The COMPLETE

PART 4

SPECTRUM

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

A Database Publication

Spreadsheets – and how to choose them

All about joysticks

Mouse power harnessed

Inside the 128

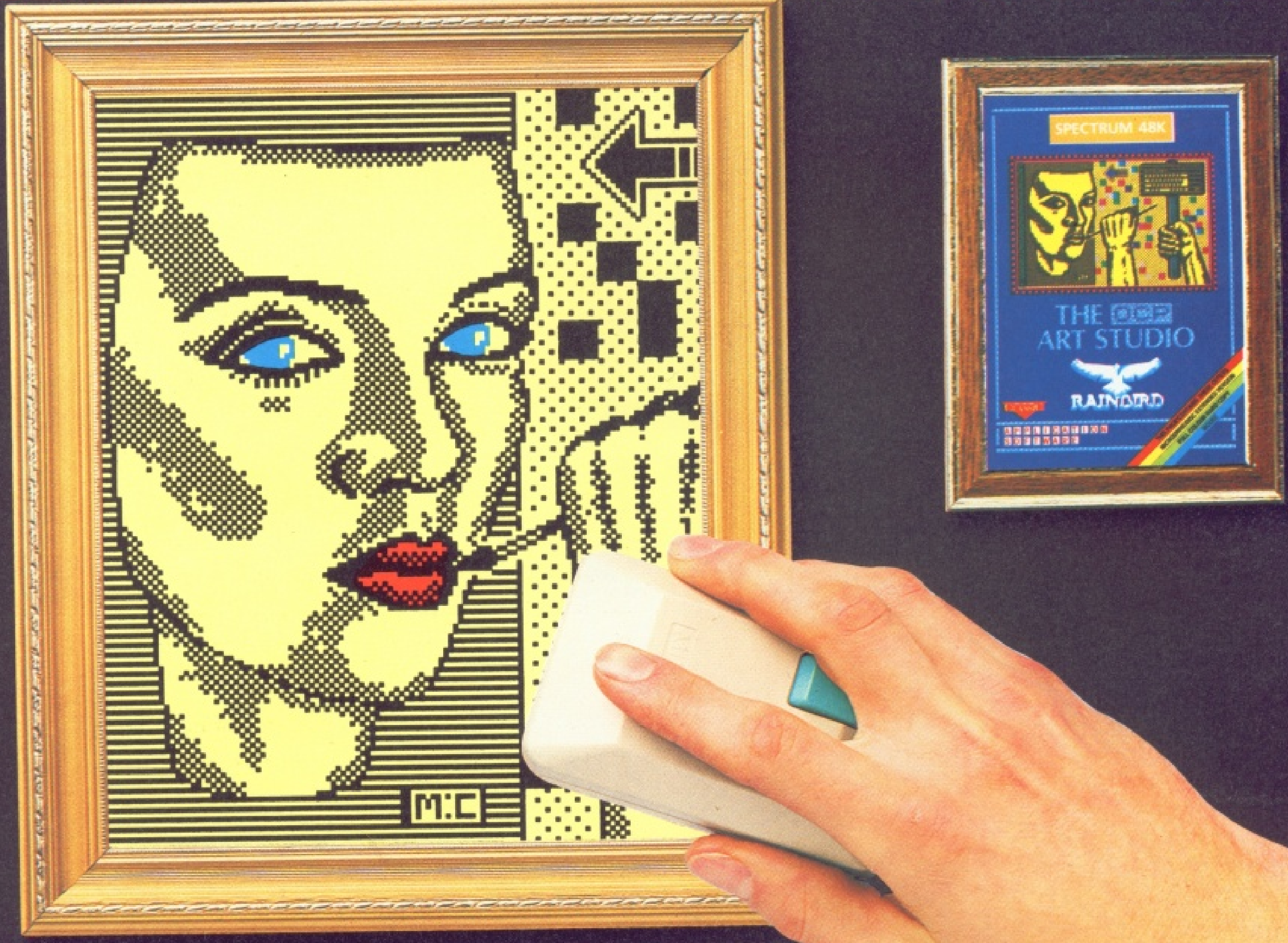
+graphics, Basic and Z80 assembler

BARGAIN OFFERS INSIDE
SAVE £££s ON MANY MORE SPECTRUM PRODUCTS

In six monthly parts

£1.50

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"

RAINBIRD

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

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.

In Part Four...

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.

In the FOR...NEXT world

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

EVEN though we've stuck to basics – or stuck to Basic – we've come a long way in our exploration of elementary Spectrum programming techniques.

First we looked at simple PRINT statements, used to put numbers on screen and do sums. Not content with this we went on to LET and saw how we could use it to create variables – labels that stand for numbers and strings.

From here we moved on to easy programs, lists of numbered statements that the Spectrum holds in its memory, unleashed by the command RUN. And, bored by the fact that these programs just consisted of one statement after another, we entered the world of FOR . . . NEXT loops. It's this world that we'll be exploring in greater detail this time.

And NEXT, FOR

Armed with all your new-found knowledge, it should be easy for you to see how Program I works. However it does hold one mystery. Can you spot it?

```
10 REM Program I
20 INPUT "How many times round the
loop?",number
30 FOR a=1 TO number
40 PRINT "Loop number";a
50 NEXT a
```

Program I

The puzzle lies in the REM of line 10. The answer is that REM is another keyword, one of the words of power the Spectrum obeys. And very useful it is too. And the reason it's useful is that it does nothing!

If this sounds a bit strange, don't worry – it's quite simple. All the REM does is to tell the micro that whatever comes after the REM is to be ignored and it might as well go on to the next line and deal with that.

This allows you to put anything you want after a REM, secure in the knowledge that the Spectrum will ignore it. And this is an extremely valuable facility, allowing you to annotate your program with REMarks.

In this case the remark is just the program name and is used to help you find the listing we're talking about. As you get to write longer, more complicated programs you'll find that peppering the listing with a liberal sprinkling of REMs helps to make them easier to understand. If you're hunting for REM, you'll find it on the E key.

Let's see what happens when you RUN Program I. The Spectrum goes to the lowest line number (10) and obeys that. In this case it finds the keyword REM so it ignores the rest of the line and goes on to the next, line 20.

Here the INPUT command puts a message on the screen, asking how many times you want the loop to cycle. The program is halted until you reply, when the figure is put into the variable *number*.

Once this has been done the program enters the FOR . . . NEXT loop formed by lines 30 to 50. This has control variable *a* which takes values from 1 to whatever *number* is. Each time round the loop the PRINT of line 40 displays the loop number.

Don't just read about it, try it out for yourself. See what happens with different values for *number*. Try having more than one line inside the FOR . . . NEXT loop. Experiment.

Screens and Scrolls

While you've got Program I in your micro, try running it and giving *number* a large value, such as 100. Now you get messages from:

```
Loop number 1
```

to:

```
Loop number 22
```

appearing on the screen and then you get:

```
scroll?
```

The reason for this is that the screen is full – the Spectrum has displayed as many lines as it can. To display any more it will have to add one to the bottom moving all the other lines up one.

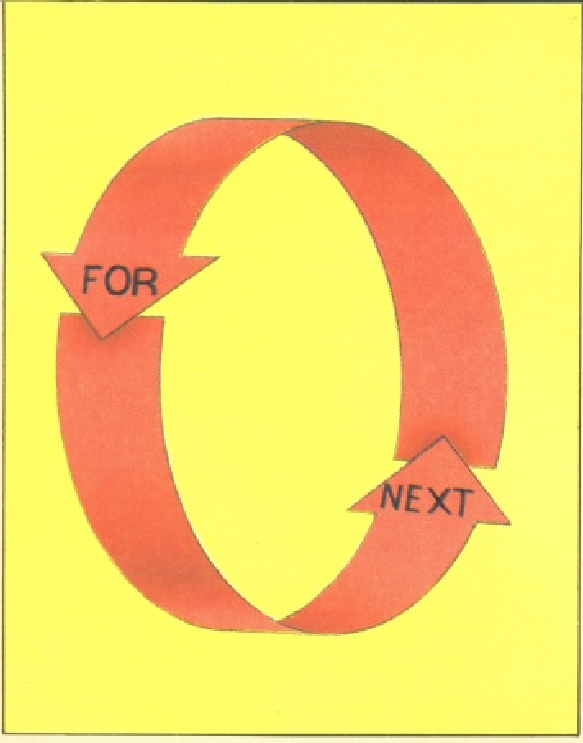
This scrolling, as it's known, has the effect of removing the top line for each line added at the bottom. Because of this potential loss of information the Spectrum asks you if you want scrolling to take place.

If you do, just press Y or Enter and you'll get another screenful of output. In this case you'll find the messages from 23 to 44 on the screen along with another request for permission to scroll.

Should you not want the screen to scroll, pressing N or Space will stop the program. At this point you can change your mind and have the program continue by using the CONT function (on the C key). Or you can get a listing of your program by pressing Enter by itself.

