The COMPLETE

PART 5

SPECTRUM

All you want to know about the world's best-selling computer

A Database Publication

Graphics hardware options

Communications - how to get started

Structured programming made simple

Sinclair Basic - the inside story

+graphics, Basic and Z80 code

BARGAIN OFFERS INSIDE

SAVE £££s ON MANY MORE SPECTRUM PRODUCTS

In six monthly parts

£1.50

3D GANGSTERS "HIT" YOUR SCREEN

SPECTRUM 48K
£7·95
SPECTRUM 48K

Interactive 'bubble' speech adds to the drama of this true life detective movie

Don't miss this picture - it speaks for itself!

AMSTRAD
£8·95
AMSTRAD

In Part Five...

The COMPLETE SPECTRUM
All you want to know about the world's best-selling computer

is published by
Database Publications Ltd,
Europa House, 68 Chester Road,
Hazel Grove, Stockport SK7 5NY.
Tel: 061-456 8383

News trade distribution by:
Europress Sales & Distribution Ltd,
Unit 1, Burgess Road, Ivyhouse Lane,
Hastings, East Sussex TN35 4NR.
Tel: 0424 430422.

READ all about DATA...

Fifth in the series that aims to explain the elements of Basic for absolute beginners

THERE's a lot to do this time, so let's get straight to work.

Have a look at Program I which assigns values to three numeric variables and then displays them.

```
10 REM Program I
20 LET first=1
30 LET second=2
40 LET third=3
50 PRINT first, second, third
```

Program I

Now compare that rather long-winded way of doing things with Program II. You'll find READ and DATA above the A and D keys respectively.

```
10 REM Program II
20 READ first,second,third
30 PRINT first,second,third
40 DATA 1,2,3
```

Program II

This does exactly the same thing, giving values to *first, second,* and *third* but it does it using two new keywords, READ and DATA. Values have been given to variables without a LET or INPUT in sight. How is this done?

In many ways the READ command can be likened to INPUT. If you come across a line like:

```
INPUT first,second,third
```

you know the Spectrum will look to the keyboard for the values to be stored in *first, second* and *third.*

When the Spectrum comes across the line:

```
20 READ first,second,third
```

it knows it is to get values for the variables following READ. This time, however, it doesn't go to the keyboard but to the program itself. The values for the variables are tucked away in a program line.

There's nothing all that odd about this. After all, when Program I used LET, the values were tucked away in three program lines. READ does the same thing, but saves on the LETs.

The values for the variables are held in the data line, line 40. Here the keyword DATA tells the Spectrum that what follows is data that at some point in the program is to be read into variables by READ. The DATA itself doesn't do anything, it's just there to tell the Spectrum where to look for values when it comes across a READ statement.

As you'll see from line 40, the data items (1, 2 and 3) are separated by commas. The READ takes the data one item at a time in the order in which they follow the first DATA. So in this case the 1 is taken, then the 2 then the 3. The Spectrum remembers where it is by means of a data pointer – a sort of electronic address book. As a Basic programmer you don't see this pointer, but it's there keeping track of all the READs and DATAs. At the beginning of a program it points at the first item after the first occurrence of the keyword DATA in a program. When a READ grabs this value, the pointer is moved so it points to the next item in the line and so on until there's no more data.

Because this elusive pointer is keeping tabs on the data, you can split the item in the data list over several lines. Try modifying Program II with:

```
40 DATA 1
50 DATA 2,3
```

and:

```
40 DATA 1
50 DATA 2
60 DATA 3
```

and you'll see that it makes no difference, the Spectrum is up to it. As you use READ and DATA in your own programs, you'll find that it pays to use short data lines. This helps when it comes to spotting any errors you may have made either programming or typing.

READ about strings

As you might guess, READ works with strings as well as with numerics. Program III shows how it's done.

```
10 REM Program III
20 READ a$,b$,c$
30 PRINT a$,b$,c$
40 DATA "first","second","third"
```

Program III

The actual mechanics of the program are similar to Program II, only now we're reading characters into string variables rather than numbers into numeric variables. Notice that the strings in the data line have to be surrounded by inverted commas. Try:

```
40 DATA first,second,third
```

and see what happens.

From what was said about the pointer earlier, it might have occurred to you that the data line doesn't have to come at the end of a program. The pointer will track it down wherever it is. This is true as you'll see if

you replace line 40 of Program III with:

```
15 DATA "first","second","third"
```

The program works just as well. However there is a strong case for putting your data items at the end of programs where you can find them easily. Alternatively, in longer programs, you can keep them near to the program lines that use them. Either way, it's best to have some logic behind where you store your data in a program. The Spectrum's pointer can keep track of it but you might not be able to if you have to alter or correct the listing. A labelling REM doesn't go amiss, either.

Mixed data and expressions

If you look at the data lines in Programs II and III, you'll see that they consist of lists of the same data types. In Program II, there are three numbers, in Program III there are three strings. Does the data have to be like this, of a uniform type? Can you mix strings and numbers in a data line? The answer is yes, as Program IV demonstrates.

```
10 REM Program IV
20 READ number,a$,figure
30 PRINT number,a$,figure
40 DATA 1,"a string",2
```

Program IV

Here the data items consist of a number, **1,** a string **a string,** and a second number, **2.** If you look at the READ line, you'll see that the variables these values are read into consist of a numeric, *number,* a string, *a$,* and another numeric, *figure.* As the combination of strings and numbers in the data items corresponds with the variables awaiting them in the READ line the program works correctly.

So, you can mix up strings and numbers but beware. The receiving variables have to be exactly right or else havoc ensues as you'll see when we come to some of the problems that can arise when using READ and DATA.

```
10 REM Program V
20 LET a=1
30 LET b$="a string"
40 READ number,c$,expression
50 PRINT number,c$,expression
60 DATA 1,b$,a*5
```

Program V

It's not just numbers and strings that can be used after a DATA. There can be variables and expressions (sums) as Program V shows.

Here lines 20 and 30 are used to set up two variables, *a* and *b$*. Notice that these are used in the data line. *b$* appears as itself, but *a* appears in an expression, a*5.

The READ of line 40 takes all this in its stride, giving *c$* the string it finds in *b$* and working out a*5 and storing the result in *expression.*

DATA on READing problems

Having waxed lyrical about the joys of using READ and DATA we now come to some of the problems that lie in wait for the unwary. READ and DATA are extremely simple to use, provided they are used properly. Sadly it's also simple to make mistakes using them. But don't make a mistake when you enter and run Program VI.

```
10 REM Program VI
20 READ a,b,c,d
30 PRINT a,b,c,d
40 DATA 1,1.5,2,3.5
```

Program VI

No prizes for understanding what it does, READing four values into four numeric variables and displaying them. However, suppose that by error you'd entered line 40 as:

```
40 DATA 1,1,5,2,3.5
```

If you can't see the difference, it's that the decimal point in 1.5 has been made a comma giving 1,5. We have two data items for the price of one. Can you guess what effect this has on the program? Try to work out what the output will be before you run it.

The answer is that you now get:

```
1    1
5    2
```

instead of:

```
1    1.5
2    3.5
```

as before.

This is because after the mistake the first four items in the data list are 1,1,5,2. The pointer doesn't know

that the comma between 1 and 5 should be a decimal point. It just takes the first four values and puts them in the variables after the READ. The result of the typing error is that the wrong values are stored in the variables. In this case only the display is changed but imagine what would happen if these numbers were being used later on in the program for a series of calculations! The errors would spread throughout the whole listing. And all because of a comma instead of a full stop.

Meanwhile, what about the data pointer? At the moment it's pointing at 3.5. As it is, Program VI doesn't use this value. However, suppose that later on in the listing there was another READ with its own set of data such as:

```
200 DATA 1,2,3,4
```

Now instead of the second READ taking its first item of data from line 200, as intended, it will look at the data pointer and take the 3.5 from line 40. The pointer will move to the 1 of line 200. In other words the data will be out of step by one item.

The trouble with all this is that the program still works, only the results are wrong. There are two conclusions to be drawn. Firstly make sure that everything after a DATA is typed correctly. Second, check the results of your programs with known answers. If the program works but gives the wrong results, have a look at the data lines.

Having looked at the problems when an inadvertant comma causes an additional item in a data list, let's look at the opposite. Suppose that the READ has more variables after it than there are items in the data list. Alter Program VI with:

```
40 DATA 1,1.5,2
```

Now there are only three numbers but there are still four variables looking for values. The Spectrum has no problems in giving *a*, *b*, and *c*, the values 1, 1.5 and 2 respectively. However when it goes to the data line for a value for *d* the cupboard is bare. The Spectrum realises what's wrong and tells you, giving the error message:

```
E Out of DATA, 20:1
```

This is an improvement on the previous case where the program worked, only badly. Now you're told that there's not enough data. The trouble is that it points out line 20 as the culprit, not line 40. This is always the case, the message pointing to the READ rather than the data line. It can cause confusion until you're wise to it. After all, line 20 is correct, it's line 40 that's wrong. The lesson is, if you have a message pointing to a READ line and can't find anything wrong, have a good look at the data lines that the READ is trying to work on.

There's yet another thing that can go wrong with READ and DATA if they're used carelessly. We've shown that the data lines can hold both strings and numbers. However, the type of the data must correspond to the type of the variables after the READ. Try altering our long-suffering Program VI with:

```
40 DATA "one",1.5,"two",3.5
```

and when you run it you'll be rewarded with:

```
C Nonsense in BASIC, 20:1
```

for your pains.

The problem is caused when the micro tries to store the string **one** in the numeric variable *a*. It can't do it, hence the message. It's the same if you try to take a number from a data list and store it in a string variable. The Spectrum doesn't like it and the program halts. You can remedy it with:

```
20 PRINT a$,b,c$,d
30 DATA "one",1.5,"two",3.5
```

which produces:

```
one     1.5
two     3.5
```

To avoid this, remember – strings are strings and numbers are numbers and where READ is involved the twain never meet.

And as the final entry in this catalogue of woe have a look at Program VII.

```
10 REM Program VII
20 READ a$,b$,c$
30 PRINT a$,b$,c$
40 DATA "one","two","three"
```

Program VII

Be careful with the punctuation of line 40. It's all too easy to mix up commas and quotes and end up with a line like:

```
40 DATA "one,""two","three"
```

If this happens the program crashes with the message:

```
E Out of DATA, 20:1
```

It's not surprising as there are now only two strings in the data line, not three as the READ expects. You'll see what's happened if you tell the micro to:

```
PRINT a$
```

when:

```
one,"two
```

is displayed.

Again, be careful when you type in data lines. They have to be exact.

READ about DATA in action

At this stage in your programming career READ and DATA may not seem all that useful. Most of the earlier programs could be done just as well with LET or INPUT. However as you get more experienced and start using your knowledge of Basic to write your own programs you'll see how useful they are. Have a look at Program VIII.

```
10 REM Program VIII
20 LET total=0
30 FOR n=1 TO 5
40 READ number
50 LET total=total+number
60 NEXT n
70 PRINT "The total is";total
80 DATA 1,7,65,n,4
```

Program VIII

This uses the variable *total* to keep a running total of the numbers in the data line. Notice that the loop control variable *n* is used in the data line. As it's the fourth item in the line, it has a value of 4 when it's read into *number*. The result is the Spectrum tells you that:

```
The total is 81
```

Position	Name	Time	Age
1	PETE	20	35
2	MIKE	21	35
3	ALAN	25	55
4	KEVIN	27	20
5	ROLAND	30	27
6	ROBIN	33	24

Figure I: Runners, times and age

Try swapping *n* with one of the other items in the list and see if you can understand what happens.

So far our invisible data pointer has been content to work its way along a data list item by item. However it often happens that we want to use the same item of data over and over again. We could of course have lines like:

```
100 DATA 1,2,3,4,5
110 DATA 1,2,3,4,5
120 DATA 1,2,3,4,5
```

but it seems a waste of typing as they're practically the same. It would be nice if we could just use the data line over and over as needed.

There is a way to do this – the keyword RESTORE found above the S key. All this does is send the data pointer back to the first item after the first DATA in the program. Using this we could just have our one line:

```
100 DATA 1,2,3,4,5
```

and use it repeatedly by having a RESTORE send the pointer back to 1 after the program has read all five items in the list.

Program IX uses RESTORE in a rather sneaky fashion to add up the numbers between 1 and whatever whole number the user desires.

```
10 REM Program IX
20 PRINT "Sum from 1 to ?"
30 INPUT number
40 LET total=0
50 FOR n=1 TO number
60 READ control
70 LET total=total+control
80 RESTORE
90 NEXT n
100 PRINT total
110 DATA n
```

Program IX

The sneaky part is the use of the loop control variable *n* in the data line. This is constantly changing in value as the loop cycles. Each time round the loop the RESTORE of line 80 sees to it that the data pointer is back pointing at *n*. The result is that each time the program READs the value of *n* into *control*, *n* has increased in value by one.

As we've seen, RESTORE sets the data pointer back to the first item after the first DATA in the program. Sometimes however we want the pointer to point to another set of data items. This can be achieved by putting the line number of the required DATA after the RESTORE. Hence:

```
RESTORE 1000
```

will have the data pointer set to the first item after the DATA of line 1000.

The elements of arrays

Have a look at Figure I which sums up the results of a race. It gives the first six runners, their names, times and ages. Now suppose you wanted to use the time values in a program. You could have a set of numeric variables such as:

```
LET PetesTime=20
LET MikesTime=21
```

and so on. The trouble is that they don't reflect the information in the table. After all the times are in an order and this is lost. A better way is:

```
LET firstTime=20
LET secondTime=21
```

and so on. Here you know which was the first time, which was the second and so on. An even better way is to use an array.

An array is just a set of variables sharing a common single-letter name or root. They are distinguished from each other by a number in brackets at the end of the root known as a subscript. To create an array, we use the keyword DIM, found on the D key. So to set up the array *n()* we use:

```
DIM n(10)
```

What this does is to create 10 elements of the array ranging from *n(1)* to *n(10)*. At first each element of the array is 0 as:

```
PRINT n(1)
```

shows. Figure II shows the 10 variables created and their contents. Notice how the array has an implicit order and also how we can get at or "access" any of the elements by changing the value of the subscript.

We can alter the value of an element using LET just

n()	Subscript	1	2	3	4	5	6	7	8	9	10
	Value	7	10	17	0	0	0	0	0	0	0

Figure II: The array n( ) after DIMn(10)

Element	n(1)	n(2)	n(3)	n(4)	n(5)	n(6)	n(7)	n(8)	n(9)	n(10)
Content	0	0	0	0	0	0	0	0	0	0

Figure III: New contents for n()

as normal. So:

```
LET n(1)=7
LET n(2)=10
```

stores 7 in the first element of the array *n()* and 10 in the second. Now we can add the two together and store the result in the third element of the array with:

```
LET n(3)=n(1)+n(2)
```

Figure III shows the results of the addition which you can check with:

```
PRINT n(3)
```

Remember that our DIM only reserved memory space for 10 elements. If you do something like:

```
PRINT n(11)
```

you'll be rewarded with the message:

```
3 Subscript wrong, 0:1
```

Arrays in action

Program X shows an array in action, holding the times of the first six runners from Figure I.

```
10 REM Program X
20 DIM t(6)
30 FOR n=1 TO 6
40 READ t(n)
50 NEXT n
60 FOR m=6 TO 1 STEP -1
70 PRINT t(m)
80 NEXT m
90 DATA 20,21,25
100 DATA 27,30,33
```

Program X

Line 20 "sets up" the array, *t()* setting aside memory space for six numeric variables, *t(1)*, *t(2)*, up to *t(6)*, giving each the initial value 0. The program then enters the FOR ... NEXT loop of lines 30 to 50.

It's in this loop that the fact that the subscript of an element can be a variable comes into play. Line 40 READs a value from the data into the array element *t(n)*. Now *n* is the control variable of the loop, so as the loop cycles successive values from the data line are stored in *t(1)*, *t(2)*, ... up to *t(6)*. Figure IV shows what each element of the array holds once the first loop has finished. Notice how the structure of the array reflects

the order of the times. We know that the first element holds Pete's time, the second Mike's and so on.

Once we've got the data in an array we can manipulate it as we wish. Program X just prints the times out in reverse order using another loop. Can you alter it so that it works out the average time?

Figure I also gives us information about the ages of the runners. This, too, can be stored in an array as Program XI shows.

```
10 REM Program XI
20 DIM a(6)
30 FOR n=1 TO 6
40 READ a(n)
50 NEXT n
60 FOR m=2 TO 6 STEP 2
70 PRINT a(m)
80 NEXT m
90 DATA 35,35,55
100 DATA 20,27,24
```

Program XI

The first FOR ... NEXT loop puts the values into the array elements while the second displays the ages of the runners in the even positions. Again notice how the array order reflects the order of the runners. Figure V shows the values of the elements of *a()*.

If you look closely at Figures IV and V you'll see that they are very similar. *a(1)* refers to Pete's age, *t(1)* to Pete's time. Similarly *a(3)* refers to Alan's age, *t(3)* to his time. As you can see, in each array the same subscript points to information about the same person. If the subscript is 5 it's talking about Roland, if it's 6 it's pointing to Robin. Cases like these where two arrays are in the same order are called parallel arrays and the subscript is referred to as a pointer. Figure VI shows the two arrays in parallel.

● ***Have a good look at them, we'll be coming back to them next time as we deal with two dimensional arrays.***

t()	Subscript	1	2	3	4	5	6
	Value	20	21	25	27	30	33

Figure IV: Times held in t()

a()	Subscript	1	2	3	4	5	6
	Value	35	35	55	20	27	24

Figure V: Ages held in a()

Subscript		1	2	3	4	5	6
Value	t()	20	21	25	27	30	33
	a()	35	35	55	20	27	24

Figure VI: Parallel arrays

Communications

Log-on to the exciting world of modems, bulletin boards and electronic mailboxes

TO get into computer communications you need a few pieces of hardware and software for your computer, a telephone, and one more essential item – a computer with something to communicate at the other end of the line.

British Telecom runs a number of services for micro users to connect to over the telephone network, but the most popular and most interesting to home computer users is Prestel.

Prestel

Prestel is a huge database which runs simultaneously on a number of mainframe computers operated by BT. These large computers can each handle hundreds of individual callers at once, and they are themselves all linked together so that new information and messages placed on one of the Prestel computers is quickly shared with all the others.

These large computers are spread around the country and this, combined with special arrangements at telephone exchanges, means that you can get into the national Prestel system at local call rates from almost anywhere in the UK.

Originally used almost exclusively by travel agents – a tiny if vociferous fraction of the population – Prestel has blossomed forth as a general leisure interest pastime.

New ways to have fun

The microcomputing section, which includes Micronet, has been bringing in a constant stream of new subscribers who are looking for new ways to have fun with their computers. Micronet is just one part of Prestel, specialising in home computer subjects and software, and making up approximately 50,000 pages out of Prestel's 300,000.

Prestel is page-based, and you can find your way from page to page either by directly entering a page number (assuming you know it) or by swimming through a hierarchy of menus with single key presses. This last is much easier than it sounds and is the preferred method used by most newcomers to find their way around. It works like this. The first page might say:

Do you want –
1. To go to Micronet?
2. To go to Prestel Main Menu?
3. To look in your Mailbox?

Press 1, and a new menu comes up offering a choice of sections within Micronet. Usually you only need three or four menus to get exactly where you want to go, provided you have a good idea of how to get there.

The problem with menus is that it can take a lot of looking in the wrong places before you find what you are after, especially if you haven't been there before, as you are bound to take a few wrong turnings. Prestel says it is working on a keyword searching facility that will allow you to simply enter the subject you are interested in and get a choice of pages on which this subject appears.

Besides the microcomputing section, other services on Prestel offer information on the stock market (updated daily, hourly, or constantly, depending how much you pay), theatre guides, home banking, travel information and booking services (for the travel agents), and endless leisure pursuits.

Some of the pages are free, some have a small charge for each page, and some are available only to people who have paid an extra subscription to belong to a closed user group (CUG). Very few of the pages are provided by BT – almost all are rented by them to information providers (IPs), who can then advertise or provide information on a pay-by-the-page or subscription basis. Micronet is one of these information providers.

Electronic mailbox

One of the most interesting and useful services provided by Prestel is the electronic mailbox which allows you to send messages to any other Prestel subscriber for delivery the next time they call in. You can even send a message into the international telex system – some small businessmen are actually buying home computers and Prestel modems as a cheap way to get access to this.

Electronic mailboxes can only be sent to people who are themselves Prestel users and they will only get them if and when they call in to Prestel, but the telex service has been in use almost as long as the telegraph, and most large businesses already have telex in some form.

Micronet

This section of Prestel is where most computer buffs spend their time, and it is a good place to mix with users of other computers, like the BBC Micro, without any possibility of violence breaking out.

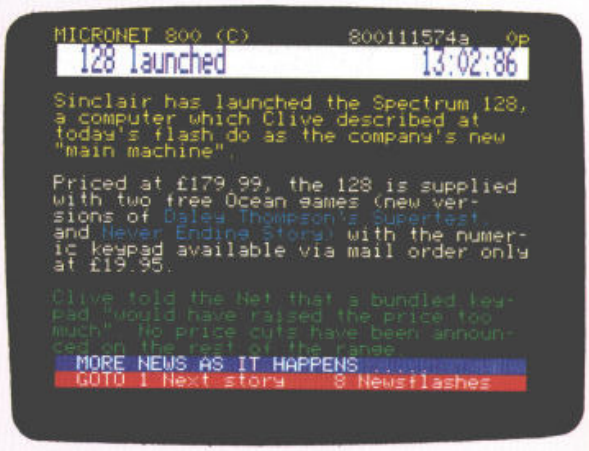
There is a conversational message service called Chatline which allows almost instant communication between users who are online at the same time. This is a lot like CB radio, with no privacy and a lot of mixing in by all and sundry, so it is great fun. Special celebrity nights bring people like Patrick Moore from The Sky at Night online to field questions from subscribers.

