises operating system |
| LOAD* | LOADs first program |
| LOAD*"n:file" | LOAD specified file (n is optional) |
| LOAD*"n:file",start,length | LOAD machine code |
| SAVE*"n:file" | SAVE file |

| | |
|---|---|
| SAVE*"n:file"LINE s | SAVE file with autorun line no. |
| SAVE*"n:file",st,len,runadd | SAVE code with autorun address |
| SAVE# | SAVE with replace, use above parameters. |
| VERIFY*"n:file" | VERIFY program/machine code |
| MERGE*"n:file" | MERGE file |
| ERASE*"n:file" | ERASE file |
| MOVE*"n:file1" TO "n:file2" | copy file1 to file2 |
| CAT#"n:" | set default drive |
| CAT*"n:" | display directory |
| FORMAT*"n:name" | FORMAT wafer |
| FORMAT*"R";baud | set baud rate (default 1200) |
| OPEN#*stream;channel | OPEN stream to channel |
| CLOSE#*stream | CLOSE stream |
| CLEAR* | CLOSE all streams |
| CLS* | set screen to default colours |

Robert Erskine & Humphrey Walwyn with Paul Stanley and Michael Bews
**Sixty Programs for the Sinclair ZX Spectrum** £5.95

**Sixty Programs for the BBC Micro** £5.95

**Sixty Programs for the Dragon 32** £5.95

**Sixty Programs for the Oric 1** £5.95

**Sixty Programs for the Atari** £5.95

**Sixty Programs for the Commodore 64** £5.95

**Sixty Programs for the Vic 20** £5.95

**Sixty Programs for the Electron** £5.95

Ian Adamson
**The Companion to the Oric 1** £5.95

Geoff Wheelwright
**The Companion to the BBC Micro** £4.95

Keith Bowden
**The Companion to the Commodore 64** £5.95

Jeremiah Jones and Geoff Wheelwright
**The Companion to the Electron** £5.95

Jean Frost
**Instant Arcade Games for the Sinclair ZX Spectrum** £3.95

**Instant Arcade Games for the BBC Micro** £3.95

**Instant Arcade Games for the Dragon 32** £3.95

**Instant Arcade Games for the Electron** £3.95

J. J. Clessa
**Micropuzzles** £2.95

Ian Scales
**Spectrum Peripherals Guide** £4.95

Tony Takoushi
**Best Software Guide: Vic 20/Commodore 64** £3.95

**Best Software Guide: Spectrum Games** £3.95

Jeff Aughton
**Invaluable Utilities for your BBC Micro** £5.95

All these books are available at your local bookshop or newsagent, or can be ordered direct from the publisher. Indicate the number of copies required and fill in the form below.

Name_____
(Block letters please)

Address_____

_____

Send to Pan Books (CS Department), PO Box 40, Basingstoke, Hants: Please enclose remittance to the value of the cover price plus: 35p for the first book plus 15p per copy for each additional book ordered to a maximum charge of £1.25 to cover postage and package. Applicable only in the UK.

While every effort is made to keep prices low, it is sometimes necessary to increase prices at short notice. Pan Books reserve the right to show on covers and charge new retail prices which may differ from those advertised in the text or elsewhere.

## THE COMPLETE REFERENCE COMPANION FOR MICRODRIVE USERS!

The advent of the ZX Microdrive, Interface 1 and Interface 2 has turned a toy into a tool, making the Spectrum the core of a powerful and flexible computer system. With the mass storage capacity and fast access of the Microdrive, and the control and communications facilities of Interface 1 and 2, the potential is astonishing.

The **Companion** sets out to explain the function of all parts of the system, how they interact with each other, with peripherals, and via the local networking facility. The standard RS232 interface allows communication not only with standard printers, but with other computers, directly or using modems, and all these possibilities are explained and explored.

The programming operations are carefully detailed and illustrated with applications programs, and the routines available in the shadow ROM of Interface 1 are used to incorporate simpler Microdrive commands and extensions to BASIC.

Any Spectrum owner wanting to explore the world opened up by the Microdrive and Interfaces will find this book an essential tool and reference.

ISBN 0-330-28662-5

U.K. **£5.95**

90000

9 780330 286626