And, should all these messages and listings make your screen look untidy, you can always clear the screen with CLS (on the V key). Notice, however, that this only clears the screen. The program is still in memory as you'll see if you LIST it (K key) or RUN it.

If you want to get rid of the program completely, just use the command NEW (A key). This wipes the program from memory, so only use it when you're sure you want to.



One more STEP

Have you noticed anything about the control variable in our FOR ... NEXT loops? It always goes up in steps of one. In Program I, the first value of *a* is 1, the second is 2, and so on until *a* is equal to one more than *number* and the loop stops. This doesn't always have to be the case as Program II shows.

```
10 REM Program II
20 FOR a=1 TO 10 STEP 2
30 PRINT a
40 NEXT a
```

Program II

Apart from the STEP at the end of line 20, this program should be fairly straightforward. When you run it you'll find that you're rewarded with:

```
1
3
5
7
9
```

What's happened is that for each repetition of the loop, line 30 has printed out the value of the control variable, *a*. Previously the control variable of our FOR ... NEXT loops has gone up in steps of 1. Now, however, the keyword STEP (D key) tells the Spectrum to increase the control variable by 2 at the end of each loop.

This means that *a* takes the values 1, 3, 5, 7, 9 in turn as the loop cycles. After 9, *a* gets increased to 11. This is greater than 10, the outer limit of the loop set in line 20, so the loop stops cycling. As there are no more lines the program stops.

The number after STEP can be any you like – within reason. Try out 3, 4 and 50 and see if you understand what's happening. And don't restrict yourself to whole numbers, you can have fractional steps as well. You'll see this if you use:

```
20 FOR a=1 TO 10 STEP 0.5
```

By now you've probably realised that lines like:

```
30 FOR a=1 TO number
```

in Program I are in fact short for:

```
30 FOR a=1 to number STEP 1
```

When the step is only 1, the Spectrum takes it for granted. Or, in other words, the default value of STEP is 1.

Program III shows a very simple use of STEP and a FOR ... NEXT loop as it adds the odd numbers between 1 and 10.

```
10 REM Program III
20 LET total=0
30 FOR a=1 TO 10 STEP 2
40 LET total=total+a
50 NEXT a
60 PRINT total
```

Program III.

Here the numeric variable *total* is set to 0 in line 20 and then used inside the loop to keep a running total of the various values of *a*. As the STEP of line 30 ensures that *a* goes from 1 to 9 in steps of two, the result is that when the loop finishes *total* holds the sum of the odd numbers between 1 and 9. This is printed out by line 60. Can you alter the program so that it gives the total of the even numbers between 1 and 10?

Stepping down

While you've been messing around with different values for the STEP parameter of a FOR ... NEXT loop, did you try any negative values? If so did you understand what was happening? Have a look at Program IV which has a step of –1.

```
10 REM Program IV
20 FOR a=10 TO 1 STEP -1
30 PRINT a
40 NEXT a
50 PRINT "LIFT OFF!"
```

Program IV

When you run the program you'll see a countdown from 10 to 1 ending in the LIFT OFF message when the program leaves the loop and comes to line 50.

It's easy to see how it works. The step of –1 reduces the value of the control variable, *a*, by one each time round the loop.

Notice, however, that the numbers in the FOR and TO part of the loop definition have to be in the right order. As the control variable is decreasing it has to go from a larger number TO a smaller.

Try changing line 20 to:

```
20 FOR a=10 TO 0 STEP -1
```

which takes the countdown to zero. Or:

```
20 FOR a=10 to -3 STEP -1
```

which takes it below zero. And notice that the number 0 is not the same as the letter o.

Once again, don't stick to negative steps of one. Try other negative values and see what effect changing the limits of the loop control variable has on the output.

Mis-steps

While the STEP command is easy to use and to understand, you can occasionally come across some problems with it. Having said that, Program V should cause you no problems, provided that you type it in correctly.

```
10 REM Program V
20 FOR a=1 TO 10
30 PRINT a
40 NEXT a
```

Program V

Now try changing line 20 to:

```
20 FOR a=1 TO 10 STEP -1
```

and see what happens. The answer is nothing.

The reason for this is that you're telling the Spectrum to enter a FOR . . . NEXT loop whose control variable will be going down in steps of minus one. So far so good. The trouble is that line 20 also says that while this is happening, the control variable will go from 1 to 10. Obviously this is impossible and something has to give. The Spectrum, faced with a potentially embarassing situation, saves its face by pretending that the loop doesn't exist.

You'll get the same thing with lines like:

```
20 FOR a=1 TO 0
20 FOR a=-10 TO -5 STEP -1
20 FOR a=10 TO 4
```

While it's fairly obvious what's wrong in cases like these, it's a lot harder when you have lines that use numeric variables such as:

```
20 FOR a=oneNumber TO anotherNumber
STEP number
```

So if a FOR . . . NEXT loop appears to do nothing, have a close look at the limits given to the control variable and the steps it takes.

Nested loops

By now the idea that a FOR . . . NEXT loop encloses a set of lines to be repeated should come as no surprise. What, however, if these lines themselves contain a FOR . . . NEXT loop? Can this work? The answer is yes, provided you keep a close eye on the loop control variables. Program VI shows how it's done.

```
10 REM Program VI
20 FOR a=1 TO 3
30 FOR b=1 TO 3
40 PRINT "a is ";a,"b is ";b
50 NEXT b
60 PRINT
70 NEXT a
```

Program VI

Here there are two FOR . . . NEXT loops. The first is formed by lines 20 and 70 and has the control variable *a*. The second FOR . . . NEXT loop is formed by lines 30 and 50 and has control variable *b*.

The first loop is called the outer loop, the second loop (inside the first) is called the inner loop. The pair are called nested loops – one is nested inside the other. So long as one loop is contained entirely within the other, that is there's no overlap, the program will work.

The first time the program enters the outer loop, *a* takes the value 1. The program then goes on to line 30 and here the micro is told to perform another loop with control variable *b*, ranging from 1 to 3.

So *b* takes the value 1 and the program goes on to line 40, printing the message telling you that *a* is 1 and *b* is 1. Now the program comes to line 50 which adds 1 to *b* (making it 2) and sends the micro back to line 30.

This time line 40 proclaims that *a* is 1 but *b* is now 2. Then on to line 50 again where the NEXT sends the program back to 30 after incrementing *b*, 40 says that *a* is 1 and *b* is now 3, and back to line 50 again.

Now the program "drops out" of the inner loop as its conditions are satisfied, and goes on to the next line. This just prints a blank line for neatness' sake.

Line 70 then adds 1 to *a*, which is now 2, and sends the program back to line 20 as the outer loop still has to run. It only finishes when the outer loop has cycled three times. And each time the outer loop cycles once, the inner loop cycles three times.

It's rather like the trainer of a football team who has his players running round the pitch. He might have them do it, say, 10 times. Then, cruel beast, he might decide that each lap they have to do 10 pressups. For one circuit of the pitch (the outer loop) they do 10 pressups (inner loop). So for every one cycle of an outer FOR . . . NEXT loop, the inner loop goes through all its cycles. Program VII should make it clearer.

```
10 REM Program VII
20 FOR a=1 TO 5
30 PRINT "Outer loop number ";a
40 PRINT
50 FOR b=1 TO 4
60 PRINT "Inner loop number ";b
70 NEXT b
80 PRINT
90 NEXT a
```

Program VII

Don't worry about the PRINTs – they're just there to make the screen more legible. From the output you should see the inner loop performing all its turns for each time round the outer one. Once you've got the idea of nested loops, see what happens if you confuse your NEXTs, getting your control variables in a twist. Try changing the lines in Program VII to:

```
70 NEXT a
90 NEXT b
```

As you'll see, the program works but badly. It's left

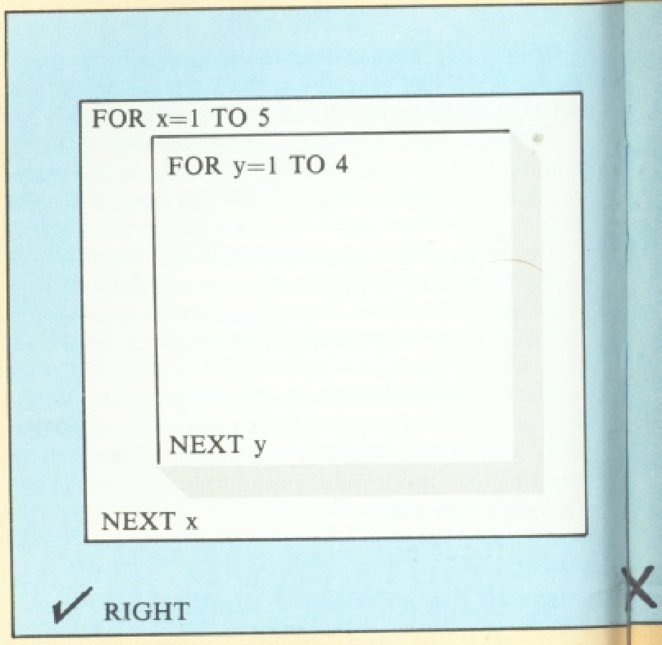


Figure I: Crossed loops

to you to figure out what's happened. The moral is, always make sure that the control variables in the FOR part of the loop match up with those in the NEXT part and that loops don't overlap. Figure I shows the idea.