More formal public contact can be made via Micronet's bulletin board services, which allow you to advertise equipment to swap or for penpals with similar interests. Once you have made contact and exchanged mailbox numbers with the person you want to talk to, you can send each other private messages on the electronic mail service.

Micronet's own news service, with a bias towards the computing world, is days – sometimes weeks – ahead of any paper-based news media.

Their news has on occasion been so instant and outspoken as to evoke startled cries of outrage from

From downloadable software . . .

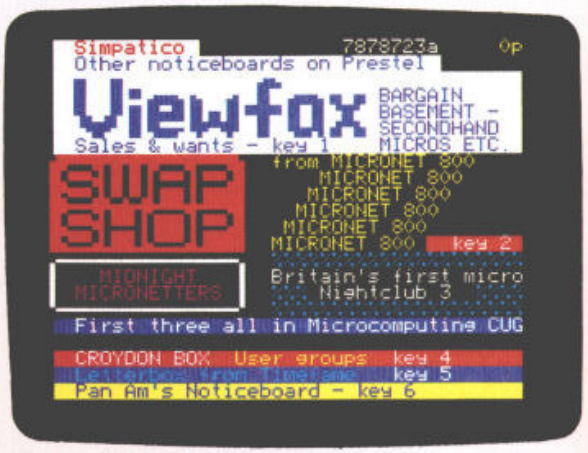


. . . to news . . .

those whose activities are reported. Even more controversy then results when an offending story is wiped from the archives, proving that electronic media are easy to censor!

The Micronet software pages provide downloadable software for many computers and the Spectrum is well supported. Prices vary from near the full shop price for well known names to very little indeed for programs sent in by amateurs. Send in your own programs for sale and you might break into software publishing.

Some of this software is actually free and you may suppose that it is worth roughly what you pay for it. This is bit more than the free label would lead you to believe, as you ought to figure in the telephone call charges and the cost of your Micronet subscription – but it is still pretty cheap, and the cost is reflected inevitably in the quality. Still, much of it is worth having and some is even better than certain stinkers that have been sold in the shops.



. . . swap shop . . .

Other systems

There are a number of other systems that can be accessed with the same equipment you use for Prestel. These are certain to be a great deal smaller than Prestel and Micronet but they have the advantage of being free. Some are run by amateurs and have only a single telephone line whereas others are run by local authorities or educational bodies. A few are actually run on Spectrums!

For the benefit of those who have equipment, here is a list of some of the systems you can try.

Aberdeen Itec	0224 641585
Basildon Itec	0268 22177
Brixton Itec	01-735 6153
C-View	0702 546373
Cardiff Itec	0222 464725
*CBBS SW	0392 53116
Communitel	01-968 7402
Cymrutel	0492 49194
Gnome at Home	01-888 8894
Hackney BB	01-985 3322
Metrotel	01-941 4285
Norview	0604 20441
Owltel	01-927 5820
RSGB	0707 52242
Stoke Itec	0782 265078
Swafax	0622 850440
Swafax II	0440 820002
System Aid	01-571 0026

** Not viewdata, but will work*

. . . clubs . . .

. . . and travel information

Save £2
-See Centre Pages

YOU can go for gold ...with the MICRO OLYMPICS

This is the package that broke all records! More than a game – it's a brilliantly written collection of ELEVEN great track and field events!

Ever imagined yourself as another Seb Coe? Then try to run against the world record holder at 1500 metres. And if that distance is too much for you then there's always the 100, 200, 400 and 800 metres to have a go at.

Not much good at running? Don't worry, MICRO OLYMPICS has many more challenges for you. Why not try your skill at the high jump or the long jump?

And if you can't beat the computer at running or jumping then you can always throw things around in frustration! The trouble is that it's just as hard to be a champion at the discus, the hammer or the javelin.

And the pole vault takes the event to new heights!

Yes, it's fast, furious fun, pitting yourself against the world's best times and distances on your micro.

You may not be another Steve Ovett or Alan Wells, but with practice you COULD become the Micro Olympics Champion!

Play Micro Olympics – and let your fingers do the running!

Send for it today

Please send me _____ copy/copies of Micro Olympics

☐ I enclose cheque made payable to Database Publications Ltd. for £ _____

Spectrum 48k £5.95

I wish to pay by
☐ Access ☐ Visa No. _____ Expiry date _____

Signed _____

Name _____

Address _____

_____

Post to: Micro Olympics offer, Database Publications, 68 Chester Road, Hazel Grove, Stockport SK7 5NY.

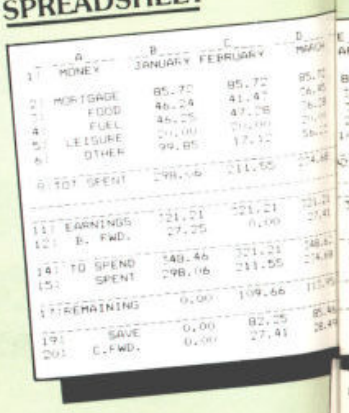
QUICK TO LEARN

THAT'S...

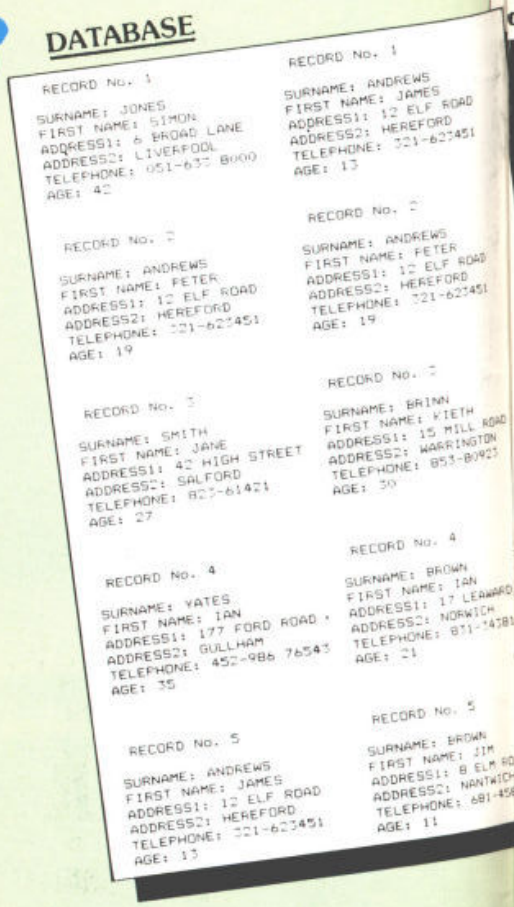
FINALIST BRITISH MICRO COMPUTING AWARDS 1985

MINI OFFICE

SPREADSHEET



DATABASE



JUST LOOK WHAT THIS PACKAGE CAN DO!

WORD PROCESSOR – Ideal for writing letters or reports! *Features:* Constant time display ● Constant word count (even shows words per minute) ● Normal or double-height text on screen or printout.

SPREADSHEET – Use your micro to manage your money! *Features:* Number display in rows and columns ● Continuous updating ● Update instantly reflected throughout spreadsheet ● Save results for future amendments.

GRAPHICS – Turn those numbers into an exciting visual display! *Features:* 3D bar chart ● Pie chart ● Graph.

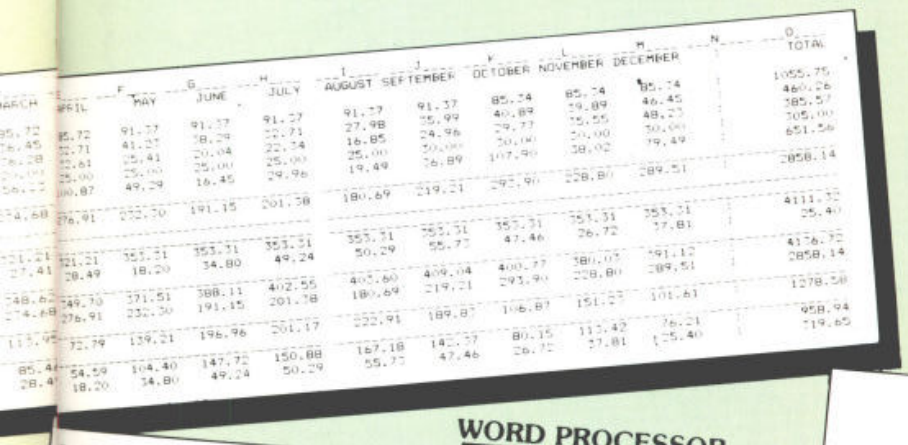
DATABASE – Use it like an office filing cabinet! *Features:* Retrieve files at a keystroke ● Sort ● Replace ● Save ● Print ● Search.

The screen dumps shown are produced on a BBC Micro with an Epson printer. Results from other computers and printers may vary slightly depending on equipment used.

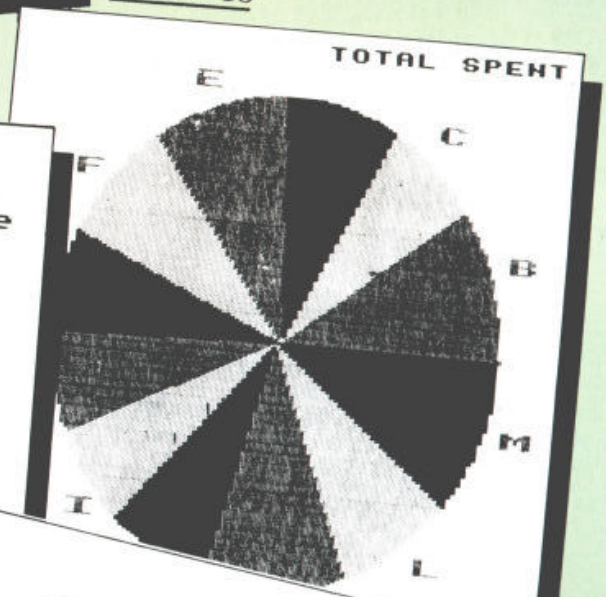
...and it's all at th price of just £5

RN, EASY TO USE

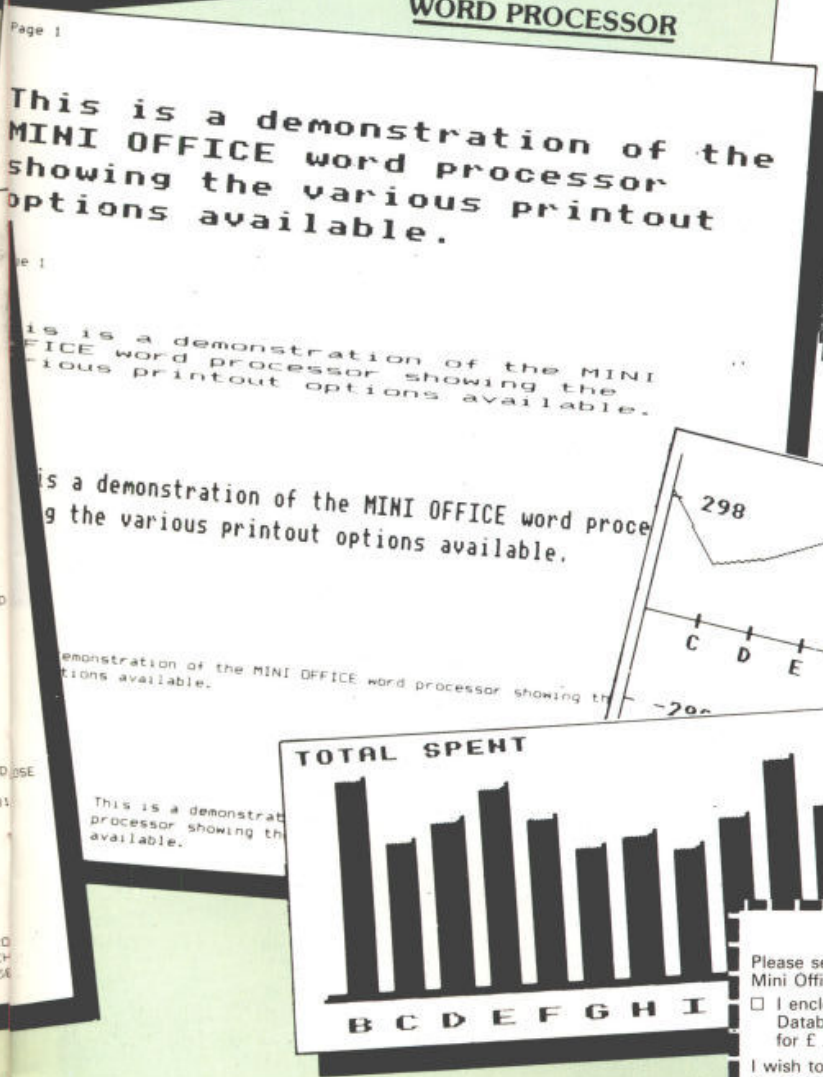
Spectrum 48k and Spectrum+

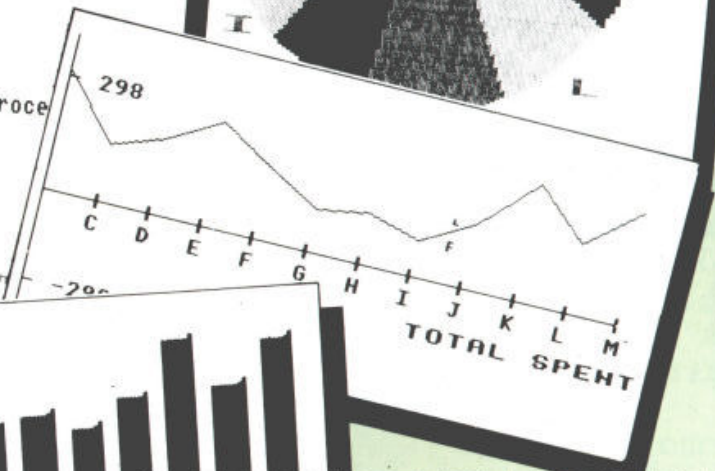


GRAPHICS



WORD PROCESSOR





at the unbelievable

st £5.95 CASSETTE

DATABASE SOFTWARE

Please send me _____ copy/copies of Mini Office for Spectrum 48k/Spectrum+

☐ I enclose cheque made payable to Database Publications Ltd. for £ _____

Only £5.95

I wish to pay by
☐ Access ☐ Visa No. _____ Expiry date _____

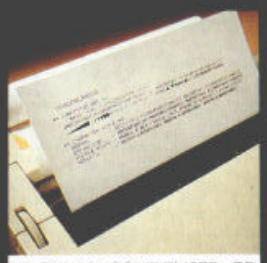
Signed _____

Name _____

Address _____

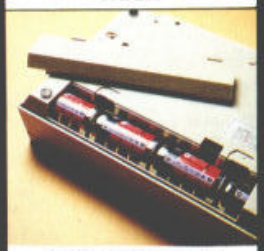
Post to: Mini Office Offer, Database Publications, 68 Chester Road, Hazel Grove, Stockport SK7 5NY.

Little Brothers should be seen but not heard.



REGULAR, CONDENSED, OR EXTENDED FACES.

CUT SHEET A4 OR ROLLER PAPER.



BATTERY OR MAINS OPERATED.

A maxim which eloquently describes the Brother HR-5.

Less than a foot across, it's nonetheless loaded with features.

But there's one thing the HR-5 won't give you. Earache.

For the annoying 'clickety clack' many printers produce is mercifully absent from the HR-5.

Quietly efficient, it delivers high definition dot matrix text over 80 columns at 30 c.p.s.

The HR-5 also has something of an artistic bent.

Being capable of producing uni-directional graph and chart images together with bi-directional text.

It will also hone down characters into a condensed face, or extend them for added emphasis.

Incorporating either a Centronics parallel or RS-232C interface, the HR-5 is compatible with most home computers and popular software.

Perfectly portable, the battery or mains operated HR-5 weighs less than 4lbs.

Which is really something to shout about.

brother

The future at your fingertips.

DEPARTMENT P, BROTHER OFFICE EQUIPMENT DIVISION, JONES + BROTHER, SHEPLEY STREET, AUDENSHAW, MANCHESTER M34 5JD. TELEPHONE: 061-330 6531. TELEX: 669092. BROTHER SHOWROOM: 83 EUSTON ROAD, LONDON NW1 TELECOM GOLD: 83: JBC002. BROTHER INDUSTRIES LIMITED, NAGOYA, JAPAN.

Which modem?

How to choose the ideal communications package to suit your personal needs

WITH most home computers, getting into modem communications means buying an add-on RS232 port plus a modem and software package to go with it. If you are fond of a good struggle and take pleasure in fighting against the odds, then you can take this route with the Spectrum too. But if you just want to get connected up quickly and with a minimum of fuss, there is only one way to go – the VTX5000.

The Prism VTX5000 Prestel modem is a complete package, with all the hardware and software you need for Prestel, Micronet – and a lot of other services – all in one neat box. Chosen as "Peripheral of the Year" in the 1984 British microcomputing awards, it is now being sold very cheaply – the company who made them did not survive for long. Some 10,000 VTX5000s passed through the hands of the receivers and have been on offer at decreasing prices ever since.

The VTX5000 will only operate at the 1200/75 baud split speed mode used by Prestel and other viewdata type databases. This is something of a drawback as most amateur bulletin boards operate at 300/300 baud.

However the Prestel speeds are becoming so widely used that some bulletin boards are offering them as an option to attract people who have come on to the scene with dedicated Prestel terminals. There are also a lot of other online services that use the Prestel system, like local authority information databases.

The VTX5000 connects to the Spectrum edge connector via its own ribbon cable and it has a carry through extension to take other add-ons. It is powered from the edge connector, so the only other lead is the one that plugs into your telephone socket. If you don't have a socket, you will have to get BT to fit one. The telephone itself then plugs into a socket on the VTX5000 and is used for dialing up Prestel or whatever.

The unit will work with the original Spectrum or the Spectrum Plus, but not with the new 128. Owners with microdrives sometimes have problems at power-up, indicated by a screen full of coloured stripes. This can be circumvented by turning the power on at the mains socket rather than by inserting the 9 volt plug at the computer.

The VTX5000 has its software in ROM, which takes control at power-up and transfers its contents into RAM. If you want to access Prestel or Micronet you have a menu of functions to do just that.

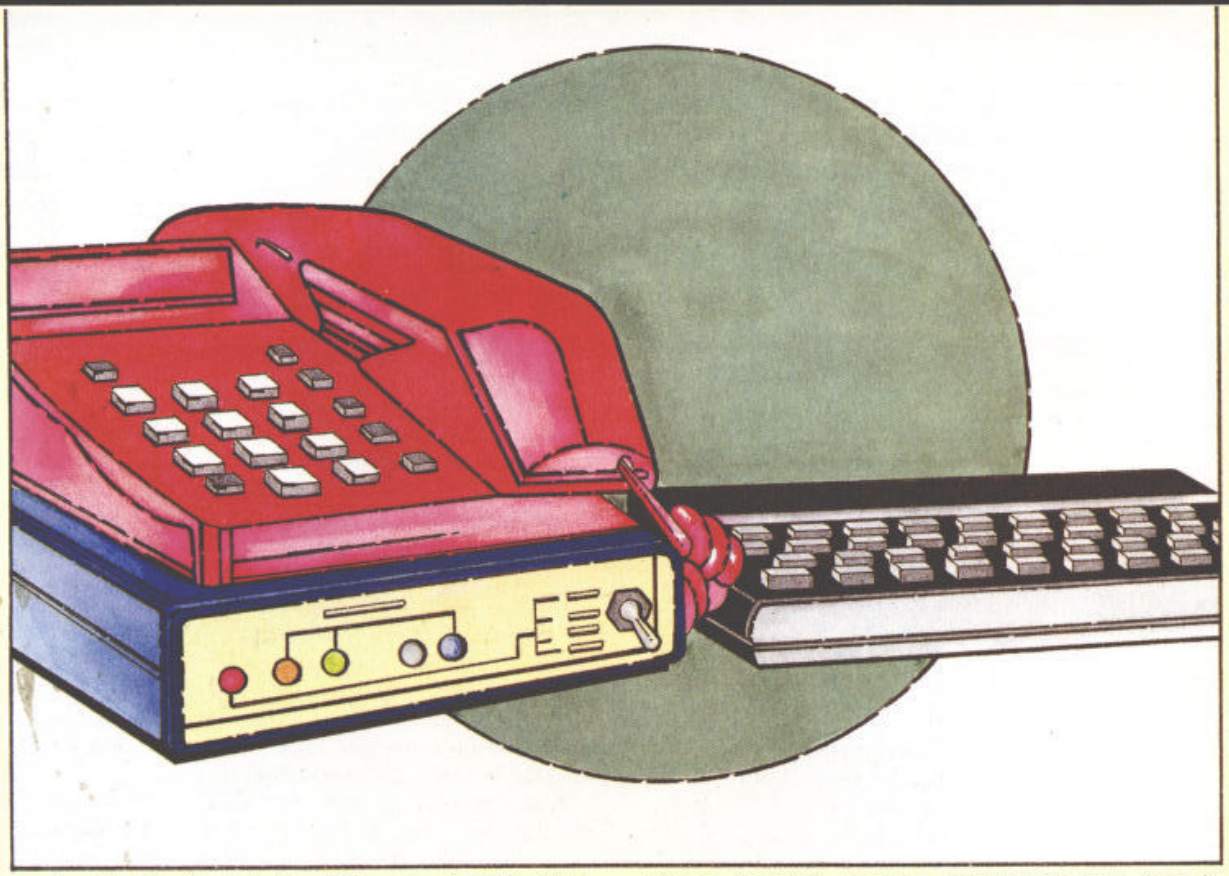
However you can also break in and rewrite the Basic portion of the software to use microdrives instead of tape, for instance. Or you can load one of the special communications programs that allows you to access scrolling type bulletin boards and databases like MicroLink – provided they use 1200/75 baud – rather than the paging Prestel types. These communications software add-ons can be bought from the modem suppliers or downloaded from Micronet.

DIY and why not

There are a number of problems to watch out for if you contemplate buying a modem set up in bits and pieces. This is particularly difficult on the Spectrum because of the lack of a standard serial port. The key to successful DIY Spectrum modem communications is the implementation of a proper RS232 serial interface and the software to drive it. Once you have achieved that, you can use almost any modem.

A lot of the add-on RS232 ports available for home computers, and the Spectrum in particular, are meant merely for driving printers and don't work properly in the full duplex mode needed for modems. The RS232

The Prism VTX5000



port on the Sinclair Interface 1 is a case in point. This port will allow both output and input, but it lacks a proper RS232 chip, and the job of this chip is actually done by the CPU. Unfortunately the CPU cannot be in two places at once, so it can send or receive but not do both at the same time, which is what is required for full duplex working.

Once you have found a serial interface which will work in full duplex and at the required baud rates (the split speed 1200/75 is often left out), then you need some kind of software to drive it. This will have to be written with the right port addresses incorporated to drive the serial interface you have chosen. As there is no standard interface, software writers are put off and there is very little communications software available for the Spectrum. What is available will usually have been written for a specific piece of hardware, which may well not be the piece of equipment you have chosen – so be wary. Ideally the interface and the software should be offered together as a package.

If you cannot buy software to suit your interface the only solution would be to write it yourself. This is a big job, and it really has to be done in machine code. A simple terminal program which lets you type in messages to the remote computer and see its reply on your screen is easy enough, but when it comes to downloading software or saving your electronic mail on tape or disc, then things get complicated. Making provision for all the special Prestel graphics characters and the fact that there are 40 columns on a Prestel page and only 32 on the Spectrum screen is another rather intricate area. This probably puts Prestel outside the DIY sphere.

Maplin Electronics supply a Spectrum RS232 interface in kit form which might suit the more accomplished DIY enthusiast. This has an eprom socket and suitable driving software available in eprom to give easy access to the port, although it is not a complete communications program. More details are available in Maplin's projects book number 8. You can order this from Maplin (see addresses) but don't expect them to give you any information about it over the phone – even if you are a journalist writing an article about their product!

Once you have sorted out the above, you can use any normal modem with an RS232 connection. There

On the left is the Maplin modem kit, while on the right is Maplin's interface kit

Numbers and terms

When buying a modem, you will find the adverts peppered with a lot of different numbers and terms. Here is a quick guide:

V21 *– Another designation for 300 baud full duplex, CCITT tones.*

V22 *– 1200/1200 full duplex, very posh, few users except Telecom Gold.*

V23 *– The Prestel/viewdata standard, 1200/75 baud, split speed duplex.*

Bell *– Indicates USA tones, not usable in the UK (unless calling USA).*

CCITT *– European tone standards, such as V21 and V23.*

Call *– Also called "Originate", these are the right tones for your end.*

Answer *– These tones only useful if you are setting up your own bulletin board.*

AutoAnswer *– As above, answers phone when someone calls your computer.*

Autodial *– What it says, but expensive and needs software to run it.*

Smart modem *– Has built in computer with lots of extra functions.*

Hayes compatible *– Extra functions are accessed by standard commands available in very expensive big computer communications software.*

Acoustic coupler *– Talks through telephone handset, so you don't need a socket – less reliable data transfer than plug in types.*

is a large range of these, made for no specific computer, some offering a choice of baud rates and other extra features like autodialling. You can pay anything from £30 to more than £500, but where the Spectrum is concerned you probably won't need (or be able to use) the features of the more expensive models. Unfortunately, the cheaper makes have not been advertised much recently, and some may have become victims of last year's economic shakeout.

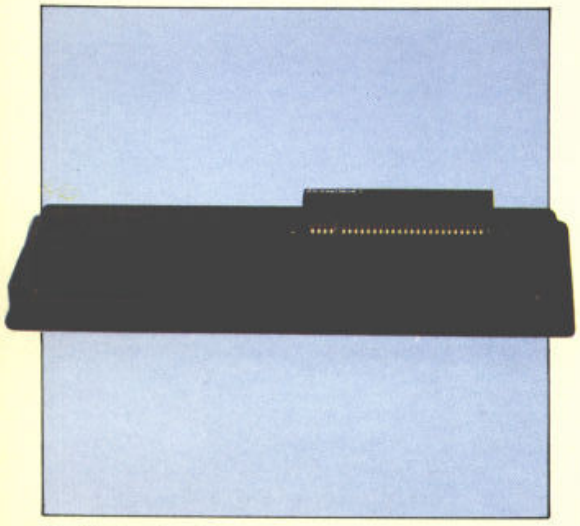
Since you are unlikely to be able to get software to cope with Prestel, there is little point in paying £100 extra for 1200/75 baud capability. Even faster speeds are becoming available at even greater prices, but few of the dial-up services use them yet, so why pay a high price to be a lonely pioneer? This leaves 300 baud. Modems limited to this speed are quite cheap and will get you on to everything except Prestel. Maplin, again, have a suitable modem in kit form. BT surplus units are also available fairly cheaply from Display Electronics.

At this end of the market, there should be no temptation to accept a unit that is not BT approved. I have not heard of anyone actually being prosecuted for using an unapproved modem (or telephone), but BT might well use the threat of prosecution to stop you from using your equipment if they should find out. The VTX5000 is approved, naturally. The Maplin model cannot be approved, as it is supplied as a kit which makes it a homemade item in the eyes of BT, but it uses an approved line transformer.

It is important to remember that the modem only supplies the appropriate audio tones for transmission down the telephone lines and it does not govern the baud rates themselves. The baud rate timing of the transmission must be done by the RS232 interface. This means that a modem with a 1200/75 setting will not work at that setting unless the RS232 interface also has a 1200/75 setting. Unless, that is, you pay enormously (£500) for an intelligent modem that will shift baud rates for you.

To conclude

DIY can be very rewarding, and you learn a great deal, but don't expect to get set up without going through a purgatory of circuit diagrams, solder burns, and wrong connections. Great fun, but is it computing?



Sinclair Interface I

Suppliers

Maplin Electronic Supplies,
PO Box 3,
Rayleigh,
Essex,
SS6 8LR.
Telephone: 0702 552911
Also shops around
the country.

Modem House,
70 Longbrook Street,
Exeter,
Devon,
EX4 7AP.
Telephone: 0392 213355

Display Electronics,
32 Biggin Way,
Upper Norwood,
London,
SE19 3XF.
Telephone: 01-679 4414

Electronic mail keeps you posted

We examine what happens on MicroLink – and why it may well appeal to Spectrum owners

WHAT do television magician Paul Daniels, Japanese journalist Yuichi Ishikawa, Icelandic fisherman Ingebar Oskarsson and Band Aid's Bob Geldof have in common?

Not a lot apart from the fact that they all subscribe to MicroLink, the UK's fastest growing electronic mail and information service.

The first impression MicroLink leaves you with is that it could turn out to be very bad news for Britain's postmen. For if this latest form of two way electronic communications takes off in a big way – as it now seems certain it will – they will have a lot more to worry about than the occasional mongrel snapping at their heels.

Even now the signs are there that conventional mail is on its way out in favour of its contemporary cousin. True, electronic mail has been available for about five years. But in the last 18 months it has really come into its own due to the now widespread use of micros.

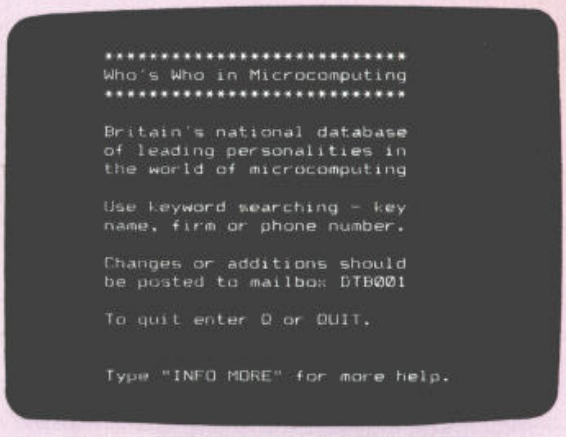
And MicroLink, which started life as part of Telecom Gold – the national computer network – is the fastest growing of all electronic mail services. But all this is jumping the gun for readers who by now will be scratching their heads and asking: "What IS electronic mail?"

The key to understanding it is to know about modems, the low-cost devices which enable computers to communicate with one another over telephone lines. Don't worry about how they achieve this, just accept the fact they can.

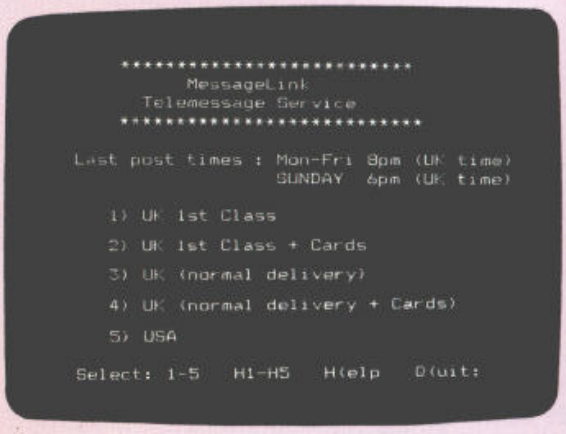
So with a modem connecting your Spectrum to a telephone, you simply type out your messages on the keyboard and transmit them – or alternatively receive ones already electronically despatched to you.

To achieve this you need to belong to a service such as MicroLink which, as the name implies, effectively links micro users together over the phone. The way it does this is through its £500,000 central computer in London which acts as a giant clearing house for messages.

Each subscriber is automatically given a mailbox, in essence a tiny electronic letterbox of his own within the mainframe. So when Paul Daniels wants to get in touch with Bob Geldof, he simply types out his message at home, then transmits it over the phone to the Band Aid mailbox. And to get the reply all he needs to do is access his own mailbox number via his micro and messages waiting for him appear on his monitor screen. These can subsequently be printed out if the nature of the correspondence dictates a copy is required.



The definitive Who's Who . . . can be updated in seconds



Send a letter . . . and the postman will deliver

With electronic mail capable of being sent or picked up 24 hours a day, the Post Office could never hope to compete with the service. But what about other people reading your personal mail? The answer is they can't. It may as well be dropping through your own front door because you need a special password key to get it.

More than mail

Nor is MicroLink confined to electronic mail on the communications front. It also offers a telex facility.

That's why Tokyo magazine editor Yuichi Ishikawa can send a request to Iceland fisherman Ingebar Oskarsson asking for the latest price of cod should he so wish. In fact Yuichi is one of a growing number of Japanese users of MicroLink. This is because they have discovered it is cheaper to send a telex to contacts in Japan via MicroLink than it is to use the Japanese telex service. And even though it means the message bounces via satellite from Tokyo to London and back to Tokyo again there is no appreciable delay in getting it through.

MicroLink subscribers a little closer to home are also discovering they are being offered a bargain with its Telemessage service. With British Telecom having recently put up the price of its Telemessages – the modern equivalent of the old telegram – from £3.50 to £4 for up to 50 words, an almost identical service on

MicroLink – the fastest growing of all electronic mail services

MicroLink has become even more attractive.

For just £1.45 users can send Telemessages of up to 350 words – and the only difference is that one is dictated over the phone while the other is sent by micro.

Yet MicroLink is more than just a new way of sending and receiving messages. The brainchild of the publisher of many of the UK's leading computer magazines, it also offers an ever increasing range of both information and entertainment services.

"We feel that to stand head and shoulders above the other electronic mail services, we have to offer that much more", says Derek Meakin, head of Database Publications, and the driving force behind MicroLink.

The end result is that MicroLink comes across like an all singing, all dancing electronic postman.

Its latest service is to make airline travel easier – and cheaper – for its subscribers. The reason for this is that it is offering instant round the clock information from the bible of globetrotters, the international Official Airlines Guide (OAG).

At the touch of a micro key, users are linked by satellite within seconds to the OAG computer in Oak Brook, Illinois, USA, which monitors the 38,000 changes of fares which take place daily and the 30,000 weekly schedule revisions. It proves to be a real money saver in that fares for all airlines on any given route are made available – from the lowest to the highest.

Say it with flowers

Additional user friendly facilities such as this are helping MicroLink set the pace in this highly competitive field – as it has since the day it was launched. Subscribers can say it with flowers at whatever hour of the day or night the mood strikes them – and their floral gift will be delivered anywhere in the British Isles. This is due to MicroLink having joined forces with the world famous flower delivery service Interflora to create FloraLink, which for the first time ever enables people to send flowers and plants by way of their computer.

And it has already saved one marriage. Every year since his wedding Paul Watson had forgotten his wife's birthday, their wedding anniversary, even her Christmas present.

"So you can imagine what I had to suffer as a result", said the 40-year-old Hull accountant and Spectrum owner.

But now Paul will never be confined to the doghouse again for missing those special occasions. As a MicroLink subscriber he became one of the first people to take advantage of the FloraLink service.

"My wife Pauline loves flowers, and suddenly I realised here was the answer. So now it will be flowers for every occasion – birthdays, anniversaries, the lot – all pre-booked to be delivered annually via FloraLink".

And the train standing on Platform 4 can now be caught courtesy of MicroLink's British Rail telebooking service. It helps subscribers choose their trains by carrying constantly updated timetables,

```
**********************************
        FloraLink
**********************************

Say it with flowers -- via MicroLink!
Send your order now using this simple
form and fresh flowers or plants will
be delivered to any address in the UK
or Eire.  Send before 12 noon (Monday
to Friday) and they will be delivered
the same day - with the help of 2,700
florists who form Interflora.  Please
note that orders for delivery on Sat-
urday MUST be made before 5pm Friday.

Please give your credit card details:

   BCD - Barclaycard
   ACC - Access
   AMX - Amer. Exp
```

No excuse now for forgetting that anniversary

```
**********************************
        British Rail Service
    Tickets  / Fares / Timetable
**********************************


1 : Ticket Bookings
2 : Pullman/Steam Bookings
3 : Seat Reservations
4 : Sleeper Reservations
5 : <<<< COMPETITION  >>>>
6 : Competition Rules
7 : Pullman Rail
8 : Steam Excursions
9 : Timetable and Fares

 1-9   H1-H9   H(elp
R(e-display or Q(uit :
```

A ticket to ride . . . British Rail's information pages

together with fares between London and 20 major cities. Should their destination be London then MicroLink people can even book a ticket for the latest show – after reading all the up to the minute reviews – on TheatreLink.

Weather or not . . .

A recent dramatic breakthrough on the computer front now allows people to discover what the weather holds in store with WeatherLink. Whether they live in Torquay or Turkey, the service means they can tap into the same information from outer space as used by the boffins from the Met Office. Exclusive to MicroLink, it enables subscribers to see the very latest pictures being transmitted by the European weather satellite, NOAA9.

Launched on February 12, 1986, WeatherLink generated such excitement that a team from Thames Television was on hand to cover the event. At exactly 3.05pm, subscriber John Wallbridge achieved a world first by displaying the first weather satellite picture on his micro in his London home.

Should subscribers who use WeatherLink find out it's going to rain, then there's always Who's Who in Microcomputing to entertain them while they are stuck indoors or the opportunity to catch up on the latest happenings in the computer world with MicroNews.

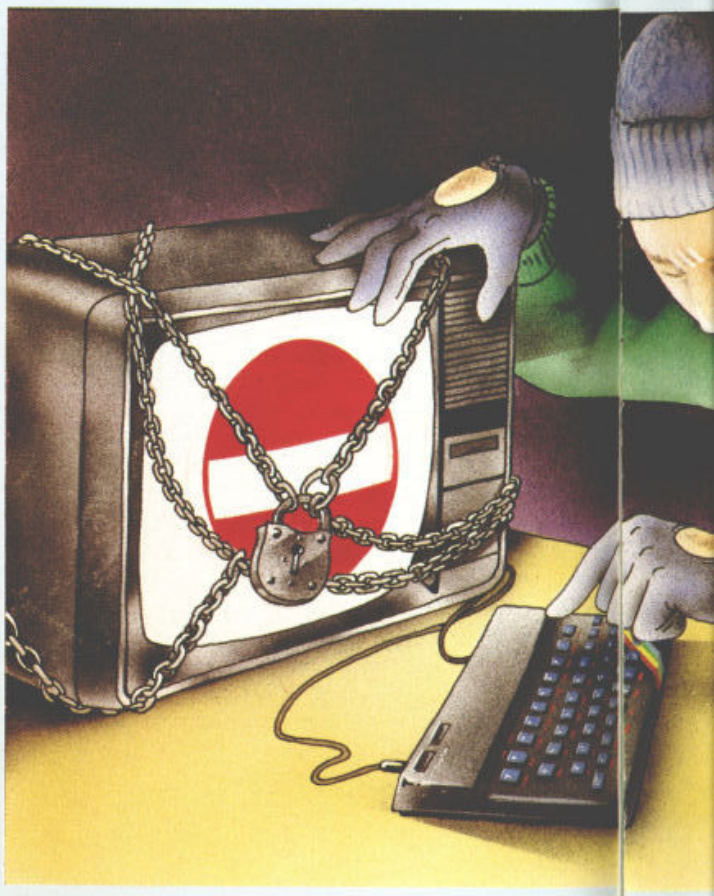
Yet for all these diverse services, MicroLink only charges an initial registration fee of £5, plus a monthly standing charge of £3.

But why provide such a range of other services when MicroLink's prime purpose is electronic mail?

"We believe that apart from being useful, electronic communications using micros should also be entertaining and fun", says Derek Meakin.

Hacking

Is it illegal or can anyone join in? Find out once and for all



IT's a bit difficult to pin down exactly what is meant by "hacking". It seems to be used interchangeably as a term for inspired programming, for breaking the protection on commercial software, or for committing massive fraud via a computer.

A number of the original and genuine hackers in the USA are reputed to have produced a glossary of hacking terms (only available as a computer file to those who are clever enough to find it) in which hacking is defined as making furniture with an axe. It could perhaps be defined more generally as the art of turning a handle in the direction it doesn't go.

The joy is in making something big and complex do tricks that it was not designed to do, especially since this usually means outwitting the designer. But as this issue is mainly about computer communications, let us confine ourselves for the moment to the narrower sense of hacking as using a modem to go exploring where one is not invited.

A lot of newcomers to the communications scene are intoxicated with the myths of hacking and make complete idiots – and nuisances – of themselves on bulletin boards before they have sussed out that there is a great deal more to computer linkups than hacking. In fact, most modem users are not involved with such activity – interested, yes, but not involved.

These new chums are the ones who use imaginative pseudonyms like "The Mad Hacker" or "The Duke of Edinburgh" their first few times on, which is handy really, as some of the things they do would embarrass them later if they used their real names.

What it's not

It might be a good idea to define what hacking is not. Merely dialling up a bulletin board or other online service is not hacking. Most of these services allow visitors or even go out of their way to invite them. Even the somewhat po-faced Prestel has a demonstration password to give all callers limited access, in the hope of interesting them in buying a subscription. To enter the visitor identity and password just keep pressing 4.

Bulletin boards in particular are designed to be accessible. These systems are meant for communication, so merely leaving a message of some kind is not hacking. In addition systems contain a directory of users – this is not hidden, so finding it is not hacking. About eight times a day someone stumbles on the Prestel directory, complete with mailbox numbers, and thinks he has found the password file. He hasn't, but he may not realise it until after he has crowed about it. How embarrassing!

Is hacking wrong?

It all depends. Merely getting into a system and looking around ought to be pretty innocent, but what if it belongs to the Ministry of Defence? How much trouble do you want to get into?

There have been very few prosecutions for hacking, at least in Britain, partly because it is difficult to find a law which has been broken. However there is a law which applies very easily to computer fraud. It is called "obtaining a pecuniary advantage by deception". This would plainly be wrong. There have also been prosecutions for "forging an electronic instrument". So it may be dangerous even to try guessing other people's passwords.

It is well to remember that assuming someone else's identity on a subscription computer system is quite likely to cost them money just for the use you have made of the system. They are entitled to feel aggrieved about this, and it could certainly be construed as theft. It is less obvious that anyone has lost anything when it is an in-house computer in a business or a university that has been invaded, because the legitimate user is not expected to pay for computer time. But the offence of "stealing electricity" may be held to apply.

Another reason for the rarity of prosecutions is simple prudence. Those with the knowledge and ability to get into a system which is really trying to keep them out are usually too sensible to do any real damage. They are interested in the game rather than looking for dishonest gain, and well aware that interlopers may be tolerated where thieves or vandals would not.

In the USA, where many hackers have been arrested, it is significant that the ones in trouble are almost always school kids. This is probably more to do with a youngster's typical lack of responsibility than any preponderance of very young people in the hacking subculture. That is to say, they get caught because their activities make it necessary that they be caught, while others, older and more restrained, escape because their activities are relatively innocuous and no one can be bothered to go after them.