Nested loops in action

Nested loops are very powerful programming structures as Program VIII shows.

```
10 REM Program VIII
20 FOR a=1 TO 10
30 FOR b=1 TO 12
40 PRINT a;" times ";b;" is ";a*b
50 NEXT b
60 PRINT
70 NEXT a
```

Program VIII

Although only seven lines long (and one of them does nothing, being a REM) it produces all the times tables from one to 10.

This is done by having the outer loop cycling 10 times. For each time round the outer loop the inner loop cycles 12 times. By multiplying the control variables in line 40 the times table appears.

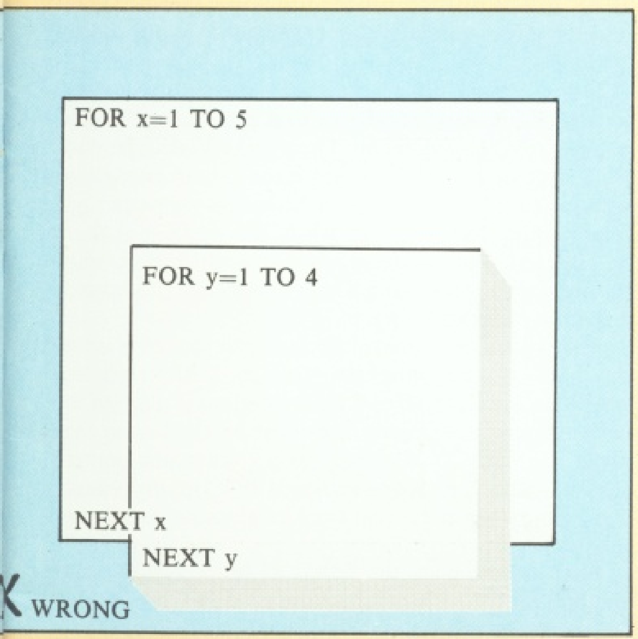
One time-honoured way of understanding nested loops is to use them to create a triangle of asterisks. Have a look at Program IX:

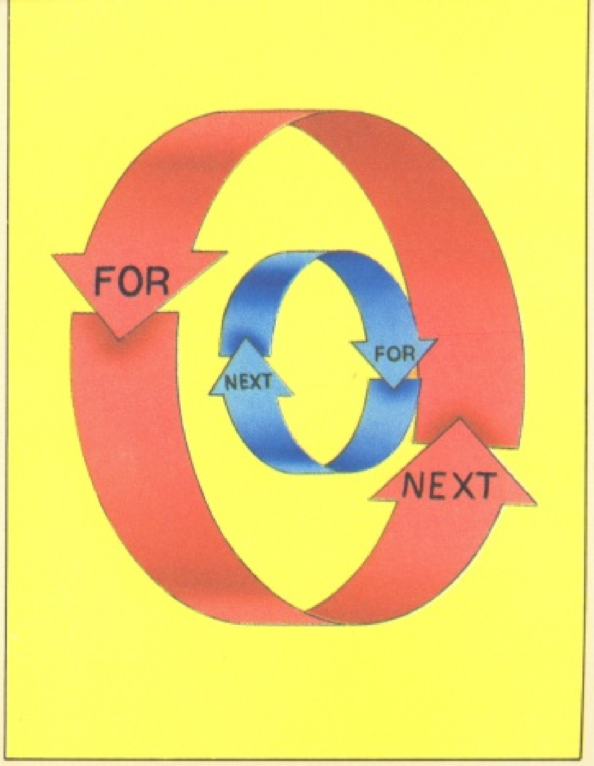
```
10 REM Program IX
20 FOR a=1 TO 5
30 FOR b=1 TO a
40 PRINT "*";
50 NEXT b
60 PRINT
70 NEXT a
```

Program IX

This consists of two nested loops with control variables *a* and *b*. The first loop ranges from 1 to 5. The second loop is more cunning – its control variable goes from 1 to *a*. The first time round the loop both *a* and *b* are 1.

The result is that the inner loop cycles once, line 40 printing just one asterisk. The next time round the loop





a is 2. This time the inner loop cycles twice so two asterisks appear. As *a* increases so does the number of cycles of the inner loop. The result is a triangle of asterisks:

```
*
**
***
****
*****
```

Try changing line 20 of Program IX to:

20 FOR a=5 TO 1 STEP −1

Can you see why this negative step produces this triangle?

```
*****
****
***
**
*
```

If you really want to test your knowledge of nested loops try altering line 40 of Program IX to:

40 PRINT TAB (5–a+b);"*";

and explaining the resulting triangle:

```
    *
   **
  ***
 ****
*****
```

and then add:

20 FOR a=5 TO 1 STEP −1

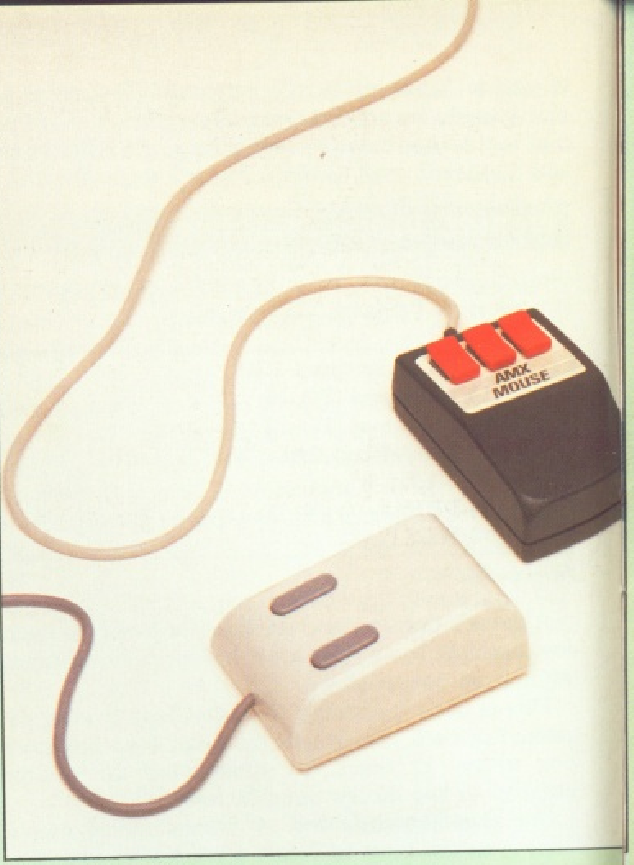
and see why the result is:

```
*****
 ****
  ***
   **
    *
```

By the time you've understood how these triangles are created you'll be ready to use nested FOR . . . NEXTs in your own programs. Good looping!

Mouse power!

What to look out for when you go mouse hunting



THE computer is now an accepted part of our everyday lives, both at work and at play. Pay slips, tax returns, stock control and invoices are all liable to be computer generated, not to mention the millions of letters that are composed and produced using word processors.

Many people who use computers did not grow up in this computerised environment and quite understandably do not always find them easy to get along with. To make computers more friendly, computer and software manufacturers have introduced a number of facilities to make the user's life easier.

One of the latest innovations is the mouse. This is a small device, about the size of a pack of playing cards, attached to the computer by a light, flexible cable – the tail!

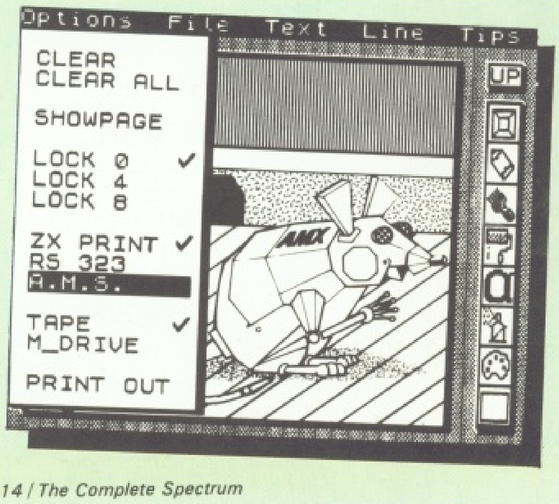
The idea behind a mouse is not only to make various operations with the computer easier to achieve but also to eliminate as many user errors as possible.

A pointer shown on the display is guided, using the mouse, to various icons or words displayed on the screen. A button on the mouse is pressed to action the command chosen.