A recent case involved teenagers who were using

stolen credit card numbers – just the numbers, not the cards – to obtain goods illicitly. A lot of this activity took place on dial-up services, because using a computer link disguised the age of the persons placing the orders. The miscreants were also using amateur bulletin boards to exchange numbers and other information.

The police finally caught them by the simple ruse of setting up their own bulletin board, baiting it with some interesting messages, and advertising its existence with messages on other boards. Then all they had to do was trace the calls of the individuals in question while they were absorbed in reading the messages.

The best way to stay out of trouble is the same method you would use in other walks of life – just try not to annoy people. If you must sneak in and read their mail, don't put their noses out of joint by bragging about it. If you can't resist letting them know you got in, then a cheeky message will get them quite angry enough. There's no need to erase files or try to crash their system.

The other side of the coin

Most systems are very well protected against unauthorised entry. Typically, you need to know at least a telephone number, a customer identity code, and a password. Systems that are very sensitive will have even more security.

One home banking service requires a different password for each transaction, in addition to the customer's own log-on password. The passwords are posted to subscribers in small batches whenever the computer reckons they are running low. Each transactional password is used once and then discarded, so if someone does happen to hit on one by a lucky guess, it cannot be used again and again.

The problem that designers of such security systems face is that the legitimate users don't like having to struggle with a difficult series of long numbers or unrelated letters every time they log on to a system. This means that they tend to undo the security of the system by making things easier for their own access. Instead of memorising their identity codes and passwords, they write them down and tape them to the computer where any casual office visitor might see. In cases where they can choose their own password, they choose something easy to remember, like someone's name, or the same number pressed five times. These combinations are obviously much easier for a hacker to guess than a random choice of letters or figures would be.

Even a four character password can be almost impossible to guess if it is made up of a random choice of symbols. Using just numerals and letters, and ignoring the difference between upper and lower case, you have 36 to the fourth power, or 1,679,616 possible combinations. Limiting the choice to first names, of whatever length, brings this figure down to a few hundred.

You can see from this that the legitimate users of a system often turn out to be the hacker's best friend, but oddly enough, the hacker is also sometimes able to do a friendly turn for the systems security man.

It is not unknown for the designer of a system to enlist the aid of hackers in testing security by the simple ruse of giving out the telephone number in a message on a bulletin board, perhaps saying, "There seems to be a computer at this number, does anyone know what it is"? He may even give out a password or use some easy ones to make sure that people will be able to get in and give the system software a good workout.

He can then sit back and monitor how his system responds to thousands of attempts to break it, and after a few months trial, simply change the telephone number when the system actually goes into service. By this time the weak spots should have been discovered, and any subsequent visitors will have a tough time.

How to hack

No one is going to tell you how to hack. You often see newcomers on the bulletin boards asking for numbers and other help, but they seldom get useful replies. Just being told what to do by someone else isn't hacking, you see.

The quickest way to get useful information about on-line computing and thence hacking is to become a legitimate subscriber to one or more of the available dial-up services. When you know how a system is meant to work, it is much easier to find ways of using it creatively in ways the designers hadn't thought of.

A good start might be to join Database Publications' own MicroLink service, which is a part of Telecom Gold. Quite aside from giving you access to one of the most successful electronic mail systems going, you will also be initiated into the mysteries of Packet Switch Stream, or PSS, and learn how to use PADs, NUIs, and NUAs. You may even find that legitimate access to such a system is so much fun that you lose all urge to hack into anything else, but if not, at least you will be better equipped.

How Basic works

First of a two-part series on how your Spectrum interprets its language

FANTASTIC, isn't it? You plug in your Spectrum, turn on the television, and as the screen warms up you know that Spectrum Basic is already awaiting your command.

In this two-part series we'll be taking a look at how it works. The general principles are pretty much the same for most other micros as well, but there are just enough differences to make it very interesting – or frustrating – if you try to transfer what you learn here directly to other machines.

Perhaps the most important difference between Basic and other high-level computing languages such as Fortran and Cobol is that Basic is an interpreted language, whereas the latter two are compiled.

That is to say, a program written in Cobol or Fortran is turned into machine code en bloc and then executed, whereas one written in Basic is turned into machine code statement by statement, each statement being executed as it is interpreted. There are compiled Basics, but they are outside the scope of this series.

This has various effects, both good and bad. The ones that most concern us here are that it makes Basic rather a slow-running language – although this isn't too apparent on most modern machines unless you go in for a lot of screen animation – and it means that the source program (the Basic program that you actually wrote) remains in the computer's memory all the time it is being executed. And that in turn means that it is very easy for us to watch it working.

Peeking around

Now it's time to actually poke around in your Spectrum's innards – or PEEK around, to be more precise. Because there are no hard and fast rules about where Basic programs are stored, the first job is to discover the beginning of the program storage area on your set-up.

This information is contained at locations 23635 and 23636, and you can discover it by keying in the following in immediate mode:

```
PRINT PEEK 23635 + 256 * PEEK 23636
```

We'll see the reasoning behind this later.

The information is actually contained in a special sort of variable called a system variable – that is, one that the computer itself sets and resets in order to keep track of where information of various kinds is stored.

We shall be looking at system variables in a little more detail next time. If you don't have discs or microdrives you will most probably get the answer 23755, and this is the number that we shall use in the examples that follow. But if the answer you got was different, use the number that you got instead.

Now type in the following Basic program exactly as it is here:

```
10 REM This is a remark
20 PRINT "This is a PRINT statement"
```

Then key in the following direct command:

```
j=23755:FOR p=j TO j+21:PRINT PEEK p; " ";NEXT p
```

and the computer will respond with this sequence of numbers:

```
0  10  18  0  234  84  104  104  115  32
105  115  32  97  32  114  101  109  97
114  107  13
```

What you are seeing here is how the first line of the Basic program you typed in is stored in the computer. Each number represents the contents of one byte of memory, starting from the point at which your program begins. As you probably know, a byte can store any number in the range 0 to 255.

The first two numbers, 0 and 10, are the number of the first line of the program. It just so happens that the Spectrum stores line numbers in the same order that humans regard as correct, with the high number first and the low number second.

Rights and wrongs

For hardware reasons, however, computers store many numbers in the "wrong" order, with the low number first and the high number second.

To work out the number encoded in the two bytes, you multiply the high byte by 256 and add it to the low byte. In this case it's 0×256+10=10, our first line number.

You can see the more usual lo byte/hi byte order in the next two numbers, 18 and 0. These mean that this line of program needs 18 bytes to hold it – not including the two bytes used for this information or the two that store the line number.

We arrive at 18 by multiplying the 0 – the high byte this time – by 256 and adding 18, our low byte. We discovered the start address of the program storage area by a similar lo byte, hi byte calculation.

After this comes the code 234 which stands for the Basic keyword REM. All the standard Basic keywords – Basic's vocabulary – are represented in a similar way, and the codes are called tokens. The simple rule here is that every command which you can input on the Spectrum by means of a single keypress is represented by a token.

Note, by the way, that the spaces which the

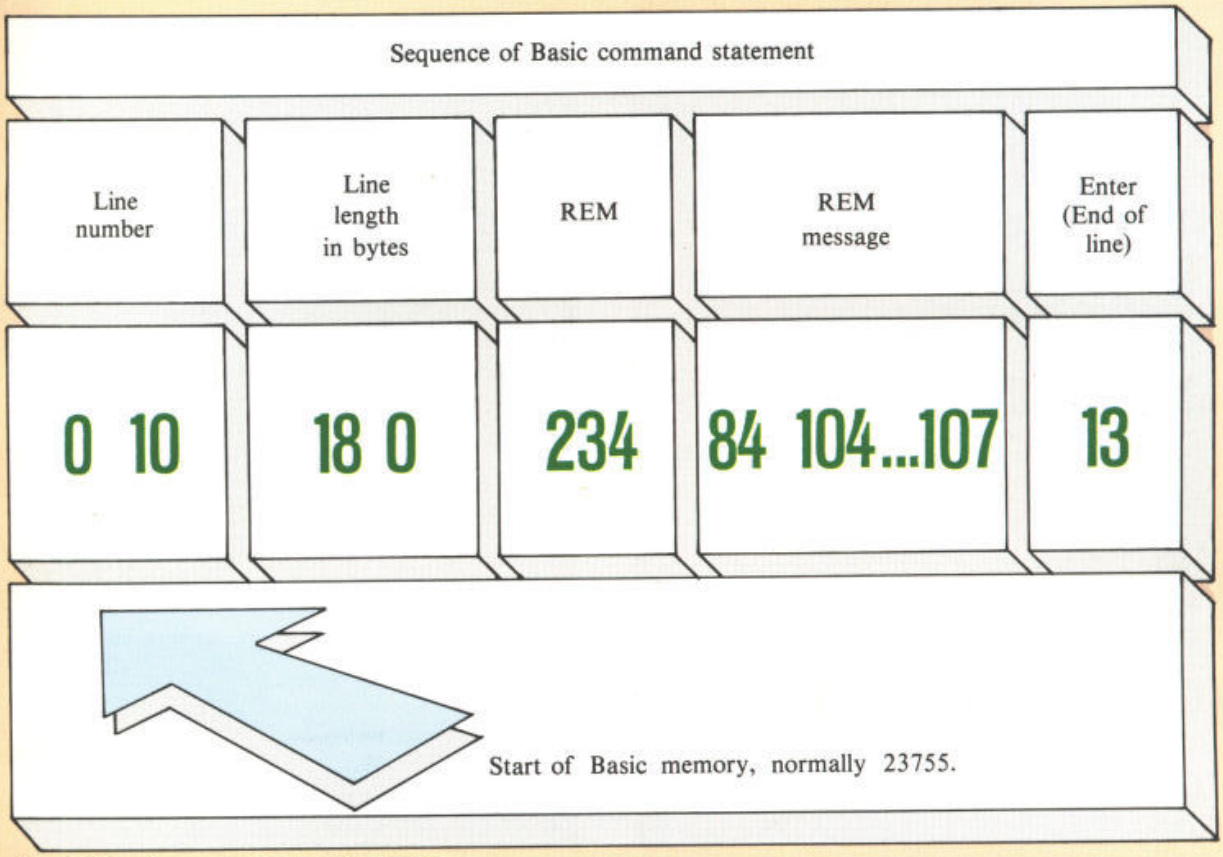


Figure I: How the program is stored in memory

Spectrum automatically inserted after the line number and again after the keyword REM are not represented in the memory. They only appeared to make it easier for you to read the line.

Then it gets simple. If you look up the next 16 numbers – as far as 107 – in Appendix A of the Basic programming book, you will see that they are the character codes for the remaining letters in the statement – 32 is a space, 84 is a capital T and so on. Finally, the sequence finishes with 13, which means Enter, as an end-of-line marker.

All Basic command statements are represented in the same general way. If you're still confused, Figure I may help to make things a little clearer.

Memories are made of this

By now you have probably worked out how to inspect the representation in memory of the second line of the program. Clear the screen (to keep things tidy) then enter the following direct command:

```
j=23777:FOR p=j TO j+32:PRINT PEEK p;
" ";NEXT p
```

and the computer will respond with:

```
0 20 29 0 245 34 84 104 105 115
32 105 115 32 97 32 80 82 73 78
84 32 115 116 97 116 101 109 101
110 116 34 13
```

You know now that the first two numbers show that this statement is number 20, and the two numbers after those (lo byte/hi byte – remember?) inform you that it is 29 bytes long. Then comes the token 245, which stands for PRINT, a double quote – 34 – and so on.

But note that here the word PRINT, which formed part of the message included in the quotation marks, is listed as a series of character codes (80, 82, 73, 78, 84) rather than as a token, because tokens only stand for instructions that the computer has to obey and not for similar words which occur inside quotation marks.

Statement of fact?

And finally there is the Enter token – 13 – to confirm that we have come to the end of the line.

At this point you would probably assume that all Basic statements are represented in memory in the same way. Actually this is not quite true. Get rid of your program by entering NEW, then enter the following:

```
10 LET p=1
```

then retype a version of our familiar direct command line to see how that program was represented.

```
j=23755:FOR p=j TO j+14:PRINT PEEK p;
" ";NEXT p
```

You should get the response:

```
0 10 11 0 241 112 61 49 14 0
0 1 0 0 13
```

Some of that is certainly familiar enough. You will recognise the line number – 0 10, – the line length, – 11 0 – and the Enter at the end – 13.

Other codes which shouldn't cause any problems are the token for LET – 241 – and 112, 61 and 49, which are the character codes for the variable name *p*, the equals sign and the number 1 respectively.

The rest of the line is a little more difficult, and we shall leave a detailed analysis of it until next time. If you want to explore it a little more in the meantime, try entering different values for the variable *p* in the program (both integers and real numbers) and see if you can work out for yourself what is happening.

A useful clue is that in statements that include numbers, Spectrum Basic represents the numbers not once but twice – once for the purposes of LISTing and once for its own internal operations.

Drawing it out...

What's the best way for you to draw graphics on screen? Our survey helps you decide

THERE has long been a search for alternative methods of entering data – and graphics in particular – into computers, as the keyboard is simply not designed for this sort of work.

In the professional world there exists a huge range of inputting devices. Most of these have lower cost – and so lower performance – versions for home micros and the Spectrum has been well supported with light pens, mice (covered elsewhere in *The Complete Spectrum*), digital tablets, tracers and such like.

The choice is basically determined by your answers to two questions:

1. How serious are you about drawing on your Spectrum?
2. How much can you afford to spend?

For most people the first device contemplated is a light pen.

Light pens

A light pen is little more than a photoelectric cell which allows a larger current to pass in the presence of light.

When held against the screen the pen can successfully detect the passing of the electron beam as it scans the TV screen. Since the computer "knows" when the electron beam began building up the picture it can calculate where the beam is and so work out where the light pen is positioned on the screen.

Light pens are relatively inexpensive but unfortunately they are fairly inaccurate – only to about two or more pixels. I have used them on many occasions but never found one where the resolution matches that of the Spectrum, and as such they are more or less useless for serious drawing.

They are useful, however, in that they are normally easy to incorporate into your own programs, and routines can easily be written to select menus from your own screen in much the same way as a mouse does.

Incidentally dk'tronics claims that its light pen can now resolve down to one-pixel accuracy, so that might be worth a look.

A variation on the light pen is the Stack light rifle which is really just a light pen that can be held some distance away. This can be used within some games rather like a joystick or keyboard but the software is basic and unadventurous.

There are a few drawbacks with light pens. The fact that you are drawing on the screen direct means that you are scribbling vertically and this rapidly becomes tiring. The light sensitivity setting may sometimes be rather too critical, making it very difficult for the pen to react properly to frequent changes of light and colour. Definitely not recommended for any other than very basic graphics.

Digital tracers

These aid the tracing of diagrams and pictures, and consist of a double-jointed arm with a pointer or cross-hair arrangement at one end. The position of the pointer can be worked out from the angle of the two joints in the arm.

On larger machines the tracer normally comes attached to the board, and quite often with a perspex cover to help in stabilising your original.

Tracers tend to be more accurate than light pens and, as they are horizontal, they are a great deal less tiring to work with. But the jointed arm does make it cumbersome to attempt freehand drawing.

Graphics tablets

The graphics tablet is probably the device to aim for. Tablets are horizontal, with no obscuring double-jointed arms, and are about as complicated to use as a pencil. There are basically two types – touch sensitive tablets and electromagnetic pads.

Touch sensitive tablets

In the touch sensitive tablet two thin films are separated by an insulating layer.

Pressure on the pad with either finger or stylus forces the two layers together and a built-in microprocessor calculates the x and y coordinates, then passes them on to the computer. The resolution is fairly good.

Obviously the main disadvantage of this type of tablet is that every pressure on the pad will be registered. This makes it a little uncomfortable to use as you must be very careful not to rest your wrist or any other part of the drawing hand on the pad.

There are two makes of tablet – the graphics tablet from Saga which has now been discontinued and the Touchmaster. This is quite a large unit which, in addition to providing drawing facilities, can be used with a variety of simple games for very young children.

The pad has an overlay for each game that allows you to interact with the computer by pressing the relevant picture on the overlay without using the keyboard.

If you want a general purpose unit that can double as an educational module in this way then this unit will do all you ask.

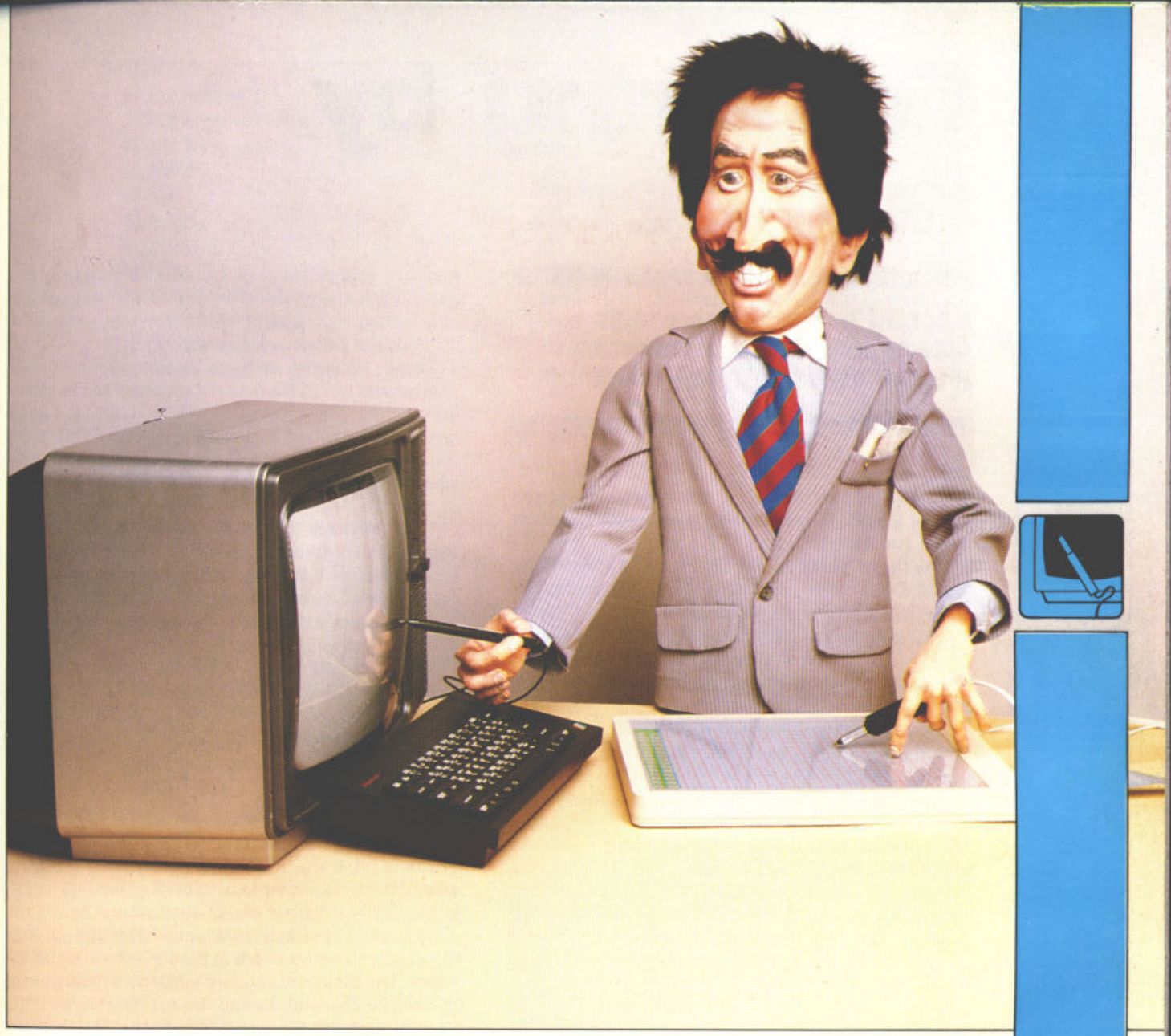
Electromagnetic tablets

If, however, you require something that will enable you to take the Spectrum to its limits graphically then there is only one choice – an electromagnetic tablet.

This is the true graphics tablet of the professional, and there is only one that will link up to a Spectrum – Grafpad from British Micro.

The board contains a grid of wires, and the stylus (linked to the board by a lead) is little more than a tiny switch. When the stylus is pressed against the tablet the on-board microprocessor detects which wires are closest to the stylus and sends this information to the Spectrum.

The system is very accurate and there is usually no problem working at a resolution of one pixel. It doesn't



suffer from false readings if you steady your hand on the pad's surface as it can only detect the single built-in drawing implement. Also, it can be used very easily to trace a picture taped to the surface of the tablet.

A strange feeling?

Using Grafpad – or Touchmaster, or even the RD Tracer for that matter – is a very strange affair at first and takes a fair bit of getting used to. The main difficulty is that all the artwork is being developed on screen while your hands seem totally removed from the result.

This can be very disorientating at first but after a few days it becomes second nature – indeed it's a good job the Spectrum's resolution is not better than it is or I would find it almost impossible to go back to using ordinary pen and paper again.

Software

All these devices come with their own software. Most supply the basic features of a graphics package and support the drawing of lines, circles and so on.

The most sophisticated is the one that comes with Grafpad, which allows you to edit at pixel level by enlarging the screen up to four times.

The screen can be mirrored or scrolled in any direction, and the attributes can be inverted. Triangles, boxes, flood fill and the ability to add text at any point are all possible.

Editing can be carried out either on screen alone, on the attributes or on both, and small sections of the screen can be worked on by setting a window which can then be repeated anywhere else required.

It's also possible to make up UDG banks very easily using the "get UDG" command. The main omissions are the absence of a rotate facility and a patterned fill. And the ability to scale the picture as in the keyboard program Melbourne Draw would also be useful.

In general small problems like this aren't important. It's always possible to transfer partially-completed SCREEN$ between packages to obtain all the facilities required.

Conclusion

Although I have tried all the previous types of graphic input device I have finally settled with Grafpad and wouldn't change for all the RAM chips in Taiwan.

Grafpad are currently revising their range, so if you're interested it's worth giving them a ring on 0923 48222. If you can afford this unit I don't think you will ever regret your purchase or exhaust its capabilities.

Colouring by numbers

Second in the series which clearly explains the complex area of machine code graphics

WE have already seen that the Spectrum's screen display is built up by combining the information stored within two separate areas of RAM.

The first is called the display file and the second the attribute file. The two sections are responsible for different aspects of the picture you see.

Last time we looked at the information stored in the display file. We saw that this section of RAM is responsible for the shapes of the characters displayed on the screen.

Now we're going to look at the attribute file and see how the information stored here is used to colour the screen display.

As we saw in the first article the display file starts at &4000 hex, or 16384 decimal. It's &1800 bytes long ending at &57FF. Immediately following this is the attributes file starting at &5800.

First try experimenting using Program I. Hold down a key and notice how the screen is built up as the program runs through the attribute file storing 255 in each memory location. The current address is printed in hex.

This demonstration shows quite clearly that the attribute file is 768 or &300 bytes long, comprising 24 rows of 32 bytes. This is exactly the same as the number of character positions we can print at.

Apart from showing us how big the attribute file is, Program I isn't much help – it merely turns the screen bright white. That's the effect of storing 255 in successive bytes.

Each character on the screen has a corresponding attribute byte holding its colour information – ink, paper, flash and bright. All this is crammed into eight bits. It might seem a lot of information to squeeze in but it's coded in rather a clever way.

There are eight inks and eight paper colours numbered 0 to 7. This range of numbers can be stored in only three bits – 000, 001, 010....111. Bits 0, 1 and 2 hold the ink and bits 3, 4 and 5 the paper. This leaves bits 6 and 7 which are used to store two flags indicating whether the character is normal or bright and flashing or steady.

There are only three true colours – green, red and blue – called the primary colours. All the others you see on the screen are made by mixing these three. White is easy – it's all three – and black is a total absence of colour. Yellow is a mix of red and green, cyan is green and blue and so on.

The three bits used for the ink or paper tell the Spectrum's ULA, a chip responsible for the screen display, which of the three primary colours to use.

If the bit pattern is:

Green	Red	Blue
1	1	0

the colour displayed is yellow – green plus red. You can see now where the Spectrum get its ink numbers from, yellow is ink 6 which is 110 in binary. You should be able to work out the others without too much difficulty. If not, Table I has all the answers.

Program II can be used to investigate the attribute file. A character is printed at the top left corner of the screen and the corresponding attribute byte is printed in binary. The bits responsible for BRIGHT, INK, PAPER and FLASH are labelled. Table II summarises the function of the attribute byte.

Now we know the size and location of the attribute file and how the colour information is coded we can write some simple machine code routines. Program III fills the attribute file with any value. It can be used to clear the screen.

We use the normal default value of &38. This is

```
 10 REM PROGRAM I
 20 BORDER 5
 50 LET h$="0123456789ABCDEF"
100 FOR a=22528 TO 23295
110 POKE a,255
130 PRINT AT 15,5;
140 GO SUB 600
170 IF INKEY$="" THEN  GO TO 170
180 NEXT a
190 STOP
500 REM --- Hex print a ---
600 LET x=INT (a/256)
610 GO SUB 630
620 LET x=a-256*x
630 PRINT h$(1+INT (x/16));h$(1+x-16
*INT (x/16));
640 RETURN
```

Program I

Colour combination			Result
0	0	0	Black
0	0	1	Blue
0	1	0	Red
0	1	1	Magenta
1	0	0	Green
1	0	1	Cyan
1	1	0	Yellow
1	1	1	White

Table I: Colours available

```
10 REM PROGRAM II
15 REM ----------------
20 REM Alter these!
30 PAPER 3
40 INK 5
50 FLASH 1
60 BRIGHT 1
70 PRINT "?"
80 REM ----------------
100 LET a=PEEK (22528)
110 LET b$="------------"
200 INK 0: PAPER 7: BRIGHT 0: FLASH
0
205 PRINT AT 3,0;"Attribute=";a
480 PRINT AT 6,0;
490 INK 2
500 FOR i=7 TO 0 STEP -1
510 IF a<2^i THEN  PRINT 0
530 IF a>=2^i THEN  PRINT 1: LET a=a
-2^i
535 PRINT
540 NEXT i
550 INK 1
560 PRINT AT 5,0;b$;AT 7,0;b$;AT 9,0
;b$;AT 15,0;b$;AT 21,0;b$
570 INK 3
600 PRINT AT 6,5;"Bright";AT 8,5;"Fl
ash";AT 12,5;"Paper";AT 18,5;"Ink"
700 GO TO 700
```

Program II

```
;Fill attribute file

LD A,&38        ;fill byte
LD HL,&5800     ;start
LD DE,&5801     ;start+1
LD BC,&2FF      ;length-1
LD (HL),A       ;fill 1st
LDIR            ;fill rest
RET
```

Program III

```
;Bright toggle

LD E,%01000000  ;mask
LD HL,&5800     ;start
LD BC,&300      ;length
.loop
LD A,(HL)       ;get byte
XOR E           ;toggle bit
LD (HL),A       ;store
INC HL          ;next byte
DEC BC          ;counter-1
LD A,B
OR C
JR NZ,loop      ;finished?
RET
```

Program IV

stored in the first location and then copied throughout the rest of the attribute file. In binary notation &38 is %00111000 which is equivalent to INK 0, PAPER 7, BRIGHT 0 and FLASH 0. Replace this with any value you want.

Program IV is quite interesting as it toggles the BRIGHT flag in bit 6. Calling it once changes every normal character to bright and every bright one to normal. Calling it again resets everything back to its original state. It's useful for explosion effects.

It works by loading each attribute byte into the A register, XORing it with a mask in the E register, %01000000, and placing it back in the memory.

A better explosion effect can be obtained by exchanging the ink and paper for each character. Program V shows how this is achieved. Call it once to swap the ink and paper, call it again to restore the screen to its original state.

It works by exchanging bits 0, 1 and 2 – which are responsible for the ink, remember – with bits 3, 4 and 5,

Bit	Function	Range
7	Flash	0-1
6	Bright	0-1
5 4 3	Paper	0-7
2 1 0	Ink	0-7

Table II: The structure of the attribute byte

the bits responsible for the paper. So an attribute byte of %00101011, INK 3, PAPER 5, would become %00011101, INK 5, PAPER 3.

● ***That's it for now. In the last part of this series we'll see how to read the keyboard and move things round the screen.***

```
LD HL,&5800     ;start
LD BC,&300      ;length
.loop
LD E,(HL)       ;get byte
LD A,%11000000
AND E
PUSH AF         ;save bits 6,7
LD A,%00000111
AND E           ;get ink
RLCA
RLCA
RLCA
LD D,A          ;save new paper
LD A,%00111000
AND E           ;get paper
RRCA
RRCA
RRCA            ;new ink
OR D            ;and paper
POP DE          ;get bits 6,7
OR D            ;done it!
LD (HL),A       ;store
INC HL          ;next byte
DEC BC          ;counter-1
LD A,B
OR C
JR NZ,loop      ;finished?
RET
```

Program V

Programmers do it with style

First of a two-part series which illustrates the many benefits of a structured programming style

THE success or failure of a beginner's first attempt to write a program depends on how much thought has gone into it before he goes anywhere near his micro.

"Structured" is an adjective often used by the experts to describe such a format. A structured program is one that has been carefully put together using a collection of separate routines, each controlled by a main section.

You can liken this to a prefabricated house with each wall, window or door representing a little program in its own right. Producing the finished building, linking the modules together according to a predetermined plan, is a much simpler task than trying to create something from the raw materials without any plans whatsoever. And the finished building is much easier to dismantle should the need arise.

You can relate this concept to programming. A well-planned idea, using individually written routines, is simpler to put together. The finished product also benefits from being easy to unravel and consequently easier to debug, both for the writer and for people who may want to alter the program.

In order to grasp what the phrase "structured programming" really means we're going to put together a very simple Minefield game that will not only be structured, but will also contain some useful programming ideas that could be incorporated into other programs.

The idea of the game is as old as the hills and very simple. There are a number of mines hidden in a square grid, and by entering X, Y coordinates you attempt to find them all in the least possible number of goes.

This version will be a slight variation on that, consisting of a 10 by 10 grid of boxes in which are hidden 10 little user-defined characters called Smileys. You enter the coordinates of the square that you think each one is in, column first, then row. If you choose correctly the Smiley is displayed accompanied by a suitably triumphant sound. Should you choose wrongly you will be greeted by a rather different sound and given clues as to the whereabouts of the nearest target.

Planning

The first thing to do is to plan the program by breaking it into a series of small sub-tasks. And do this on paper – not in your head – so you can keep track of things. The most experienced programmers discipline themselves in this way, and if it's good enough for them it's good enough for you. So here's a list of tasks the program will be carrying out:

1. Initialise arrays and variables.
2. Define characters.
3. Draw and set up screen.
4. Position each Smiley.
5. Take input, check and validate.
6. Check input against 4, give clue.
7. Show Smiley if correct.
8. Close game or re-run.

Spectrum + Plan = Program

Each of our mini problems will be handled by a subroutine, so to start with we'll use REMs to create a skeleton program.

The first part consists of a series of GOSUBs, one for each of our tasks, neatly REMmed so we know its purpose. The second part consists of dummy subroutines – just a labelling REM, a RETURN, and a separator in the form of a line of asterisks – corresponding to each of our GOSUBs.

```
 40 GO SUB 400: REM characters
 60 GO SUB 600: REM draw screen
 80 GO SUB 800: REM position Smiley
100 GO SUB 1000: REM take input
120 GO SUB 1200: REM check for Smiley
140 GO SUB 1400: REM game over
170 REM ****************************
400 REM create characters
430 RETURN
440 REM ****************************
600 REM draw screen
760 RETURN
770 REM ****************************
800 REM position Smiley
860 RETURN
870 REM ****************************
1000 REM take input
1090 RETURN
1100 REM ****************************
1200 REM check for Smiley
1340 RETURN
1350 REM ****************************
1400 REM game over
1480 RETURN
1490 REM ****************************
1700 REM found Smiley
1730 RETURN
1740 REM ****************************
```

Program 1

And really that's the game in a nutshell, neatly built up in sections controlled from the first part of the listing. Granted, all the meat is missing – hence the missing line numbers – but the logic's all there. I've deliberately left large gaps to allow room to expand and add lines whenever I wish, without running out of line numbers. We'll fill in a lot of them later. Because the standard 16k or 48k Spectrum has no RENUMBER facility, unless you leave yourself room to expand you'll find it very difficult to keep to a neat easy-to-follow numbering scheme.

Take another look at Program I – the following points are important factors in assisting good programming style.

- All GOSUBs are REMmed to identify the purpose of the subroutine being called.
- Each individual subroutine is REMmed for the same reason.

Initialisation

Initialisation is the vital first stage of our program. Here we set the stage by giving some variables their start values, and performing other housekeeping activities. Once we're through initialisation, apart from the character definition routine, the rest of the program is one large loop that gives us repeats of the game.

You should always label a program with its name in a REM somewhere early on in the listing. So our first line will be:

```
10 REM Smiley Hunt
```

During the game we are going to be using a lot of variables, and where the Spectrum allows it we'll give them names that actually means something. This not only helps us to pick our way through the listing, it also helps anyone else who may need to do the same.

Three important variables in our game are *best*, for the best score so far, *turns*, the number of guesses we've had, and *smileys*, the number of faces we've found.

It is important that *best* is initialised early on in the program so that it doesn't get reset in the main loop each time the game is re-run. For this reason it is at line 20. I've given it an arbitrary value of 50 for you to beat, but any number will do to get the game started.

```
20 LET best=50
```

We now need to set up a two-dimensional grid to store values in the various squares. We can achieve this by using the array *b(10,10)*, reserving sufficient memory for 100 boxes (10×10), all accessible by the numeric variable *b*. This is by far the most efficient way to tackle the problem. It is also placed early in the program outside the main loop as we don't want it done each time the game is re-run. So line 30 reads:

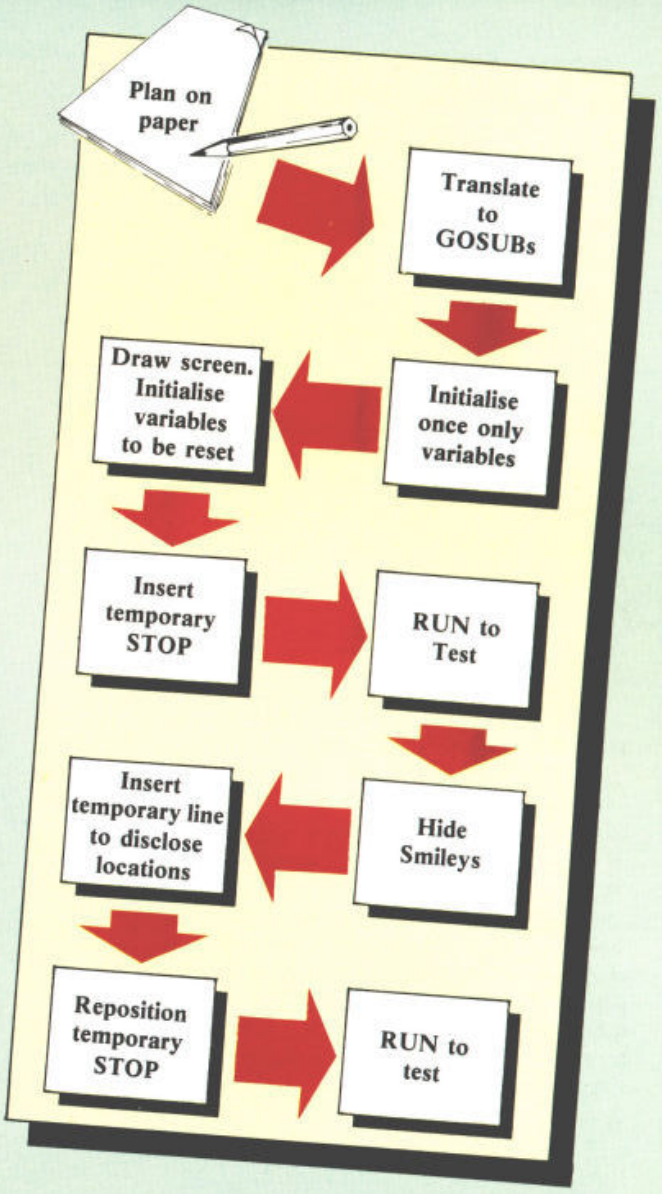
```
30 DIM b(10,10)
```

Setting up the screen

The initialisation stage is now complete, so if you look back at our list you'll see that the next job is to draw the screen. Apart from items of text like the title, the main part of the screen consists of a grid of boxes that conceal the Smileys.

You could show these boxes as any character you wanted from the Spectrum's character set, but we're going to implement a user-defined block for this task. Not only does it look better, we can also get some practice at creating our own characters.

Line 40 calls the "create characters" subroutine at line 400 to achieve this. The box is not the only shape we need. We also need our Smiley, so they are both defined in this routine which READs the DATA contained in lines 3010 and 3020 and then POKEs it into memory using the USR command. The box is allocated to the character "a" (Ascii 144), and the Smiley to the character "b" (Ascii 145).




```
 410 FOR c=0 TO 7: READ bits: POKE US
R "a"+c,bits: NEXT c
 420 FOR s=0 TO 7: READ bits: POKE US
R "b"+s,bits: NEXT s
```

```
3000 REM character data
3010 DATA 0,BIN 01111110,BIN 01111110
,BIN 01111110,BIN 01111110,BIN 01111
110,BIN 01111110,0
3020 DATA 0,BIN 01111110,BIN 10011001
,BIN 11111111,BIN 10111101,BIN 110000
11,BIN 01111110,0
```

Once the user-defined characters are set up we can move on to set up the screen. The subroutine to do this starts at line 600 and is called by line 60.

We first choose the INK then clear the screen. Then the two scoring variables *turns* and *smileys* are

```
610 INK 2: CLS
620 LET turns=0: LET smileys=0
630 PRINT AT 1,8; INK 2;"**********
****"
640 PRINT AT 3,8; INK 2;"*"; INK 0;"
Smiley Hunt "; INK 2;"*"
650 PRINT AT 5,8; INK 2;"**********
****"
660 PAUSE 100
670 FOR x=9 TO 18: FOR y=10 TO 19
680 PRINT AT y,x;CHR$ 144
690 LET b(x-8,y-9)=0
700 NEXT y: NEXT x
710 INK 1: PRINT AT 8,9;"0123456789"
720 FOR y=10 TO 19: PRINT AT y,28;y-
10: NEXT y
730 PRINT AT 7,14;"x": PRINT AT 14,2
1;"y"
740 INK 2: PRINT AT 11,0;"Guesses":
PRINT AT 12,3;turns: PRINT AT 14,24;
"Smileys": PRINT AT 15,27;smileys
750 PRINT AT 17,2;"Best": PRINT AT 1
8,3;best
```

initialised so that each time the game is re-run they are set to zero. Three lines of PRINT statements are next used to put the title on the screen inside a box of asterisks.

To create the grid we need to PRINT our box character, CHR$ 144, on the screen. This is achieved with two nested FOR . . . NEXT loops. The first generates the X location of each box, while the second generates the Y.

These FOR . . . NEXT loops are also used to place 0s in all the elements in our array *b( )*. Line 690 does the job. Remember, when typing in the NEXTs in a nested loop, you must put them in reverse order to the FORs, as in line 700.

We now need to print the numbers 0-9 for the X and Y coordinates and this is done in lines 710 and 720. We're using single digits rather than the numbers 1 to 10 because they can be entered using one key press when it comes to the input routine.

The horizontal numbers are easily printed in a string but the vertical numbers are slightly more difficult to deal with. We obtain them by printing the FOR . . . NEXT variable *y* with an offset 10 subtracted from it. The first time through the loop *y = 10*, so *y–10*, 0, is printed at location 10. The second time through *y = 11, so y–10*, or 1, is printed at location 11 and so on until *y* is 19 and 9 is printed.

Finally the subroutine prints a small x and y to show the axes of the grid, and the headings for the scores. Underneath these are the numbers that will hold the scores, which at the moment hold the values we have assigned to them.

Now enter line 65. This will stop the game once the screen is set up.

```
65 STOP
```

Run the program and you should see on the screen, in addition to 100 red boxes, the numbers 0 to 9 running across the top of the columns, and also down the right-hand side, with a small x and y showing the axes. If your screen differs in any way check your typing for errors, including all punctuation marks.

Hide and seek

The next task is to hide the Smileys in the grid. This is achieved using the subroutine starting at line 800 and called by line 80.

```
810 FOR s=1 TO 10
820 LET smileyx=INT (RND*10)+1: LET
smileyy=INT (RND*10)+1
830 IF b(smileyx,smileyy)=1 THEN GO
TO 820
840 LET b(smileyx,smileyy)=1
850 NEXT s
```

A FOR . . . NEXT loop generates 10 random numbers for *smileyx* and *smileyy* – the coordinates of our smiling face. Then using line 840 we place 1s in the elements in our array.

Line 830 checks to see whether any selected element already has 1 in it, and if so it sends the program back to line 820 until an array element is encountered with a 0. You can check whether your Smiley generator has worked by removing line 65 and replacing it with line 85:

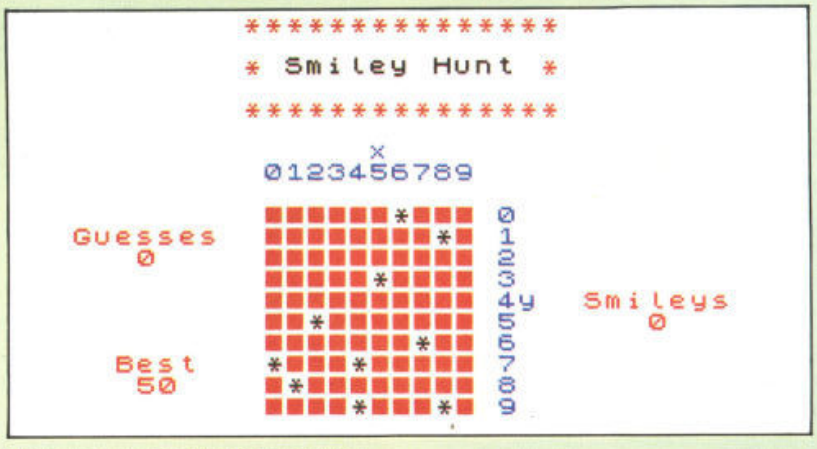
```
85 STOP
```

Now type in line 845, which is another temporary line to be removed later.

```
845 PRINT AT smileyy+9,smileyx+8;
INK 0;"*"
```

Now when you run the program you should find 10 asterisks in the grid boxes showing the locations of the hidden Smileys. You can leave this line in for a while as it will be useful later for testing purposes.

● ***Well, that's enough for one session. Next time we'll progress on to the input routine, and show how to check for any correct or incorrect guesses.***



The screen showing the hidden Smileys

Analysis

Do you find making decisions difficult? Can't you even decide on the answer to that question? Never fear, help is at hand with Decision Maker, the silicon version of tossing a coin.