Sometimes the initial command will cause a list of further options to be displayed within a separate box, or window. These Windows, Icons, Mice and Pointers are known by the acronym WIMP.

The mouse contains a ball that revolves as the mouse is moved across a flat surface. This ball drives two rollers that in turn rotate small discs. These are indirectly the source of a series of electrical impulses that effectively define the X and Y coordinates of the position of the mouse as it is moved around.

The pulses are derived either optically – with the disc having a series of slots or mirrors – or magnetically where the disc has several small magnets imbedded into its structure.



Two companies have recently introduced mice for the Spectrum, so now you too can enter the world of the WIMPs. Advanced Memory Systems offer us the AMX Mouse and Kempston Micro Electronics the Kempston Mouse.

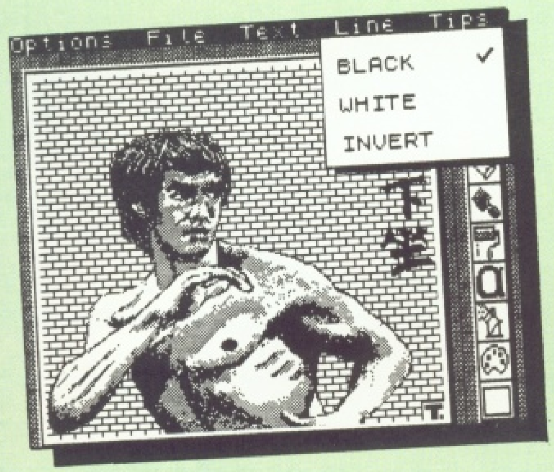
The AMX Mouse comes with a graphics designer package, the software to enable you to incorporate WIMPs within your own programs, an icon designer and a couple of demonstration programs. The Kempston Mouse at this stage only has a graphics designer program and a very simple example of how the Spectrum can recognise the position of the mouse.

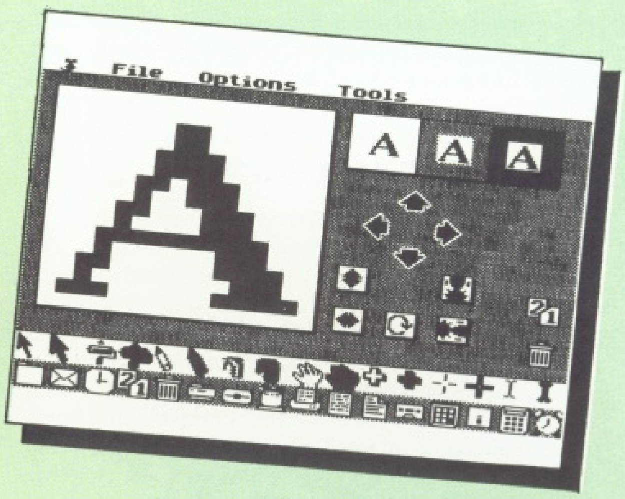
Kempston are preparing their own control software for their mouse, to be used along the same lines as the AMX. This will be supplied free of charge to all registered users of the Kempston Mouse as soon as it is available.

Both mice are similar in size and are both very easy to use. The Kempston Mouse uses a rubber roller ball and the AMX a nylon one. The nylon ball does tend to slip a little on a very smooth surface but not enough to worry about. The Kempston Mouse is supported on four captive ball bearings while the AMX has four nylon studs. Both systems allow smooth control, with the Kempston generating a little noise when it is moved around a wooden desktop.

Both come with a neat interface unit that plugs into the back of the Spectrum, though neither has an extension socket. So if any other interface is planned to be used, the mouse interface must be the last in the chain. The AMX unit also has a Centronics printer socket – this looks like a standard BBC Micro printer socket but the instruction booklet gives no details on this point.

Each mouse gives the feeling of being well thought out and both have a graphics designer package to give you some experience in using the mouse. So let us look at how good these are.





AMX Art

There is no doubt that this program helps the user in learning about the facilities that a mouse has to offer. Sadly, as a graphics design program it is not too hot.

It's fine if all you want to do is draw pretty pictures on the screen but if you want to include these pictures in your own programs there are several problems. Not the least is getting the colours in the right places.

The instructions generally are a little thin and have several small literal mistakes that may lead one to question the accuracy of the rest. On page 9 for instance, under IMPORT, the word "impossible" is used instead of "possible"!

Once loaded, AMX Art does indeed show you the ease which a mouse can bring to such a program. On the right are a series of icons, depicting things like a pencil, an eraser, a paint brush, a spray can, and so on.

Move the mouse to position a pointer over the one you want, press a button and move the mouse to position a new pointer – shape dependent upon your choice – out over the work area.

Along the top of the display are OPTIONS, FILE, TEXT, LINE and TIPS. Moving the mouse to action one of these initiates a pull-down menu from which other choices may be made. The icons on the right and the pull-down menus are the heart of the mouse system. Trying to illustrate these advantages inevitably leads us to discuss some of the disadvantages of the graphics design program.

Because of the icons down the right hand side you do not have a work area showing the full width of the screen. Save this as a SCREEN$ file and there will be a blank area (where the icons were) on the right. This work area does in fact consist of four connected blocks and you can scroll it so as to work on adjacent areas to the right or below.

But there is no indication of what constitutes a normal Spectrum screen and neither does creating a circle, box or a fill command extend outside the work area shown. This means that you can produce a SCREEN$ title screen but it is not that simple – you would have to mark out the boundaries by counting character cells and drawing your own construction lines.

There is a facility for reducing all four blocks and viewing them as one picture but as you can only work on the expanded "quarters", any but the lightest texturing will fill-in and spoil the effects you have been trying so hard to obtain.

Also there is nothing you can do with this combined picture except look at it with AMX art. Even then you do not get all of the picture – a pixel or two is lost around the edges although this is only of concern if you have drawn a neat box around the border.

The other main drawback to AMX Art is that pictures are only drawn in black and white. To get colours you must save the picture as a SCREEN$ file and load this into a different program called Colour Palette. As AMX Art does not have an option for displaying a background grid it is not very easy to be sure that the colours you can add will appear in the right places.

There is a long-winded way round this. You can define that the movement caused by the mouse occurs in eight pixels steps – the width of the Spectrum's colour cells – but this could prove a most tedious performance.

On the plus side there are 39 textures or patterns that may be used by fill or paint (but there is no indication of the current choice). There are four typefaces available for text and these may also be double height. Text may only be displayed in a horizontal format.

All in all, the graphics package with the AMX Mouse spoils a very good product. It seems amazing that if it was felt this was the right type of program to accompany the mouse, Advanced Memory Systems did not check its viability with a few genuine users first.

The Art Studio

Although this program is supplied as part of the Kempston Mouse package, it will also accept control from the keyboard, joysticks or the AMX Mouse. It is a very professional program and will certainly do almost all that a graphics design package would be expected to do.

It has all the major options supplied in AMX Art plus a host of others. These make the program not only very easy to use but also able to produce pictures that can be readily incorporated within the user's own programs.

A very versatile facility is the use of windows, not in the form of WIMPs but as a definable box drawn on the work area. The contents of this box may be operated on in a number of ways. What is within the box may be erased, rotated, rescaled, repeated elsewhere on the screen or merged with other areas already created.

The font editor enables different typefaces to be designed, stored on cassette and loaded into memory as required. The existing Spectrum font of 96 characters may be altered and as this option does not restrict the user to alphanumeric characters, a limited number of graphics designs may also be included. This allows a variation of the more commonly found User Defined Graphics (UDGs) capability.

The Art Studio supports a number of different printer interfaces. After defining your particular input mode, say, mouse or joystick, and saying whether you will be using microdrives and printer, a customised program may be saved to tape or microdrive. This is well worth doing as the customised program takes an appreciably shorter time to load than the initial program.

Further details of The Art Studio may be found in the section on graphics packages in this issue. The instructions are comprehensive and are clearly laid out and they also include details of the further facilities available with Extended Art Studio – a program only at present available for disc and microdrive users.

The accompanying table lists the facilities of both mouse graphics packages for comparison. Some options listed are not available from either package but do enable the serious user to consider which other graphics designers would complement these.

Mouse control

Even with the inherent drawbacks of AMX Art, the use of a mouse with either of the programs proves the validity of the system. Almost no contact is required with the keyboard and this allows a greater concentration on the creative aspects of graphic design.

Pull-down menus and the use of icons reduces the number of times you have to refer to the instructions and this itself increases the speed that a picture can be transferred to the screen from one in the mind.

AMX Control

AMX Art may not be all that one would hope for but the software for including mouse control within your own programs is quite a different kettle of fish. All the obvious requirements – and some others – seem to be catered for.

AMX Control adds some 27 additional commands to your Spectrum's Basic, including the expected windows, pull-down menus, icons, and mouse pointer positioning and recognition.

Two proportional typefaces are available as are commands that enable the Centronics printer output. There is also an ON ERROR GOTO command that will trap a number of possible errors but not those generated by the Spectrum's ROM such as "Nonsense in Basic".

Icons are an important part of the mouse philosophy and an icon designer program is also included. 96 icons may be stored – 64 are predefined but may be altered and there are 32 blank for you to design yourself. The icon designer follows normal UDG designers in having commands – mouse controlled of course – to rotate and create mirror images in both axes. Sets of icons may be stored to tape or microdrive for use in your own programs.

Which one?

This really is the $64,000 question! Both mice behave very well and do all that is claimed for them. Of the graphics packages The Art Studio is streets ahead of AMX Art, but Kempston still have to finalise their control package.

You may not wish to write your own software, in which case all we can say is: "Yer pays yer money and takes yer choice".

	AMX Art	Art Studio
Background grid		✓
Lines/points	✓	✓
Different line sizes		✓
Radii		✓
Circles	✓	✓
Arcs		✓*
Boxes	✓	✓
Triangles		✓
Fill	✓	✓
Scroll attribute screen		
Different brush sizes	✓	✓
Tints/shades (spray)	✓	✓
Textures	✓	✓
UDG editor		
UDGs from work area		
Multiple UDGs (banks)		
Font editor		✓
Text	✓	✓
Rotate (character)		✓
Eraser	✓	
Copy (to printer)	✓	✓
Save	✓	✓
Load	✓	✓
Merge		✓
Include in Basic programs		✓
Magnify	✓	✓
Coordinates (X,Y)		
Undo		✓
AMX Mouse compatible	✓	✓
Kempston Mouse compatible		✓
Joystick cursor operation		✓
Screen memory compression		✓*
Cut and paste		✓
Rescale		✓
Microdrive compatible	✓	✓

Note: ✓ *– *'Extended Art Studio' used with microdrive*

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 £ ________

I wish to pay by
☐ Access ☐ Visa No.________ Expiry date________

Signed________

Name________

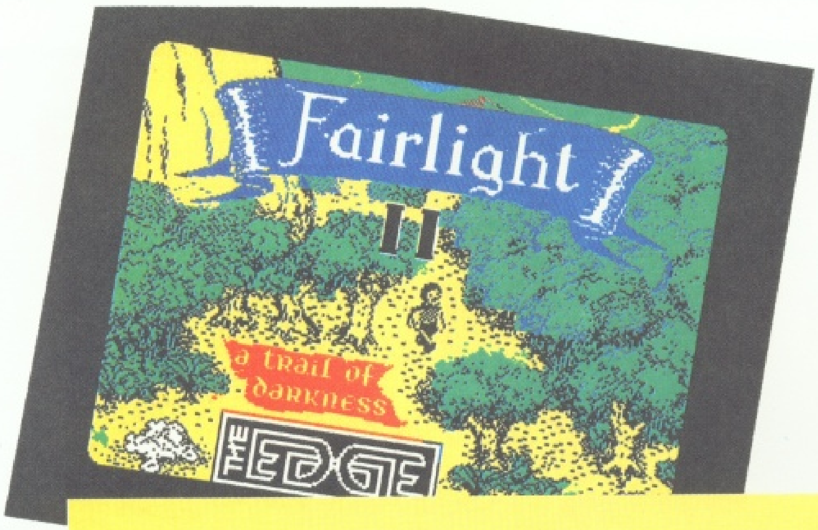
Address________

____

Spectrum 48k
£5.95

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

Q W E R T Y U I O P A S D F G H J K L Z X C V B N M



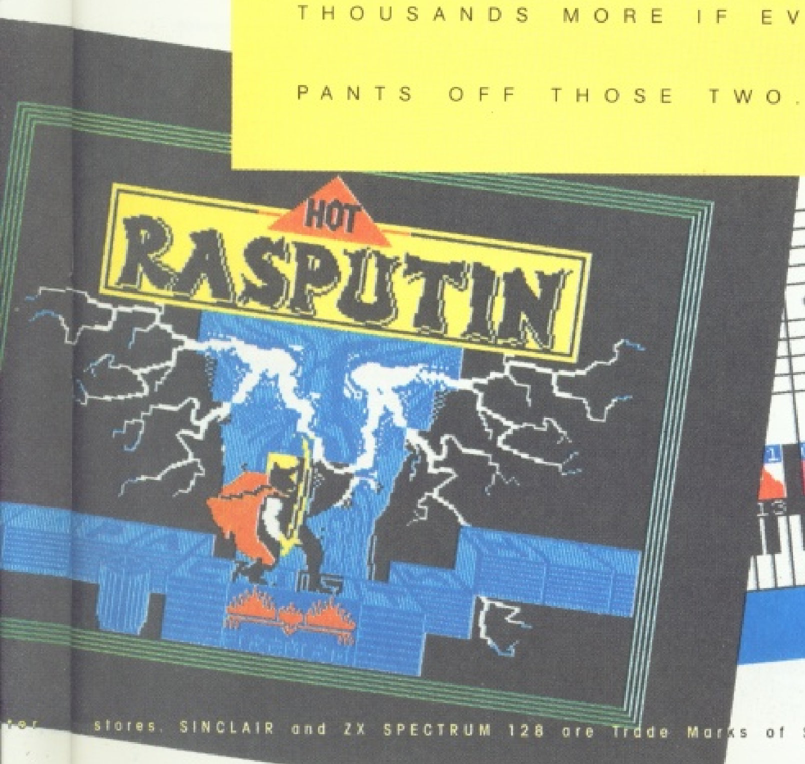
THE NEW ZX SPECTRUM 128 CAN HANDLE HARDER GAMES. WHETHER YOU CAN IS A DIFFERENT MATTER.

From Boots, Currys, Dixons, Laskys, John Lewis, Power City, W H Smith and other good comput

SLOT A CASSETTE IN AND GET READY FOR A REAL PASTING. THE NEW SPECTRUM'S 128K MEMORY WILL SHOW YOU JUST HOW TOUGH IT IS TO THRASH DALEY THOMPSON AT HIS OWN GAME. ROUGHLY THREE TIMES AS TOUGH.

OR IT'LL KEEP YOU PINNED TO THE EDGE OF YOUR SEAT PLAYING A GAME CALLED NEVER ENDING STORY. IT TAKES THREE TIMES AS LONG. AND BECAUSE WE WANT YOU TO SEE JUST HOW BRILL' ITS GRAPHICS ARE, AND HOW ITS NEW SOUND SYNTHESISER PLAYS MUSIC AND SPINE CHILLING SOUND EFFECTS. WE'RE GIVING BOTH TITLES AWAY FREE WHEN YOU BUY IT. BY THE WAY, THERE ARE THOUSANDS MORE IF EVER YOU DO BEAT THE PANTS OFF THOSE TWO.

Q W E R

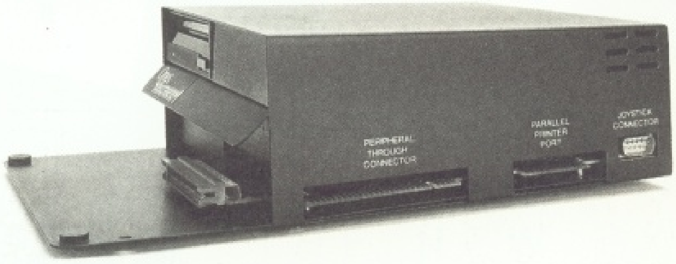


J K L Z X C V B N M

stores. SINCLAIR and ZX SPECTRUM 128 are Trade Marks of Sinclair Research Ltd.

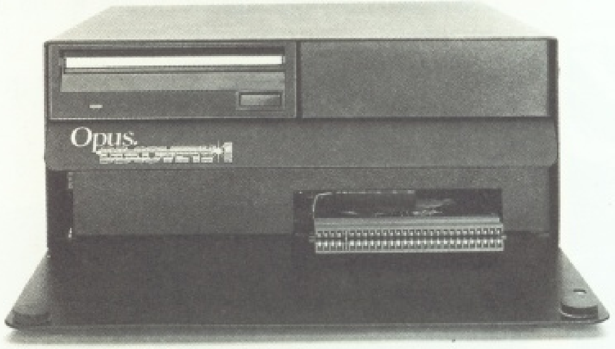
sinclair

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 stystem – 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.