Give information about the program to the user but not to the micro.

```
10 REM Decision maker
20 REM RAND and AT in action
```

Assign values to five numeric variables. These values are in fact line numbers and will be used in combination with GO SUB and GO TO commands.

```
30 LET screen=1000: LET question
=2000
40 LET answer=3000: LET anothergo
=4000: LET repeat=50
```

Calls the subroutine that handles the screen. This clears the display and prints out the possible answers. The variable is used as GO SUB *screen* is much easier to follow than GO SUB 1000.

```
50 GO SUB screen
```

Invokes the subroutine that lets the user type in a question.

```
60 GO SUB question
```

Has the subroutine at line *answer* working out which of the replies to select.

```
70 GO SUB answer
```

Stops the program crashing on into the subroutine definitions that follow.

```
80 GO SUB anothergo
```

Whistles up the code that checks if the user wants another go.

```
90 STOP
```

```
1000 REM displays answers
1010 CLS
```

Clears the screen.

Section of the program that puts the answers on the screen. These stay on screen while another part of the program moves the asterisk about.

```
1020 PRINT AT 1,2;"Definitely yes."
1030 PRINT AT 4,2;"Forget it."
1040 PRINT AT 7,2;"Play for time."
1050 PRINT AT 10,2;"Good
question."
1060 PRINT AT 13,2;"What do you
think?"
1070 PRINT AT 16,2;"Does it really
matter?"
```

Each of the answers is displayed. Their screen position is determined by using PRINT AT *y,x* where *y* is the row number (ranging from 0 to 21) and *x* the column number (0 to 31) of the first character in each answer.

```
1080 RETURN
```

Sends the program back to the main program. It takes things up again from the next line after the GO SUB that called the subroutine.

```
2000 REM ask question
```

Asks the user to type in the question and stores it in the string array *q$*. Try leaving out LINE and see what happens.

```
2010 INPUT "Type in your question",
LINE q$
```

Writes the question at the bottom of the answers.

```
2020 PRINT AT 21,2;q$
```

Deal with accepting the question.

```
2030 RETURN
```

```
3000 REM find answer
```

Moves the asterisk up and down the list of answers, eventually stopping by one.

```
3010 FOR n=1 TO 10
```

Forms a FOR . . . NEXT loop which cycles 10 times, moving the asterisk each time.

```
3020 LET y=1+3*INT (RND*6)
```

Calculates a new row number for the asterisk using RND to give a random number and INT to make sure that the answer is whole. The expression can only give the values 1, 4, 7, 10, 13 and 16 as results. Compare these with the row numbers of the answers.

```
3030 PRINT AT y,28;"*"
3040 BEEP 0.05,y
3050 PRINT AT y,28," "
```

Makes the noise. As *y* varies, so does the pitch of the note (measured in semitones above middle C).

```
3060 NEXT n
3070 PRINT AT y,28;"*"
```

Now you see it, now you don't. The asterisk appears and then disappears, obliterated by having a space drawn over it.

```
3080 RETURN
```

```
4000 REM finish or repeat
```

Asks the user for a decision and stores it in *g$*. Maybe the decision maker could be used to supply the answer!

```
4010 INPUT "Another go? Answer
Y/N", LINE g$
```

```
4020 IF g$="Y" OR g$="y" THEN
GO TO repeat
```

Determines whether the main part of the program is repeated.

A mugtrap that ensures all irrelevant replies are discarded.

```
4030 IF g$< >"N" AND g$<>"n"
THEN GOTO 4010
```

If the answer is Y for yes, the program is sent back to the first subroutine again.

```
4040 RETURN
```

By the time the program gets to this line *g$* must contain N or n.

Calls, jumps and loops

Continuing our exploration of the Z80 instruction set – Part Five

WE have already used the simple, unconditional CALL instruction. However it is possible to have a call which is conditional upon a flag, so that the call is only performed when the flag in question is set, or only when it is reset.

Similarly, there are conditional RET instructions, which test the condition of a specified flag and only perform the return if the condition is met.

This is the way a program makes decisions. By making a call or jump conditional, we can control the flow of a program, causing it to do different things under different conditions. To make a CALL instruction conditional, a few extra letters are added to the mnemonic:

CALL addr	Unconditional call
CALL Z,addr	Call if zero flag set
CALL NZ,addr	Call if not zero (flag reset)
CALL C,addr	Call if carry flag set
CALL NC,addr	Call if carry flag reset
CALL M,addr	Call if sign flag set
CALL P,addr	Call if sign flag reset
CALL PE,addr	Call if parity even (flag set)
CALL PO,addr	Call if parity odd (flag reset)

There are special call instructions called restarts – mnemonic RST – which can only be used to jump to a few specific locations at the beginning of the memory. These are used in the Spectrum ROM to call routines that are used a great deal, like the character printing routine, because the instructions only use one byte of memory. Each of the eight restart opcodes has a fixed call address, so no address needs to be specified.

A JP (jump) instruction is just like a CALL, except that it doesn't store a return address on the stack. Jumps may also be made conditional on the state of the flags. The mnemonics are as the list for the CALL instructions, but substitute JP for CALL.

Relative jumps

There is a special kind of jump called a relative jump – mnemonic JR. With this jump you do not specify an address to jump to, but the number of locations to jump, forward or backward. However since you have to give this number in a single byte, you are limited to 255 possible locations, so you can only jump forward by 127 or backward by 128. This number is called a displacement.

The forward jumps are easy to calculate. You just count the number of locations from the instruction (maximum &7F) that you want to jump. The location in the memory just after the displacement number is location zero in this count.

Jumping backwards is trickier, as the displacement has to be counted backwards from &FF. This is slightly awkward, as relative jumps are most often used to jump backwards, to perform small loops through routines that need to be performed repetitively.

In Program I the column at the right shows the hex number that would be entered as a displacement in the relative jump to jump to the memory location on that line.

Address	code		Displacement	
7CFF	00	←	F7	
7D00	00	←	F8	
7D01	00	←	F9	Jump below comes to here
7D02	00	←	FA	
7D03	00	←	FB	
7D04	00	←	FC	
7D05	00	←	FD	
7D06	18	←	FE	&18 is JR op-code
7D07	F9	←	FF	Displacement &F9 jumps back 7 places to &7D01
7D08	00	←	00	Counting from here – & 7D08
7D09	00	←	01	
7D0A	00	←	02	
7D0B	00	←	03	
7D0C	00	←	04	

Program I

In Program I, the memory is mapped in the same direction that our hex handler presents it, but some authorities would draw their diagrams the other way up. Now is as good a time as any to go into the top and bottom business. These terms are often applied to memory addresses, and it can be confusing.

When someone says that an address is near the top of the memory, does that mean it is near address zero, or address 65535? You might think it means that the top is at the beginning of memory, since when you step through the memory with our hex handler (or any memory listing utility for that matter) the earlier, lower number addresses are at the top of the screen.

However, it is more likely to mean that the lower number addresses are at the bottom, and the higher numbers are at the top. High at the top, you see? I try to avoid using the terms top and bottom, but when you do come across them, make sure you know which way round your interlocutor is facing.

As with the CALL and JP instructions, the JR instructions can be made conditional on the state of the flags.

There is a special loop-counting relative jump, the DNJZ instruction. When this is encountered, the B register is decremented, and the jump is performed only if the result was not zero (conditional on the zero flag being reset).

This makes it easy to set up a loop to be executed any number of times – up to 256 – just by loading the count into the B register.

The stack

We have already discussed the stack, and seen how CALL and RET instructions affect it. There are instructions for incrementing or decrementing the stack pointer, or loading it with the contents of the HL register, but these are best left alone unless you are certain you know what you are doing. It is quite easy enough to make a mess of the stack without manipulating the pointer!

PUSH and POP are the usual ways of getting a number on to or off the stack, but there is also an instruction, EX (SP),HL, which can be used if you should ever wish to exchange the number on the top of the stack with the HL register. You can also do this with the stack and IX or IY.

Block instructions

These are like little subroutines all in one instruction. They are used when the same manipulation needs to be carried out on a block of memory locations.

LDIR and LDDR are the block moving instructions, and the ones in this group that are most often used. First you load HL with a "source" address, DE with a "destination" address, and BC with a "count". The LDIR instruction will repeatedly move the byte from the address in HL to the address in DE, then increment both of these pointers, and decrement the counter in BC.

When BC reaches zero, LDIR stops its work and control passes to the next instruction in the program. The effect is to move a block of the size held in BC from the address held in HL to the address held in DE. LDDR is exactly the same, except that the two memory pointers are decremented instead of incremented, so they should be loaded with the end addresses of the blocks rather than the beginning.

CPIR and CPDR work in much the same way as the block moving instructions, but they are used to search a block of memory for a byte of a particular value. The start (or end, with CPDR) of the block is loaded into HL, the size of the block into BC, and the byte to search for into A. DE is not needed.

If a byte matching the contents of A is found, then the instruction finishes with HL pointing to the address after the one holding the byte with the zero flag set, otherwise it finishes when BC reaches zero and all the flags will be reset.

There are also non-repeating versions of the four instructions above, which allow the programmer to add more instructions of his own to make more complicated decisions before looping back to process another byte. The mnemonics are the same except for the omission of the R.

IN and OUT

The IN and OUT instructions are used for communication with peripherals. In the Spectrum this means things like reading the keyboard and cassette port.

The Spectrum does not use proper rigorous port addressing for its peripherals, so more than usual care must be taken in the selection of addresses for use with IN and OUT. The address is not decoded into a unique port number, instead a port is selected whenever a particular bit (or bits) of the address are active.

A careless choice of port numbers may activate peripherals you did not mean to disturb, giving unexpected effects, especially with microdrives. Reading any one of the eight keyboard port addresses, for instance, will also read some bits from the MIC socket.

With IN and OUT, the desired port address is put either in BC, in which case any register can be written to or read from the port, or the high byte of the address is put into A and the low byte follows the instruction opcode in the routine, in which case only the A register can be read into or written to the port.

This last method is a hangover from the earlier 8080 processor which only had single byte port addressing. The Z80 port can have a full 16 bit address the same as the memory, but remember that the Spectrum peripheral does not decode it properly.

There are Z80 IN/OUT instructions similar to the block handling instructions which could theoretically be used to read or write a number of adjacent ports in a systematic way. This might be very valuable in another larger system, but it's quite useless with the Spectrum's Mickey Mouse port decoding.

Interrupt handling

Interrupts are an extremely useful feature of most microprocessors, and the Z80 has a versatile set of interrupt facilities. The Spectrum only uses these in a very simple way but many programmers have found ways to add greatly to the Spectrum facilities by changing the interrupt handling.

In the Spectrum, there is a 50-cycle clock which is part of the hardware for timing the television frame scanning. This clock is also connected up to give the Z80's maskable interrupt pin a blip 50 times a second, and every time this happens, assuming the interrupts are enabled, the Z80 drops whatever else it may be doing and has a quick read of the keyboard.

This is called a maskable interrupt because it can be disabled by an instruction in software, to stop the CPU being called away in the middle of a sensitive job (like when it is loading or saving to cassette). There is also a non-maskable interrupt, but this is not used in the Spectrum, and cannot be useful without changes in the ROM. The mnemonics for enabling and disabling the interrupt are EI and DI.

There are three interrupt modes available in the Z80. The mnemonics for selecting the modes are IM 0, IM 1, and IM 2. Interrupt mode 0 is not used in the Spectrum. In systems where it is used, a peripheral may place a single instruction (usually a restart) on the data bus and force its execution with a signal on the interrupt pin.

The interrupt mode used by the Spectrum is IM 1, which is the simplest of the modes. In this mode, the Z80 always executes a Restart 0038, which calls the keyboard scanning routine at address &0038, whenever an interrupt is received, enable state permitting.

Interrupt mode 2 is the interesting one. In this mode the CPU forms an address using a byte previously placed in the I register for the high byte, and the byte currently on the data bus for the low byte.

It then fetches a new address from the memory at the address it has so formed, and this second address is the one it calls for the interrupt routine. This means that you could set up a whole page of addresses in a vector table – with the page address in I – and the peripheral would be able to control which address in the table the CPU would pick to call, by putting the low byte of the address on the data bus.

However IM 2 can't be used in exactly this way in

Some programming aids

As you see from the short examples in this series, writing a program in mnemonics is much less arduous than writing it in opcodes. You can remember the mnemonics more easily because they have a consistent form. LD A,&02 is much the same as LD HL,&02, even though the opcodes are quite different from each other.

If you spend a long time writing code in hex, you will get to the point where you can remember quite a few of the opcodes you use regularly, but you will still find that you have to look up most of them. The one basic mnemonic, LD, translates to nearly 100 different opcodes, and the same is true to a lesser extent for other mnemonics.

Obviously things would be a lot easier if you could just write the program in mnemonics and have done, and that is just what assembly language and assemblers are all about.

An assembler will process a text file filled with mnemonics into a machine code program in just a few seconds, or maybe minutes if it is an enormous program. Assemblers are actually just programs for looking up the opcodes that go with particular mnemonics, coupled to some calculation facilities to take the hard work out of figuring out all those addresses and relative jump displacements.

Now, you may have noticed throughout *The Complete Spectrum* that there are small differences between the assembler listings in different articles. This is simply because the editorial staff do not all use the same assembler program, and the different programs have different conventions about how labels and even opcodes should be entered. As with punctuation in Basic, sometimes these small differences are vital, not so much to the operation of the program, but to whether it will assemble properly.

When you write in assembler you are using a fairly dumb text editor, or a word processor, that hasn't much of an idea what an assembly program should look like. It will not pick up on your syntax errors every time you enter a line, like the Sinclair Basic does. It is not until the program is written and you try to assemble it that it spits out everything it doesn't understand with an error code.

These trivial differences in syntax become very important if you ever change from one assembler to another. I once went through a rather traumatic time when I changed from an assembler which didn't require a colon after every label to one which did.

For that reason, I would recommend that you use HiSoft's Devpak assembler package, which comes with an editor, disassembler, and monitor. It has its faults but they are few, and it is reasonably priced. But most of all, it is available on other computers, such as the Amstrad and CP/M systems, so you will be able to take your reflexes with you if you upgrade to another Z80-based machine. There is also a QL version, but that is bound to be a bit different, as there is no Z80 in a QL.

The Spectrum version I have will assemble files from tape or microdrive – useful when an assembler listing takes up too much space to get it all into the computer at once – but it doesn't work with disc.

If you are willing to do without a monitor and disassembler, you can get a much cheaper editor/assembler package from Picturesque. This also is said to work with microdrives, though I haven't tried this package.

The monitor and disassembler are not used so much in writing a program as in debugging it, and in investigating the workings of other people's programs. A monitor will usually let you single step through your code, examining registers and the like, to find out how you have gone wrong. Disassembly is the reverse of assembly, it takes hexadecimal opcodes and converts them to the more readable mnemonics. Not necessary, but highly educational.

Suppliers

HiSoft Devpak,
180 High Street North,
Dunstable,
Bedfordshire LU6 1AT.
(0582) 696421

Picturesque,
6 Corkscrew Hill,
West Wickham,
Kent BR4 9BB.

the Spectrum because the 50-cycle interrupt does not provide a predictable low order byte on the data bus and some peripherals, notably the Kempston joystick interface, may in fact place almost any number on the bus.

It is still possible to use IM 2 to capture the interrupts, but you have to use a vector table of 256 bytes which all point to the same address, so that whatever number is found on the data bus, the same address will be found for the interrupt routine.

This is the technique many programmers use for protection of commercial software, for interrupt driven music, and for extending the Basic language interpreter.

There are two special return instructions for use when returning from interrupt routines. RETI and RETN are used when returning from a maskable and non-maskable interrupt routine, respectively. In fact any return will work, but these make sure that the state of the interrupt enable is restored to the state it was in before the interrupt.

The final instruction associated with interrupts is HALT. This stops the microprocessor in its tracks until an interrupt occurs, which in the Spectrum will be every 50th of a second with the interrupts enabled, or never if the interrupts are disabled.

A non-maskable interrupt cannot be disabled, but this is not used in the Spectrum. The HALT instruction is used by the Basic when timing the PAUSE command.

And finally . . .

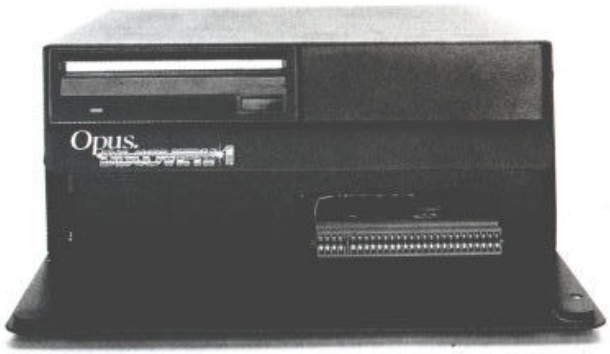
There are pairs of instructions for manipulating the carry flag. SCF sets the carry flag, and CCF complements (or inverts) it.

● ***In Part Six we will try some practical programming and find out more about using the Spectrum ROM.***

IT'S A DISCOVERY FROM ALL ANGLES

A FEATURE-PACKED SYSTEM FOR JUST £149.95 INCLUDING:

- 3.5" 250 K disc drive
- Double density disc interface
- Parallel printer interface
- Joystick interface
- Video monitor interface
- Peripheral through connector
- Built-in power supply
- Utilities on ROM including format and verify
- RAM disc facility
- Random access files fully supported
- Connections for second drive upgrade
- Comprehensive user manual



Whichever way you look at it, Discovery – the multi-feature Spectrum disc drive system from Opus – is a lot of hardware for your money, especially now we've made it even more affordable.

At an incredibly low **£149.95** it explodes the myth that you can't buy a full-blooded Spectrum disc drive for under £200. In fact to beat our price you might have to settle for a non industry-standard system – microdrive, waferdrive or even a 2.8" disc drive lookalike. Hardly worthwhile perhaps when for very little more Discovery offers not just an industry-standard 3½" 250 K disc drive but all these amazing features:

- **DOUBLE DENSITY DISC INTERFACE**
- **JOYSTICK INTERFACE**
- **PARALLEL PRINTER INTERFACE**
- **COMPOSITE VIDEO MONITOR INTERFACE**
- **BUILT-IN POWER SUPPLY FOR DISCOVERY AND SPECTRUM**
- **PERIPHERAL THROUGH CONNECTOR**
- **RAM DISC FACILITY**
- **SIMPLE ONE POINT CONNECTION TO THE SPECTRUM**

ONLY £149.95

Discovery takes no USER RAM from the Spectrum. Random Access Files are fully supported and formatting and back-up routines are included in ROM. You can also choose from a wide selection of 3½" disc software specially written for the Discovery games and business user.

Discovery's price of only **£149.95** even includes VAT, free delivery and a full 12-month warranty.

As you can see Discovery is much, much more than just a simple disc drive and it's available from all good computer stores nationwide. Call 0737-65080 for details of your nearest dealer or to order your Discovery direct.

Opus Supplies Ltd,
55 Ormside Way,
Holmethorpe Industrial Estate,
Redhill, SURREY RH1 2LW.

Opus.

NOW A NEW DISCOVERY FOR THE 128K SPECTRUM — CALL 0737 65080 FOR DETAILS

THE VOLEX TTX2000S

· FREE COMPUTER PROGRAMS · INSTANT SPORTS RESULTS · CONSTANT T.V. UPDATE ·

ALL FREE FROM ANY T.V. VIA THE VOLEX TELETEXT – TELESOFTWARE ADAPTOR

A WORLD OF INFORMATION AT YOUR FINGERTIPS

ONLY £99.99 Incl.V.A.T P&P

TELETEXT

is extra information available day or night whilst television transmitters are on air and without any charge and can now be enjoyed without the need of having a special Teletext T.V. set. All you need is to plug in your VOLEX TTX2000S Adaptor to receive 4-Tel from Channel 4, CEEFAX or ORACLE services.

TELESOFTWARE

is the name for computer programs which are broadcast as Teletext. Thus they may be loaded "OFF-AIR" into your Spectrum instead of being loaded from, say, tape or microdrive. 4-Tel is Channel Four's Teletext magazine and they are now broadcasting FREE telesoftware. Because of the special nature of teletext these programs are frequently updated – so you will not see the same thing each time you run the program.

EASY TO SET UP AND USE

As simple procedure tunes the VOLEX Adaptor to your local teletext channels. All loading of Telesoftware is via an index page on 4-Tel which the Adaptor will automatically find and display for you. Absolutely no modification is necessary to T.V. or computer. A comprehensive manual being supplied.

STORING PAGES

With the Volex Adaptor it is possible to send received pages to your printer or to Microdrive for storage and later recovery.

THE VOLEX TTX2000S

IS THE ADAPTOR APPROVED BY CHANNEL FOUR T.V.

Simply write out your order and post to
VOLEX ELECTRONICS, STOWELL TECHNICAL PARK,
ECCLES NEW ROAD, SALFORD, M5 2XH. TEL. 061-736 5822.

I enclose cheque/P.O. payable to VOLEX ELECTRONICS
OR charge my ACCESS/BARCLAYCARD/VISA number:

NAME

ADDRESS

TTX2000S

VOLEX ELECTRONICS
A DIVISION OF VOLEX GROUP p.l.c.

EVERY WILD IMAGINATION NEEDS A LITTLE SELF CONTROL.

As a Spectrum user you already enjoy some pretty sophisticated equipment.

So it's a pity if you are still missing that sophisticated, positive control your equipment deserves.

Let's face it, there's not much joy in a joystick, and keyboards can be all fingers and thumbs. Frustrating, especially if your imagination is much faster than your fingers!

What you need is an AMX Mouse.