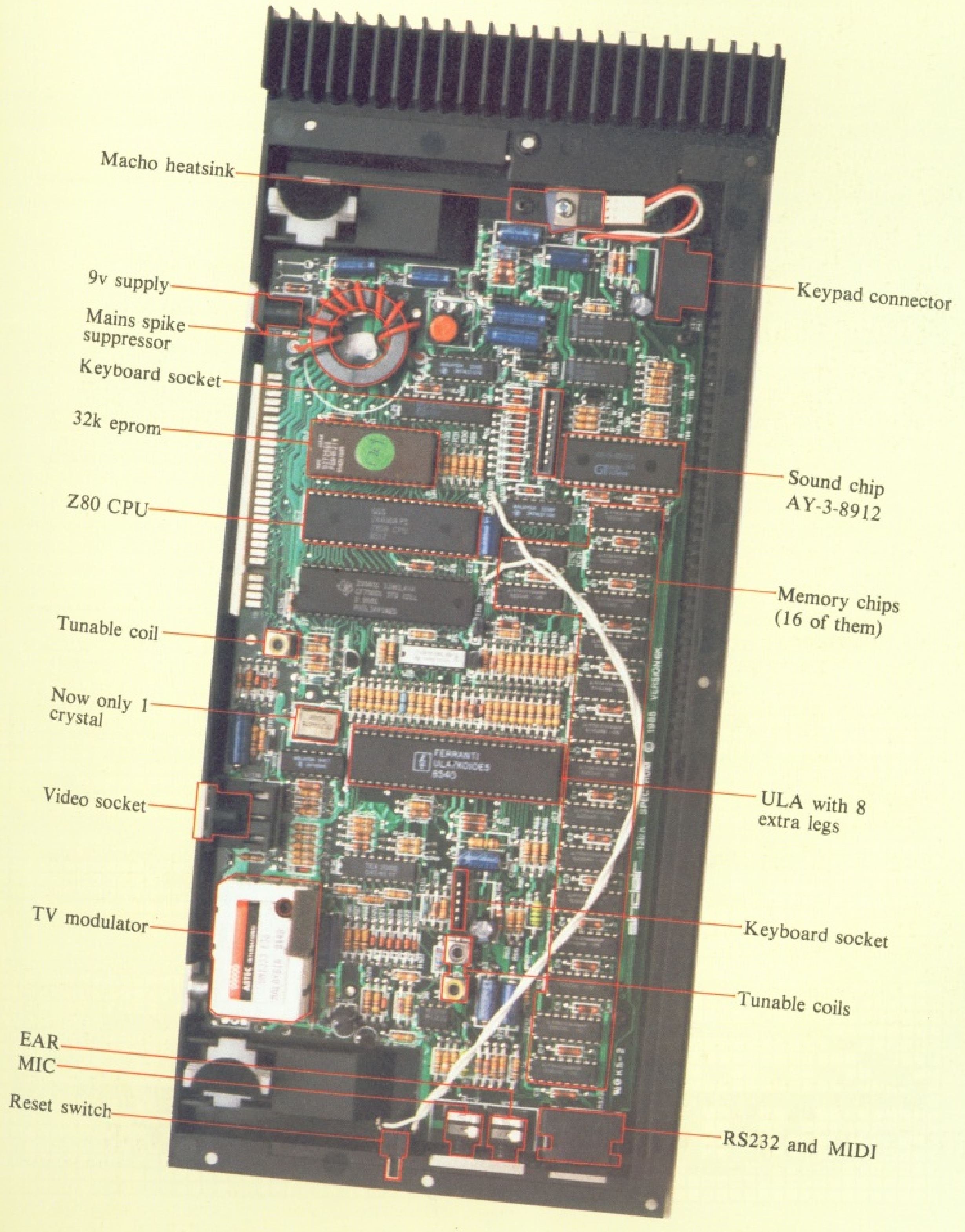
Opus Supplies Ltd,
55 Ormside Way,
Holmethorpe Industrial Estate,
Redhill, SURREY RH1 2LW.

Opus.

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

Inside the 128

As soon as we got our hands on to our Spectrum 128 we couldn't resist lifting the lid. We found quite a few interesting changes from the old Spectrum so we thought we'd pass the details on . . .



Animation aesthetics

Final part of the series that helps you get things moving

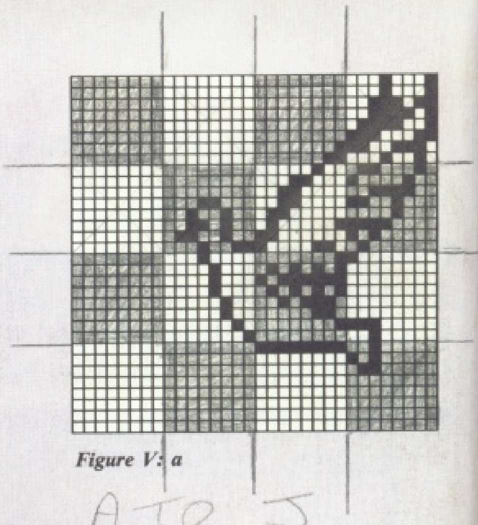


Figure V: a

A TO J

IT lies beyond the scope of this series to explore game design but in this article we shall look at the aesthetics of animation.

If you intend to put animation of any kind in your programs it is worth stressing again that it is very time consuming to do so properly. It certainly pays to get your animation sequences sorted out independently of the main program. So, on with a few examples.

Identified flying object

The basic ingredients of a great number of games are aliens – so I have chosen a flying saucer.

If you look at Figures I to IV you will see a four-frame sequence. I'm very fond of four frame sequences for this type of repetitive action.

Looking carefully at the bands at the bottom and the dots in the middle of the saucer you will see that the bands are moving right to left while the dots are moving left to right. The movement, in both cases, is one pixel at a time.

This imparts a more complicated illusion of movement to the saucer.

In flight entertainment

The second example in Figures V to VIII is a bird in flight. Again I've used four frames of animation but this time put them together in a six-frame sequence which shows the frames in the following order:

1. FRAME a	4. FRAME d
2. FRAME b	5. FRAME c
3. FRAME c	6. FRAME b

This is cheating. I should really use six frames of animation but I can get away with four because the bird is only small and the slight differences in wing position for sequence steps 5 and 6 will not show sufficiently at this resolution.

Note also the apparently large gap between frames b and c, visible by comparing the amount the wing tip has moved between successive frames.

This is a good example of the need to avoid smooth movement in certain circumstances. The wing is moving fastest at this time either upwards or downwards, and to mimic the speed more movement is allowed between the frames.

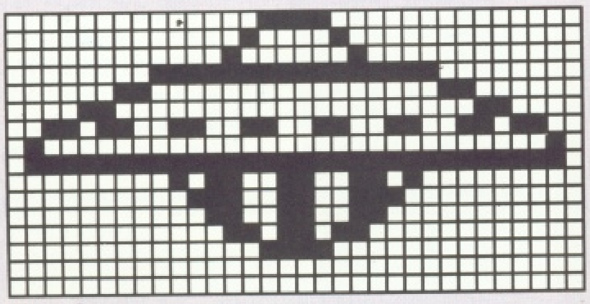


Figure I

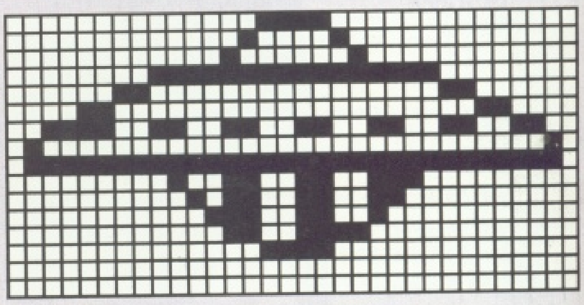


Figure II

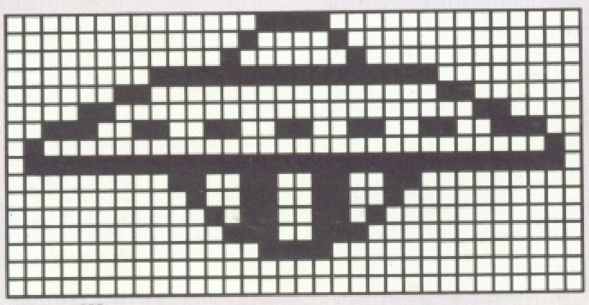


Figure III

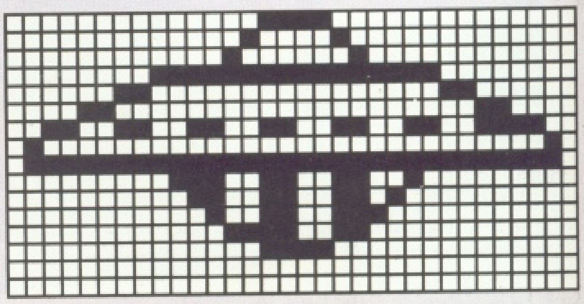


Figure IV

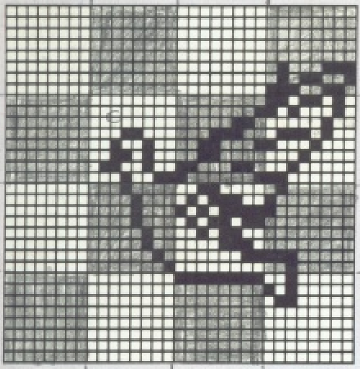


Figure VI: b

K TO Q

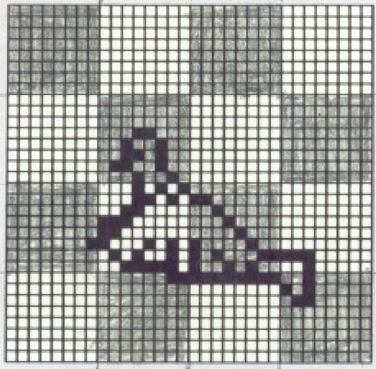


Figure VII: c

P TO J

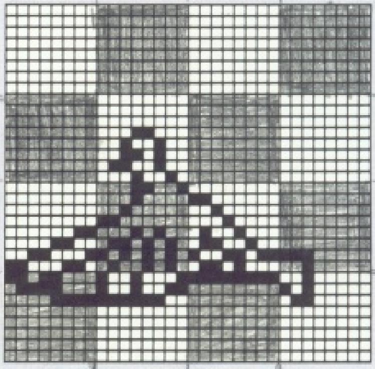


Figure VIII: d

U TO

Explosive qualities

Figure IX is yet another four-frame cycle, this time an explosion – actually it's five frames as the last frame has to be completely blank to erase the last bits of the blast.

Notice how the explosion occurs and that it is already beginning to dissipate by frame c. Also note that an attempt has been made to follow a few pixels through from frame a to frame d. It should be possible to find these by "back-tracking".

Such detail can make or break even simple animation. This is obviously not cyclic animation and should only be used once in a sequence.

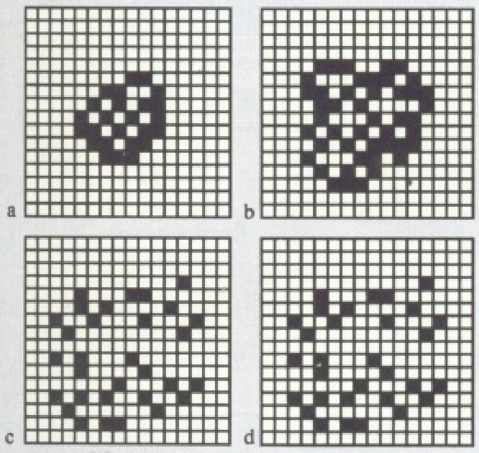


Figure IX

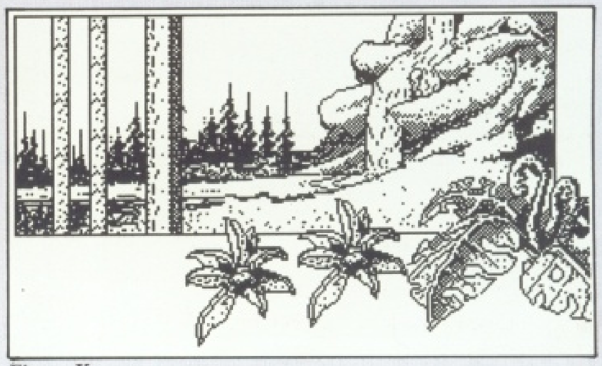


Figure X

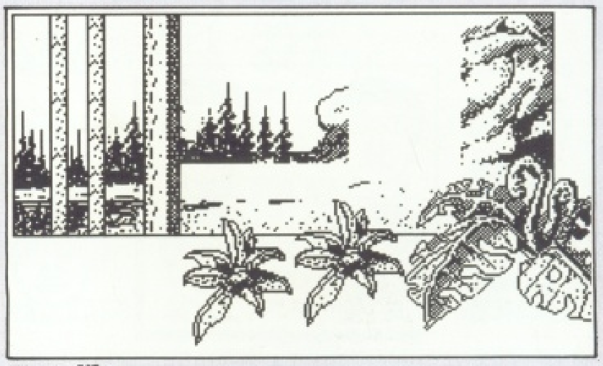


Figure XI

The waterfall

Look at the picture of a waterfall in Figure X. This is an example of an animated picture such as might be present in an illustrated adventure game or as an introductory/loading screen.

It is made up of two main elements:

1. *The main background minus the waterfall (Figure XI).*
2. *The animated section with four alternate frames (Figure XII).*

Note that only a very small area of the entire picture is animated. The frames must be shown in the correct order with a very short time interval between them.

If you look carefully at frame a and choose a point somewhere in the water you should then be able to follow it into frame b and so on, the movement between

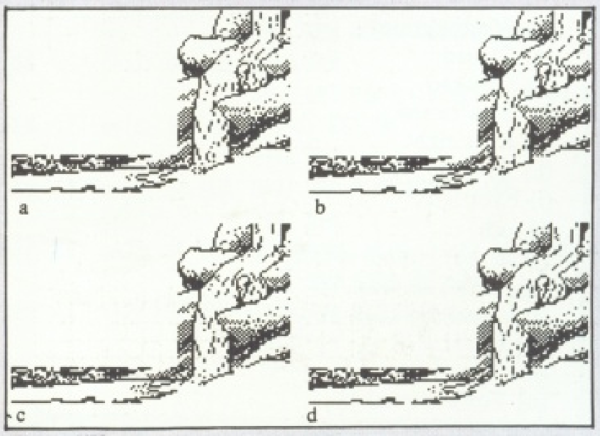


Figure XII

frames being roughly even.

You should also notice that after frame d it should again be possible to find the first point now much lower down back on frame a. This gives the illusion that the animation is more complex than it really is, and when the program is running it looks as though there are far more than four frames in sequence.

The technique can be used whenever a small repeated movement is needed – waterfalls, fountains, fire, smoke and so on.

When designing frames for your own sequences only work on very small areas of the animation at a time, and follow a small section of the action from one frame to the next.

Animated expressions

Another area little used as yet in commercial software is that of animated character faces. It has been tried, in particular, in The Great Space Race by Beyond and Minder from dk'tronics, but the results have scarcely scratched the surface of the potential.

We have included a facial generation kit so that you can experiment with this method.

If you look at Figure XIII you will see the pieces needed to create almost any expression, mood, or lip movement.

Some of the "sound" mouth expressions can also be used for mood creation.

Figure XIV is the normal expression magnified to give a clearer idea of the dimensions. If you wish to create the expressions shown remember that:

- The eye frame measures 4×2 characters
- The mouth frame is 4×3
- The hair frame is 6×4

Remember also when making up a SCREEN$ for the Framemaker program (see Page 84, *The Complete Spectrum* Part 3) that Framemaker reads from left to right and from top to bottom and that any frames need to be set in:

PAPER 5: INK 0:BRIGHT 0:FLASH 0

Frames must not touch, must not be on the bottom two lines of the screen and must not touch the border.

A few minor changes to the program are needed to enable it to print at more than one location. These changes are given in Program I.

The character codes in lines 3350 to 3570 do not correspond to the labelling of Figure XIII. Instead they refer to the order that the frames are to be read by the Framemaker program.

Please note that control over the sequence in which the frames are printed comes in the form of a character string – in this case b$. The program looks at each character in the string in turn and acts accordingly, and it is possible to develop very complicated sequences.

A frame will remain on the screen until cancelled by a new one of the same type (mouth/eye), and often a combination of eye and mouth expressions are needed to get the appropriate effect. Different effects will be

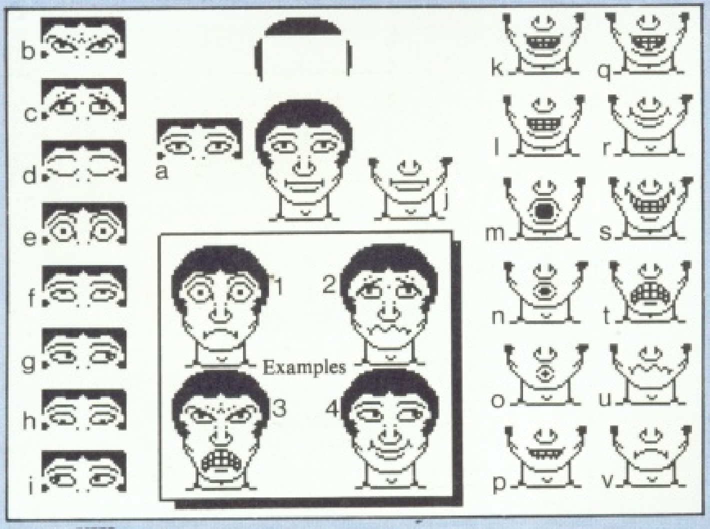


Figure XIII

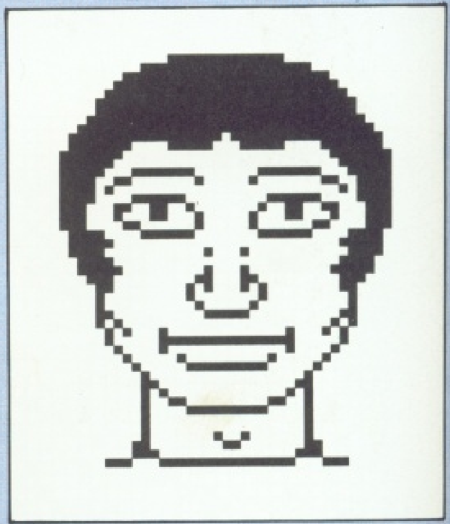


Figure XIV

Eyes
(a) Normal
(b) Angry . . . brows knitted
(c) Worried/tired
(d) Closed
(e) Shocked
(f) Eyes raised
(g) Eyes right
(h) Eyes down
(i) Eyes left
Mouth
(j) Mouth closed/normal
(k) Vowel sounds A/I
(l) Vowel sound E and C/D/G/K/N/R/S/T/H/Y/Z
(m) Vowel sound O
(n) Vowel sound U
(o) Sound W/Q
(p) Sound F
(q) Sound L
Other mouth expressions
(r) Smile
(s) Grin
(t) Teeth bared
(u) Worried
(v) Unhappy
Examples
1. Unhappily shocked – made up of eye frame (e) and mouth frame (v).
2. Worried – eye frame (c) and mouth frame (u).
3. Angry – eye frame (b) and mouth frame (t).
4. Happy/anticipation – eye frame g and mouth frame r.

generated by the path chosen from one frame to the next.