COMPATIBLE WITH SPECTRUM 128

Already thousands of Spectrum owners have adopted an AMX Mouse and wouldn't be without it.

We didn't claim it was 'the best input device'. The press said it for us. In fact it has received outstanding critical acclaim, and no wonder!

The AMX Mouse brings to Spectrum users the same freedom and versatility which has, up to now, been the exclusive province of much more expensive computers. So, it's no surprise that nearly all the new 16 bit 'State of the Art' computers now come with a Mouse as standard. Proof, if proof were needed, that the Mouse is here to stay.

There are three superb programs included with the Mouse. AMX ART – Making full use of on-screen windows, icons, pull-down menus and pointers, you'll be astonished at the quality of the work you can produce, save and print using either ZX or Epson compatible printers. AMX COLOUR PALETTE – The wonderful pictures you create with AMX ART can be brought vividly to life with rich vibrant colours. AMX CONTROL – Now you can create a 'Mouse environment' in your own programs, AMX Control adds 28 commands to normal Sinclair Basic.

There is also a growing list of programs available from other leading software houses, which also utilize the Mouse, including, Artist II and the Writer from Softechnics, and Art Studio from British Telecom's Rainbird software collection, and many more titles will be available soon.

Isn't it about time you trapped an AMX Mouse?

The AMX Mouse package costs only £69.95 and includes the Mouse, interface which also includes a centronics printer port and a fully illustrated operating manual.

This superb package is available from all good computer dealers or direct, using the FREEPOST order form below.

☎ FOR INSTANT ACCESS/VISA ORDERS TELEPHONE (0925) 413501/2/3

PLEASE RUSH ME **POST FREE** _____(Quantity) AMX MOUSE SPECTRUM PACKAGE/S @ £69.95 EACH, INCL. VAT & P&P

I ENCLOSE CHEQUE/POSTAL ORDER FOR £_____

OR DEBIT MY ACCESS ☐ OR VISA ☐ (tick as appropriate)

CARD No. | | | | | | | | | | | | | | | | |

EXPIRY DATE _____

SIGNATURE _____

NAME (Block Capitals Please) _____

ADDRESS _____

_____ POSTCODE _____

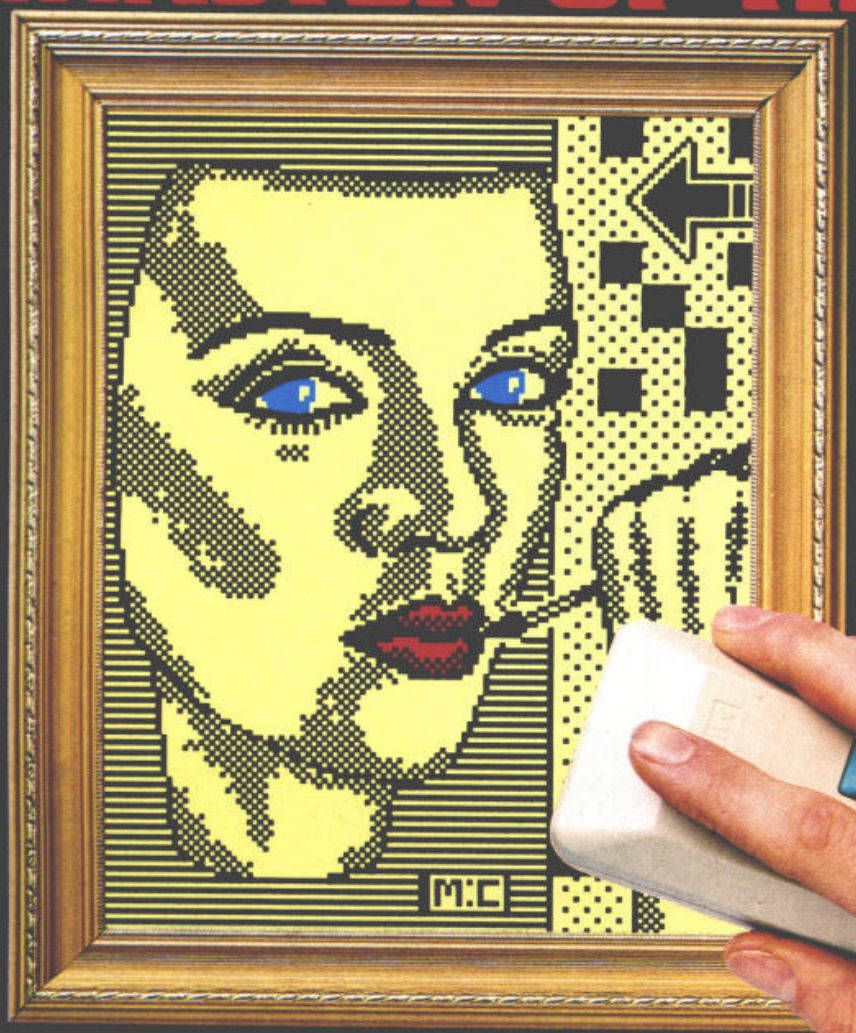
☐ PLEASE SEND ME FURTHER DETAILED INFORMATION ON THE AMS RANGE OF PRODUCTS (Tick if appropriate)

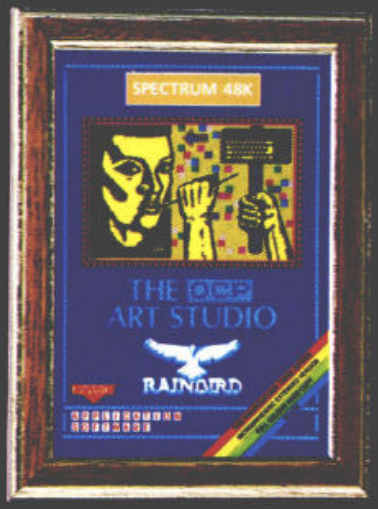
AMX MOUSE

IMAGINATION AT YOUR FINGERTIPS

SEND TO: ADVANCED MEMORY SYSTEMS LTD., FREEPOST, WARRINGTON WA4 1BR.

MASTER OF THE ART....





AT LAST A SPECTRUM GRAPHICS PACKAGE THAT IS FUN AND EASY TO USE. THE OCP ART STUDIO CONTAINS EVERY FEATURE YOU WILL NEED TO CREATE BEAUTIFUL ILLUSTRATIONS. IT WORKS WITH THE AMX MOUSE FOR EVEN GREATER EASE AND OUR HARD COPY OFFER MEANS YOU CAN HANG YOUR MASTERPIECE ON THE WALL

DO IT ALL – CREATE AN IMAGE. SHRINK IT, EXPAND IT, MOVE IT, ROTATE, COPY IT, COLOUR IT, SPRAY ON A PATTERN OR SHADE. MAKE ELASTIC LINES, TRIANGLES, RECTANGLES, CIRCLES – STRETCH AND MANIPULATE. ADD TEXT OR CHARACTERS, UP, DOWN, SIDEWAYS – ANY SIZE OR PROPORTION. ZOOM IN TO DRAW IN FINE DETAIL. SHRINK THE WHOLE PICTURE TO ADD BACKGROUND.

- Pull down menus. * Icon driven.
- Keyboard, joystick, mouse control.
- Dot matrix printer dumps, 5 sizes and grey scale – up to 80 columns.
- Supports 17 printer interfaces.
- 16 pens, 8 sprays and 16 brushes.
- 32 user-redefinable texture fills.
- Wash texture. * Undo facility.
- Snap facility. * Pixel edit.
- Cut, paste, turn, enlarge, reduce.
- Magnify (3 levels) pan and zoom.
- Text. 9 sizes, 2 directions, bold.
- Font editor with invert, rotate flip, clear, capture from window.
- Elastic line, triangle, rectangle.
- Low cost full colour prints offer.
- Upgrade offer. * Mouse offer.

SINCLAIR USER CLASSIC

"An extremely powerfull utility which should be of use to professional artists and designers as well as the home user"

THE OCP ART STUDIO £14.95

OR

EXTENDED ART STUDIO £24.95

For use with disc or microdrive only, and compatible with K DOS and SP DOS disc interfaces (supplied on cassette). Includes disc and microdrive operating systems, screen compression programme, four extra fonts and Kempston mouse compatibility (in addition to AMX). **(Available Mail Order Only)**

FOR 48K ZX SPECTRUM

RAINBIRD

THE OCP™ ART STUDIO

MAKE CHEQUES OR P.O. PAYABLE TO RAINBIRD SOFTWARE
BARCLAYCARD AND ACCESS ORDERS TEL: 01-240-8837 (24 HOURS)

RAINBIRD SOFTWARE BRITISH TELECOM
WELLINGTON HOUSE UPPER ST MARTIN'S LANE
LONDON WC2H 9DL
TEL: 01-240-8838

RAINBIRD is a division of British Telecommunications plc.

Combining numbers bit by bit

Part 4 of the series that helps to take the mystery out of the way your Spectrum operates

NOT too difficult

IN previous articles we've seen that binary numbers can be added and subtracted just as our more familiar decimal numbers are. And, of course, we can multiply and divide them.

There are, however, other ways of combining two binary numbers that are extremely useful in dealing with computers. They're also easy to use, so let's have a look at them.

Firstly, we'll see how we can NOT a binary number – simple, one-bit numbers first. By the way, we're going to be dealing exclusively with binary numbers this month, so we can drop the % sign. The rules for doing a NOT are simple:

If the bit is 1 then it becomes 0
If the bit is 0 then it becomes 1

If you like, the NOT converts a bit into its opposite.

So **NOT 1 = 0**
And **NOT 0 = 1**

Why do we use tne word NOT? Well, mathematicians often use the number 1 to mean TRUE and 0 to mean FALSE. So NOT 1 means NOT TRUE, which means FALSE, which is 0. That is, NOT 1 is 0. And, as NOT FALSE is most certainly TRUE, NOT 0 is 1.

If we are to NOT a binary number consisting of several bits, we simply apply the rule for NOT to each bit individually.

So **NOT 10110010**
becomes **01001101**

Some people think of this process as "turning the number on its head" – so it's sometimes called inverting. Others call it "taking the complement of the number".

AND quite easy

NOT just works on a single binary number. However, there are other sums or operations that have a set of rules for combining two binary numbers. For instance we can AND two binary numbers. Let's look at the rules for ANDing a single bit with another bit.

When you think about it, there are four possible combinations of bits that we could AND – 0 with 0, 0 with 1, 1 with 0 and 1 with 1.

We write that we are ANDing, say, 0 with 1 as 0 AND 1. The rules for ANDing are:

0 AND 0 = 0 (case a)
0 AND 1 = 0 (case b)
1 AND 0 = 0 (case c)
1 AND 1 = 1 (case d)

Notice that the only time the result is 1 (TRUE) is when the two bits ANDed are both 1 (TRUE). This helps us to see why we use the word AND to describe the operation.

If you think of the first bit as "this" and the second bit as "that", what we're doing when we're ANDing is asking whether "this and that" is true

"This and that" can only be true when both "this" is true AND "that", is true – hence the use of AND to describe the process. For example, consider the statement that it is:

dry AND sunny

This is true only if dry is true and sunny is true (case d). If either of the two (or both) are false (cases a, b, c) the whole statement is false, since it isn't both dry and sunny.

We can AND pairs of binary numbers of more than one bit – just apply the rules of ANDing to each bit individually. For example:

10010110
AND 10110011
gives **10010010**

OR even simpler

We can also OR two binary numbers. The rules for ORing a single bit with another bit are as follows (again there are four possible combinations):

0 OR 0 = 0 (case e)
0 OR 1 = 1 (case f)
1 OR 0 = 1 (case g)
1 OR 1 = 1 (case h)

In this case, you only get a FALSE result (0) when both bits are FALSE. If either or both bits are TRUE (1) the result is TRUE. It's easy to see why we use OR to describe this. If one, OR the other, OR both is true the whole thing is true!

Let's use the meteorological analogy again. Let's consider the statement that it is:

dry OR sunny

This is only FALSE when it is NOT dry and NOT sunny (case e), otherwise it is TRUE (cases f, g, h).

To sum up, with OR, the whole thing is true if either or both the things being ORed is true. As we did with AND, we can OR pairs of numbers with more than one

bit – we just apply the rules of ORing to each bit individually. For example:

```
          10010110
OR        10110011
gives     10110111
```

EOR – an exclusive

Let's continue our exploration of logical operations with a look at EOR.

EOR stands for Exclusive OR – sometimes people call it XOR. Either way it's the same thing. EOR is a variant on the way we normally use the term OR. For example, if I say:

Mike OR Pete wears glasses

this is true if Mike wears glasses, OR Pete wears glasses, OR both Mike *and* Pete wear glasses.

Now it's this last case of OR we're interested in, where they both wear glasses. EOR works just like OR up to this point. However, EOR does not "allow" both of them to wear glasses. Either one does, or the other, but not both.

To put it another way, the one who wears the glasses does so *exclusively*. If both are wearing glasses then while:

Mike OR Pete wears glasses

would be true,

Mike EOR Pete wears glasses

would be a downright lie!

We could signify that a statement is true with the letter T, and use F for false. At school our teachers used ticks for truth and crosses for false. Since we're using computers, though, we'll use numbers. 1 will denote true and 0 will denote false. We've chosen 1 and 0 because they fit in so well with the binary system.

So, in the above example, if Mike has glasses we can give Mike the value 1. If Pete hasn't glasses we can give Pete the value 0. Table I shows the idea, applied to each combination of spectacle wearer. The ones and zeros are known as truth values, states or conditions.

As you can see, there are four possible cases as far as Mike and Pete wearing glasses are concerned. Neither

	Wears glasses		
	Mike	Pete	
Case 1	0	0	neither wears glasses
Case 2	0	1	Pete wears glasses
Case 3	1	0	Mike wears glasses
Case 4	1	1	Both wear glasses

Table I: Mike EOR Pete wears glasses

can wear them as in case 1, where both Mike and Pete has 0 value. Then again, Pete may wear them (1) whereas Mike does not (0), case 2, and so on.

If you look carefully at the numbers involved in all four cases, you see that we've got four pairs of bits we can combine. Each pair of bits is made up of the "truth bit" for Mike and the "truth bit" for Pete.

Table II combines these pairs for all four cases in accordance with our OR rules. We've stored the result in a third column. We call such a table a truth table. In

Mike wears glasses	Pete wears glasses	Mike OR Pete wear glasses
0	0	0
0	1	1
1	0	1
1	1	1

Table II: Truth table for OR

this case, it's the truth table for OR. We can use it to work out the result for any OR combination of two bits. All we have to do is to find the row that starts with the two bit values we're combining and then look in the third column for the result.

Table III shows a similar table for:

Mike AND Pete wear glasses

Again the first two columns are identical, covering all four possible cases. The third column combines them according to the AND rules.

Mike wears glasses	Pete wears glasses	Mike AND Pete wear glasses
0	0	0
0	1	0
1	0	0
1	1	1

Table III: Truth table for AND

Look again at Table II. This corresponds in a sense to our binary rule for OR – you get a 1 if either or both bits you combine contain a 1.

However if when talking about Mike and Pete you mean OR in the exclusive sense, EOR, then the combination of Mike wearing glasses and Pete also wearing glasses would have to be false. This is because EOR means either one or the other wears glasses, but not both – it's *exclusively* one or the other.

If we do mean EOR in this exclusive sense we'd write our statement about them as:

Mike EOR Pete wear glasses

Its truth table is given in Table IV:

Mike wears glasses	Pete wears glasses	Mike EOR Pete wear glasses
0	0	0
0	1	1
1	0	1
1	1	0

Table IV: Truth table for EOR

If you look at each case, you'll see that the only time Mike EOR Pete is true is when either one or the other wears glasses, but not both (or neither).

More formally, if both bits are 0, or both bits are 1 the result is 0. If either is 1 and the other is 0 the result is 1. To put it another way, if the bits are identical the result is 0, otherwise the result is 1.

Let's have a look at how we EOR binary pairs of

numbers. It's the same as for OR and AND – just apply the rules for EORing to each pair of bits in succession. For example:

```
  10110110
EOR 11100101
gives 01010011
```

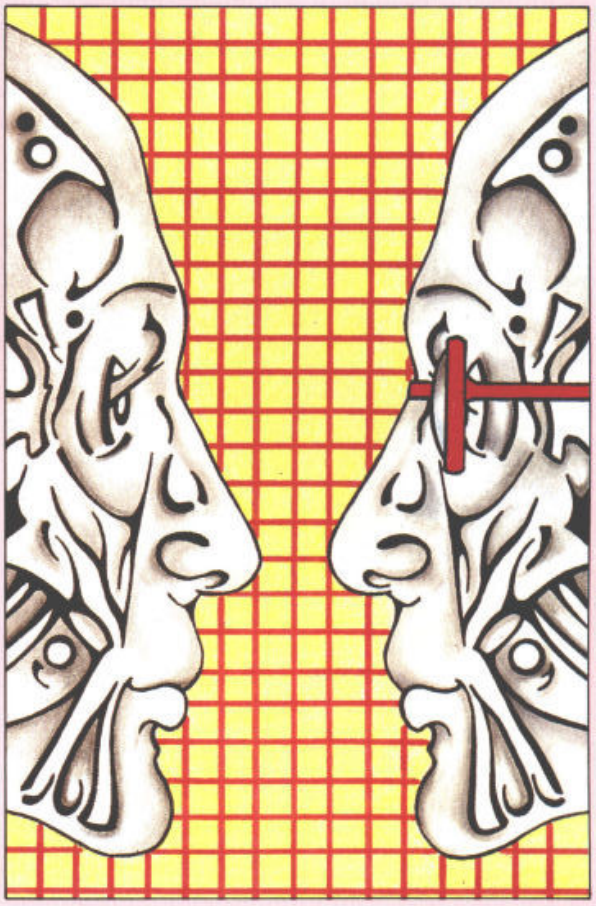
Take a look at what happens when you EOR a number with zero:

```
  10110110
EOR 00000000
gives 10110110
```

that is, when you EOR a number with zero it leaves that number unchanged. Also something interesting happens when you EOR a number with itself:

```
  10110110
EOR 10110110
gives 00000000
```

Whenever you EOR a number with itself, the result is zero. This is as it should be: remember, when you EOR two identical bits the result is zero.



Now you see it, now you don't

Now EOR has a property which makes it quite useful – let's look what happens when we take a number, EOR it with a second number and then go on to EOR the result once more with that second number.

First number	**10101101**
Second number	**EOR 01101000**
Result	**11000101**
Second number	**EOR 01101000**
Final result	**10101101**

As you can see, the first number has magically re-appeared! This always happens when you EOR twice with the same number as, in a sense, the two EORings cancel each other out.

Table V summarises the process for all four possible pairs of one-bit numbers. As you can see, for all the cases the final resulting bit (when the first bit has been EORed twice with the second) is identical to the first bit.

Another way to think of it is that we are doing:

first number EOR second number EOR second number

Taking the underlined part first, we've already seen that any number EORed with itself gives a zero result. So what we're really doing is:

first number EOR 0

which, as we've also seen, must leave just the first number, since EORing with zero leaves a number unchanged.

All this may seem rather abstruse, but actually it's quite useful. In fact we tend to use AND, OR and EOR quite often in graphics, particularly in animation. To simulate movement we frequently print something on the screen, then after leaving it there for a while to register on the eye, we blank it out and print it in a new position and so on.

Sometimes we blank the character out by printing it again in the same place but in the background colour. We can, however, use EOR. If we use EOR to place our character on the screen – never mind exactly how for the moment – when it comes to wanting rid of it, we can just repeat ourselves.

That is, we just EOR the character on again. As we've seen, the effect of two EORs is to cancel each other out. In this case, they cancel out to the original background – and the character disappears. Don't worry too much about the details, I just want to convey the general idea.

The point is, logical operators, as AND, OR and EOR are known, can be invaluable to both the Basic and machine code programmer.

First bit	Second bit	Result 1st EOR	Second bit again	Result 2nd EOR
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

Table V: EORing twice

Exploring the 128

We continue our assessment of the Spectrum's latest model

The Spectrum 128 can function in two modes. In 48k mode there is no difference between the 128 and the Spectrum Plus, but in 128k mode there are a few new facilities and one new keyword. In addition, the single key entry of keywords has been junked – you now have to spell out the keywords letter by letter.

The new functions are available as choices from a pair of menus. The main start up menu gives you a choice of an automatic tape loader, a tape volume tester, a trivial calculator function to turn your £180 computer into a £5 calculator, entry to 48k mode (with no way to return except reset) and finally, the 128k Basic.

When in 128k Basic you have a full screen Basic editor instead of the traditional two lines at the bottom. Once you get used to seeing the cursor at the top of the screen you can have great fun with the cursor keys, moving all over the program lines in any direction to make your editing changes. If you do have a keypad, you can use a wider variety of cursor movement controls to move (or delete) by a word at a time instead of single characters, for instance. But even without the keypad it is much easier to edit a program.

The Edit key is no longer needed to get a line down to the bottom of the screen for editing, so now it is used to call up the edit menu. From this you can select a renumber function although this is very limited as it only renumbers by 10, starting at 10, with no other choices.

There is a print function, which actually lists the program to the RS232 port – why not use LLIST? – and, of all things, a function to restrict program editing to the bottom two lines of the screen. Presumably this last is meant as a panic button for agoraphobics. This menu doesn't seem very well thought out, and it could easily have been replaced by a few new keywords.

Keypad

Sinclair has wisely decided to make this an optional extra in Britain, unlike Spanish retailers who inflict it on their customers willy-nilly. I see no reason to come up with the extra £20 to buy one. The keypad does have some extra editing keys not available on the keyboard, but it is otherwise a waste of space.

PLAYing around

The new keyword is PLAY, which is the command for both the sound chip and the Midi output. PLAY is followed by a string statement in the same way as a PRINT command, and the contents of the string dictate the output. The letters "a" through to "f" represent one octave of musical notes, and the capitals give the notes an octave higher.

Sharps and flats are indicated by the addition of hashes and dollar signs, and lengths of notes are indicated by a number representing a standard musical

interval, minim, crotchet, or whatever.

The whole thing is extremely powerful and easy to program. Music strings can be manipulated in the same way as other strings, and handed to the PLAY command as a string variable or array. All the special features of the chip are accessible via PLAY, including white noise and enveloping. The basic two-octave range can be shifted up and down the scale to cover about eight octaves.

To direct the music to a Midi peripheral instead of the built-in sound chip, you need only add a letter and number to the front of the string. This makes the changeover from chip to Midi a real doddle. Five out of the 14 pages in the introduction pamphlet are devoted to this command, which covers it pretty well.

The sound chip is one bit of hardware which Sinclair apparently failed to "lock out" of 48k mode (see below), but of course PLAY won't work in 48k mode. The fact that it is available in this mode is probably only of academic interest, but commercial programmers might put extra sound routines into 48k games to use it when it is there, so giving enhanced sound while still using 48k for compatibility with early models.

Compatibility

Despite the presentation of the Spectrum 128 as being two computers in one box, some changes have been made to the 48k mode ROM, so it is no longer exactly the same as the old Spectrum. The only time this is likely to cause trouble is with games that use a particular, rather draconian, protection method (checksumming the ROM), and a few that use the once empty block between &386E and &3CFF to vector the interrupts in interrupt mode 2.

Sinclair Research has gone to considerable lengths to retain compatibility with the old software, to the extent of purposefully not fixing any of the bugs from the old ROM, but even so the changes mean there are a few games that will not work on the new machines. The good news is that they claim that 97 per cent of the old stuff will run.

The other side of the compatibility coin is how many new programs are going to be written that actually use the big memory in 128k mode? The software houses have come out with a nice bunch of 128k releases to start with, but remember, anything they write with the new features will not be saleable to the millions of punters who already own the older models. Whatever they produce from now on will run on the 128, but it may well not use the big memory or the sound chip. It may even run in 48k mode.

The hardware compatibility picture is less clear. Sinclair has taken pains to "lock out" most of the new features when the computer is in 48k mode to avoid spurious effects from old software encountering unexpected new hardware.

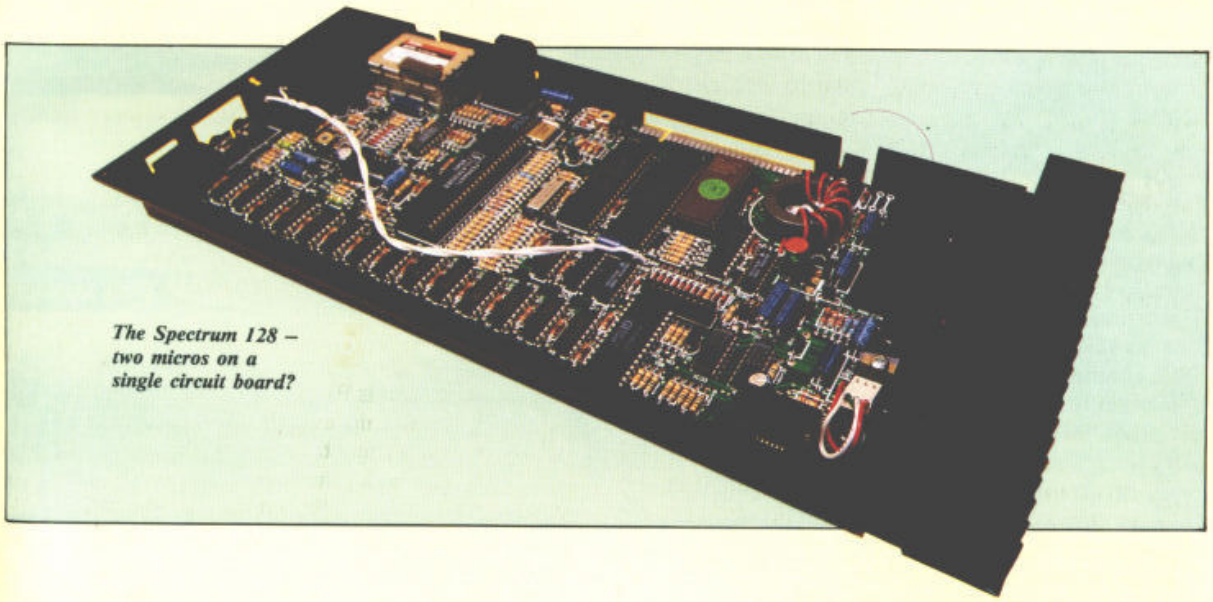
Apparently they have learned from all the trouble everyone had when they made a minor change in the ULA a few years back. Most add-ons will still work. Microdrives and joysticks in particular will still function in both 128k and 48k mode.

The casualties are likely to be the more complex non-Sinclair items. Before you buy a disc drive or modem set-up it would be well to wait for the dust to settle, and let the manufacturers sort out their patches. One such peripheral which won't live under the new regime is the VTX 5000 Prestel modem. This has a shadow ROM that takes control at power-up, and since the 128 powers up in 128k mode rather than 48k, the VTX is required to get along with the new half of the computer. This it will not do.

Conclusion

You get a lot more, for a lot more money – perhaps too much more money. The "real" Spectrum is still available, and the price is bound to come down now that a new model is out.

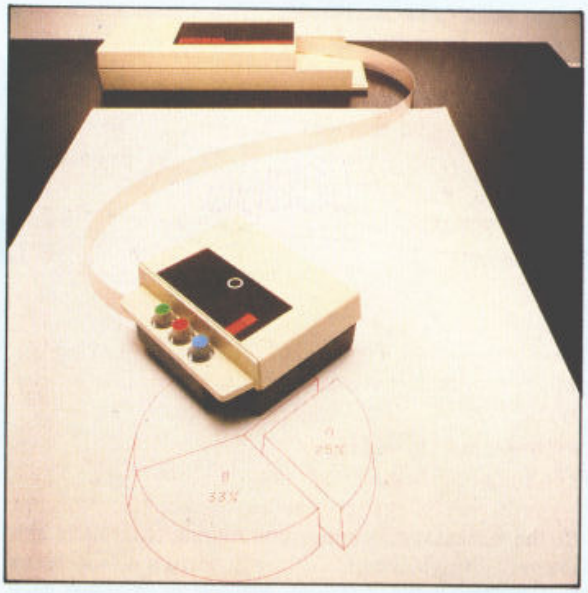
Some people may still be happy with their rubber key, vintage 1982 model, and won't bother to change, but new customers may well feel the top model is the only one to buy.



The Spectrum 128 – two micros on a single circuit board?

Put pen to paper

We take an in-depth look at one of the new breed of plotters



A PLOTTER is a computer-controlled device for making drawings on paper. It does for computer aided design what a printer does for word processing – by providing a hard copy.

Traditional plotters are bulky affairs requiring a framework surrounding a plotting surface the size of the piece of paper they are to plot on. Serious use generally dictates a paper size very much larger than one expects a printer to handle, so plotters can be very large indeed.

However, the Penman plotter described here is not the traditional style of machine, but more like a Logo turtle – in fact it can be used as a turtle as well as a plotter. This versatility, coupled with the small size and low price makes it much more appropriate for use with a home computer like the Spectrum.

The feature that makes the Penman much more than a turtle is the ability to sense the edge of the paper with great accuracy. Once it has located itself in the home position – by finding two edges of the paper with its photosensors – it can find any point on the paper in steps of a tenth of a millimetre. It does this by crawling across the surface of the paper using two wheels attached to very precise stepping motors.

The Penman's movements are controlled by its own on-board ROM, which interprets commands output from the Spectrum via the Interface 1 RS232 port. This can be done using the channels and streams, by opening a stream to the "t" (RS232 text) channel, and printing the commands to the plotter.

These commands are a string of letters and numbers and are fairly easy to set up. For instance, "DRM100,1200" sent through the RS232 link would cause the plotter to lower the pen and draw a line from its current position to a point 10mm to the right and 120mm up.

Ups and downs

In this command string "DRM" stands for "down pen and relative move". This command is like DRAW x,y in Basic. If the move is made with the pen up, and an A is substituted for the R to make it an absolute move, then no line will be drawn and it will be similar to the Basic command PLOT, x,y. Absolute moves are made in relation to the home position, while relative moves are made in relation to the current position of the plotter.

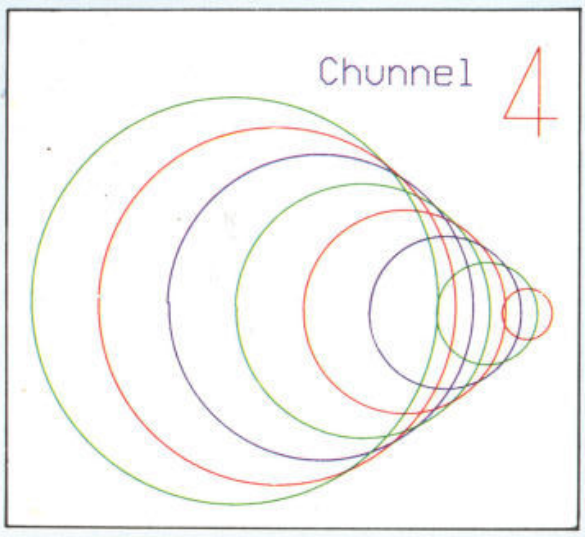
Other commands allow you to select any of the three different coloured pens or to draw curves and circles, or even to get the plotter to print text in any size from 1mm to 127mm. Text is simply sent to the plotter as a string and the characters are drawn by routines in the plotter's inbuilt ROM.

Further control options allow you to make the plotter behave as a turtle, or control the stepping motors directly in "robotic" mode. The plotter also sends information on its movements back up the RS232 link, so you could even use it as a mouse. These possibilities are for the more advanced user and may well involve you in writing your own software. In particular, it may not be possible to find a version of the Logo language for the Spectrum which is set up to use the Penman plotter as a turtle.

The Penman comes with a black perspex plotting surface, a few sheets of the recommended paper and a selection of colour pens. Replacement pens are easy to get as you can use Hewlett Packard plotter pens or just Pentel refills (available anywhere) with the supplied adapters. Most paper will work with the plotter, but you definitely get better results with the special "Mellotex" stuff.

A3 or A4 size paper is recommended, but the commands actually allow for control over much larger areas. The length of the ribbon cable that connects the plotter "turtle" to the control box might prove to be the limiting factor.

The 32-page manual is a nicely produced and very



```
1 REM Vector Hat Routine
10 OPEN#4;"t": OPEN#5;"b": PR
INT#4;"IHP2"
20 FOR j=0 TO 359 STEP 6
30 LET x=j/180*PI: PRINT#4;" U
A M ";6*(ABS INT (SIN x*128))+80
0;",";6*(ABS INT (COS x*80))+200;
",";" D R M ";6*INT (COS x*90);",
";6*INT (SIN x*65)
40 NEXT j
50 PRINT#4;"HP1,LVector Hat";#5;C
HR$ 10;#4;"V1.8"
```

complete piece of documentation, and there is a further set of technical update notes running to 15 photocopied pages. Five stars for the paperwork.

Software driver

It is possible, using Interface 1 extended Basic, to send all the necessary Penman commands to control the plotter through the RS232 port. However this does involve you in the learning of a new command syntax and the rewriting of screen graphics routines in order to transfer the graphics to paper. This process can be made easier with a piece of software from Softest.

The Softest Penman driver provides extended Basic commands to control the plotter with the more familiar Spectrum PLOT, DRAW, INK, LPRINT and CIRCLE commands. When this software is loaded, the normal screen graphic commands can be used, with the addition of an asterisk (*), to direct the output to the plotter rather than the screen. This makes it easy to rewrite your Basic graphic routines for hard copy, once they have been set up and tested on the screen, merely by adding asterisks to the Spectrum graphic commands.

The Softest driver's CIRCLE command in particular is much easier to use than the counterpart in Penman command syntax. This command, and the RS232 cable that comes with the Softest software, are enough to make the package a good buy.

This product is fairly new and there are still a few rough edges, but Softest seem very keen to get the thing right and actually worked over the weekend to fix a bug I had pointed out to them.

Problems

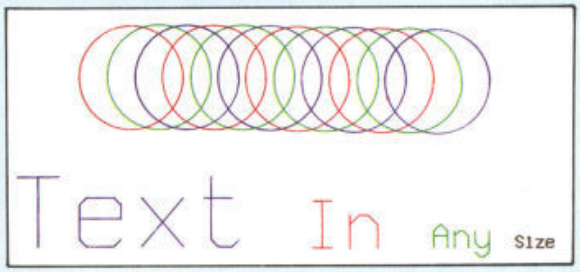
The Penman is not a product specifically aimed at the Spectrum, but meant to be used with any computer that has an RS232 interface. This means that you will still be able to use it if you should ever upgrade to another computer. However, it also means that no account has been taken of features that are peculiar to the Spectrum.

One of these affects the text printing. The Penman expects a line feed character, &0A, to be used at the end of a line of text, to direct the plotter to start a new line. Unfortunately, the Spectrum outputs a carriage return (&0D) instead. The plotter interprets a carriage return as an "End of Text" marker and returns to command mode, interpreting the next line of text as a command string. Naturally a muddle results.

It is possible to get around this by printing line feeds (use CHR$ 10) via the "b" or binary channel (the "t" channel won't send them), but this will only work with PRINT, not with LIST, which sends carriage returns regardless. So you won't be able to get pretty coloured listings from the Penman.

The other problem is accuracy. The Penman does not have a frame, but keeps track of its movements by counting its fractional wheel turns. Inevitably some error is bound to creep in after a session of crawling around on a piece of paper, especially if a wheel goes over the edge of the paper. When accuracy is important you must plan things so that the plotter never steps off the paper – probably easiest done by using large sheets – and you must "home" the plotter frequently during the plot so that it can reorient itself at the paper edges. The latest version of the plotter has an "autohome" feature which will take care of this, once enabled by the appropriate command.

Paper quality is very important, both for accurate movement and smoothly drawn lines. If the paper is too thin it can ruck up, and some photocopy paper has a low friction surface which causes wheelspin and consequent inaccurate movement. Running the wheels across fresh ink is also to be avoided, but there seems to be no way to plan around this.



Suppliers

Penman Products
8 Hazelwood Close
Dominion Way
Worthing
W. Sussex
BN14 8NP
Tel: (0903) 209081

Softest
10 Richmond Lane
Romsey
Hampshire
SO5 8LA
Tel: (0794) 513676

Cassette handling

All the facts you need to know about those cassette commands – at your fingertips

LOAD "" J key

Loads from cassette. Must be followed by file name in quotes, or just a pair of quotes – the empty or null string. Further specifiers for code or data loading optional. Basic programs load complete with variables.

MERGE "" T key, Ext mode

Merges a Basic program from tape with one already in memory. If there are any line numbers which are the same in both programs, the one from the tape will overwrite the one in memory. Won't work on machine code or data.

SAVE "name" S key

Saves program to cassette with file name *name*. Further specifiers for CODE and DATA may be used to save other than Basic programs. All saves must have a name enclosed in quotes. Basic programs are saved complete with variables.

System variables

23570: POKE with 16 to turn off cassette messages, POKE with 6 to turn them back on. Why? To stop the names of files from messing up a title page screen when loading multiple files. Also interferes with the screen listing of the program when editing, so be sure to turn it back on again. Does not deal with "Start tape and press any key" message when saving, unfortunately.

VERIFY "" R key, Ext mode

Like LOAD, except that it does not transfer program to memory, but reads the tape and compares it with the memory contents. Use after a SAVE to make sure program or code or data has saved properly. Also acts on variables when verifying Basic, which causes confusion.

LINE x 3 key, Ext mode

Placed after the name in a SAVE command to make the program run from line number x when it is subsequently loaded. Only works with Basic programs, not machine code routines.

CODE x,y I key, Ext mode

Used following name in save or load to specify a block of code instead of Basic program. The block runs from *x* and is *y* bytes long, but *x* and *y* need only be specified when saving.

DATA x() D key, Ext mode

Used following name to save or load the numeric array x(). Only arrays may be saved, not ordinary variables.

DATA x$() D key, Ext mode

Used following name to save or load the string array x$(). Only string arrays may be saved, not ordinary strings.

SCREEN$ K key, Ext mode

Used instead of CODE 16384,6912 to save or load the screen memory. Nice little shortcut.

USR "a",168 L key, Ext mode

Used with CODE to save the UDGs (168 bytes, beginning at the address returned by the USR function for user-defined graphic a).

The COMPLETE

SPECTRUM

With Part V of The Complete Spectrum, you can see how its comprehensive coverage makes it the ultimate reference work on computing with your Spectrum.
To keep your copies in perfect condition we have produced an attractive binder to hold all six parts. It is bound in smart black pvc, with the logo printed in silver. All you have to do before inserting the monthly parts in your binder is to remove the outer cover and the 'Special Offers' section in the centre.
For addresses in the UK, the binder costs £3.95, or £6.95 for overseas. If you missed Parts I, II, III or IV, or would like to subscribe for all the remaining parts, you can do so by using the order form in the centre of this issue.

Contents of Part I included:
● A step by step guide to your first 30 minutes using the Spectrum. ● Advice on saving and loading. ● How to avoid errors when typing in listings. ● An investigation of the Z80, the chip behind it all. ● Computer jargon clarified. ● And the start of our regular series – a simple guide to basic Basic, an introduction to machine code, and a how-to-do-it guide to graphics.

Contents of Part II included:
● Discovering discs. ● All about microdrives. ● Upgrading to 48k. ● Composite video. ● Graphics commands. ● Books. ● Animation techniques. ● And our regular series – basic Basic, machine code, graphics – continue to explain the mysteries of the Spectrum.

Contents of Part III included:
● The Spectrum 128. ● Printers and Interfaces. ● All about databases. ● Speech synthesis. ● Keyboard add-ons. ● Graphics commands. ● Animation techniques. ● And our regular series – basic Basic, machine code, and graphics – continue their investigation of the techniques of Spectrum programming.

Contents of Part IV included:
● Inside the 128 ● The Mouse ● Joysticks ● Graphics packages ● Spreadsheets ● Machine code graphics ● And our regular series – basic Basic, and machine code continue the exploration of your Spectrum's hidden secrets.

In Part Six..

We bring to a close our exciting investigation of the Spectrum world!

Adventures – what makes them such a popular pastime.
Top Adventures – a survey of some of the more popular games on the market.
Creating Adventures – writing your own or using a commercial product.
Classroom Spectrum – a look at the Spectrum's involvement in education.
Educational criteria – the rules for writing first class educational software.
Inside Basic – how those vital keywords are stored.

PLUS pages of hints and tips to help you get the best from your Spectrum.

On sale June 4

How does 20p a day turn your micro into a MAINFRAME?

Impossible you say! Just 20p a day for all that power?

No it's not a joke, nor a trick question. The answer is Micronet. The network that links your computer by modem to the most exciting interactive database in the U.K.

All for 20p a day subscription and the price of a local, cheap rate telephone call.*

So what does Micronet offer that has excited 20,000 people to join.

Well, for a start Micronet is an excellent way to keep up-to-date with the very latest information in computing. Our database is continually updated by our team of professional journalists who search out the stories behind the news.

But Micronet is much more than a news service – our revolutionary mainframe Chatlines give you the power to hold real-time conversations with up to 20,000 other users, instantly.

Our free national electronic mail system allows you to send and receive confidential mail to friends and businesses 24 hours a day.

You can even take part in the latest multi-user strategy games. Starnet for example, allows you to compete against hundreds of other "Star Captains" for domination of the galaxy.

Or win big weekly cash prizes by competing in the 'Round Britain Race' which makes use of the latest viewdata technology to challenge you to find secret locations.

Every day new *free* and discounted software is made available for downloading direct to your micro.

Teleshopping is the ultimate way to seek out high street bargains... or holiday and hotel bookings... computer dating... rail and airline information... Jobsearch... homestudy and schooling... ideas, information and entertainment facilities too numerous to list. As if all this wasn't enough you can also access Prestel's™ enormous database which is included in your subscription.

Micronet: the stimulating, challenging and informative way to add a whole new dimension to your micro. All you need is a modem. So cut the coupon today for your information pack.

For just 20p** a day can you afford to be left out in the power game?

micronet 800

MAKE THE RIGHT CONNECTIONS

*For 98% of telephone users.

Prestel is a trademark of British Telecommunications plc on Prestel.

Micronet 800, Durrant House, 8 Herbal Hill, London EC1R 5EJ. Telephone: 01-278 3143.

To: Micronet 800, Durrant House, 8 Herbal Hill, London EC1R 5EJ. Telephone: 01-278 3143.

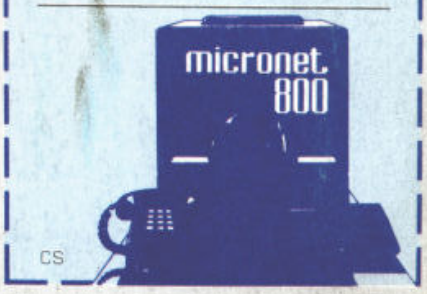
Please rush me the full facts about Micronet and tell me how I can turn my micro into a mainframe for 20p per day. (£16.50 per quarter**).

NAME

ADDRESS

TELEPHONE AGE

MAKE AND MODEL OF MICRO



CS