As an example, if the character starts off normal in all respects, the following frames – i, h, q, l, n, j, i, l – can be used to make the character appear to blink, become wide-eyed, gape open-mouthed, and finally blink in disbelief.

A similar result can be obtained with the sequence – l, j. The final expression is the same but the effect is nowhere near as amusing. Using graphics in this way is akin to acting, albeit in a Gerry Anderson puppet sort of way, and requires a lot of experimentation.

The entire facial feature codes take up only 23 character codes a to w (lower case). It is therefore very easy to define actions using other character codes and merely add them to the "IF . . THEN . ." block of statements.

To be honest this is not a very economical method of programming but it is easy to do and allows you to try out a large number of ideas much more quickly than by other methods.

Possible additions include using the numbers (0 to 9) to give pause facilities of varying length, defining a symbol, say "!", to indicate repeating the entire sequence, indicating a branch to another part of the program – for example, to print a word balloon or define a new sequence string.

And finally . . .

In conclusion we hope this introduction has inspired you to try graphics animation. Many libraries contain a section on animation, and books like The Art of Walt Disney, The Do It Yourself Film Animation Book or any book that includes freeze frame photography can be useful.

Ultimately it's up to you to do your research carefully and find out how to make an object move. And remember – a cine projector, video with single frame advance, or even acting in front of a mirror can all help you to analyse movement for use in graphics.

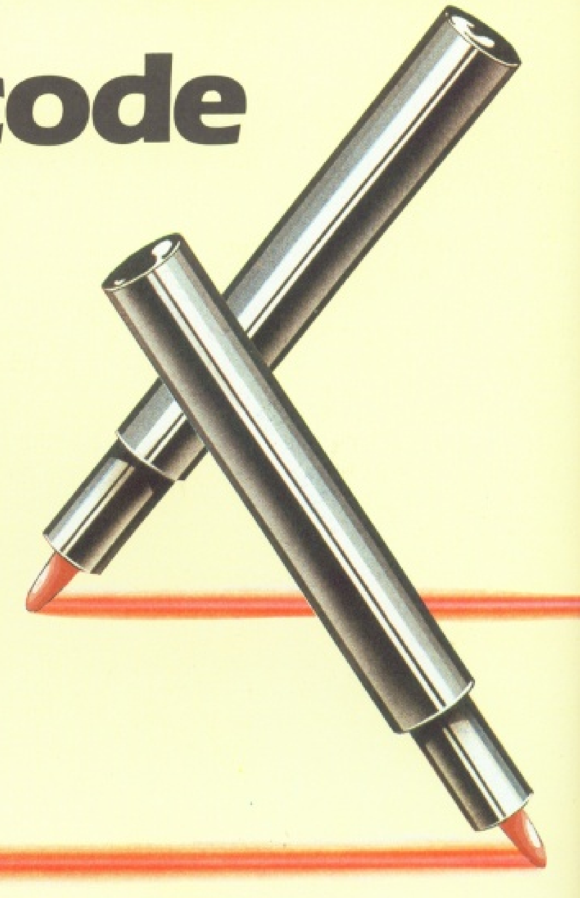
Post script

When attempting to include animation in your games, the following may be of some use:

- *A compiler to convert your Basic program into machine code. I use the Blast compiler from Oxford Computer Systems.*
- *A good aid to game production, possibly White Lightning or the new Laser Basic from Ocean, which looks very interesting.*
- *A good drawing program or utility. I personally use Grafpad from British Micro. Most drawing packages contain very similar facilities but the ones I have found most useful include screen enlargement to work on small sections at a time and edit individual pixels, the ability to take a small section of the screen and repeat it at other locations, and mirroring of a section of the screen in either the horizontal or vertical plane.*

```
3002 REM PATCH
3003 REM
3015 LET b$="acdefghijklmnopqrstuv":
REM SEQUENCE STRING
3020 LET seq=LEN b$
3025 REM
3026 REM
3027 LET v=6: LET h=10: REM SET POSIT
ION ON SCREEN
3028 LET hair=5900: LET eye=5930: LET
 mouth=5960: LET pk=5990: REM SET ADD
RESSES OF SUBROUTINES
3301 REM
3305 REM MAIN LOOP
3306 REM
3308 GO SUB 6000
3310 LET n=2: GO SUB hair: REM PRINT
HAIR
3315 FOR m=1 TO LEN b$
3320 LET t$=b$(m)
3330 LET l=4: REM LENGTH OF FRAME IN
CHARACTERS
3350 IF t$="a" THEN  LET n=1: GO SUB
eye
3370 IF t$="c" THEN  LET n=3: GO SUB
mouth
3380 IF t$="d" THEN  LET n=4: GO SUB
mouth
3390 IF t$="e" THEN  LET n=5: GO SUB
eye
3400 IF t$="f" THEN  LET n=6: GO SUB
mouth
3410 IF t$="g" THEN  LET n=7: GO SUB
mouth
3420 IF t$="h" THEN  LET n=8: GO SUB
eye
3430 IF t$="i" THEN  LET n=9: GO SUB
eye
3440 IF t$="j" THEN  LET n=10: GO SUB
 mouth
3450 IF t$="k" THEN  LET n=11: GO SUB
 mouth
3460 IF t$="l" THEN  LET n=12: GO SUB
 mouth
3470 IF t$="m" THEN  LET n=13: GO SUB
 eye
3480 IF t$="n" THEN  LET n=14: GO SUB
 eye
3490 IF t$="o" THEN  LET n=15: GO SUB
 mouth
3500 IF t$="p" THEN  LET n=16: GO SUB
 mouth
3510 IF t$="q" THEN  LET n=17: GO SUB
 eye
3520 IF t$="r" THEN  LET n=18: GO SUB
 mouth
3530 IF t$="s" THEN  LET n=19: GO SUB
 mouth
3540 IF t$="t" THEN  LET n=20: GO SUB
 mouth
3550 IF t$="u" THEN  LET n=21: GO SUB
 mouth
3560 IF t$="v" THEN  LET n=22: GO SUB
 eye
3570 IF t$="w" THEN  LET n=23: GO SUB
 eye
5800 GO SUB pk
5810 NEXT m
5820 GO TO 1
5830 REM
5831 REM END MAIN LOOP
5832 REM
5900 REM
5901 REM HAIR SUBROUTINE
5902 REM
5905 LET l=6: LET hor=h: LET test=3:
LET vert=v: GO SUB pk: RETURN
5930 REM
5931 REM EYE SUBROUTINE
5932 REM
5935 LET hor=h+1: LET test=2: LET ver
t=v+1: RETURN
5960 REM
5961 REM MOUTH SUBROUTINE
5962 REM
5965 LET hor=h+1: LET test=3: LET ver
t=v+3: RETURN
5990 REM
5991 REM POKE SUBROUTINE
5992 REM
5995 POKE 23606,x(n): POKE 23607,w(n)
: GO SUB 3200: RETURN
6010 FOR m=1 TO frames
6020 LET n=m
```

Machine code graphics



First in a series which clearly explains this complex area of assembler programming

BEFORE we look at any machine code graphics, we'd better see how the Spectrum organises its screen memory. We need to know which byte does what; that is, we need a screen memory map. The screen display is built up from the information contained within two distinct areas of random access memory. The two sections are responsible for different aspects of the picture you see on the screen.

The shapes of the characters displayed are created from the data stored in the first section of memory. This is called the display file. Data for the colour of each character displayed is stored in the second section of memory. This is the attribute file.

Figure I is a memory map showing the location of the display and attribute files. As you'll see, they are placed immediately following the Basic ROM.

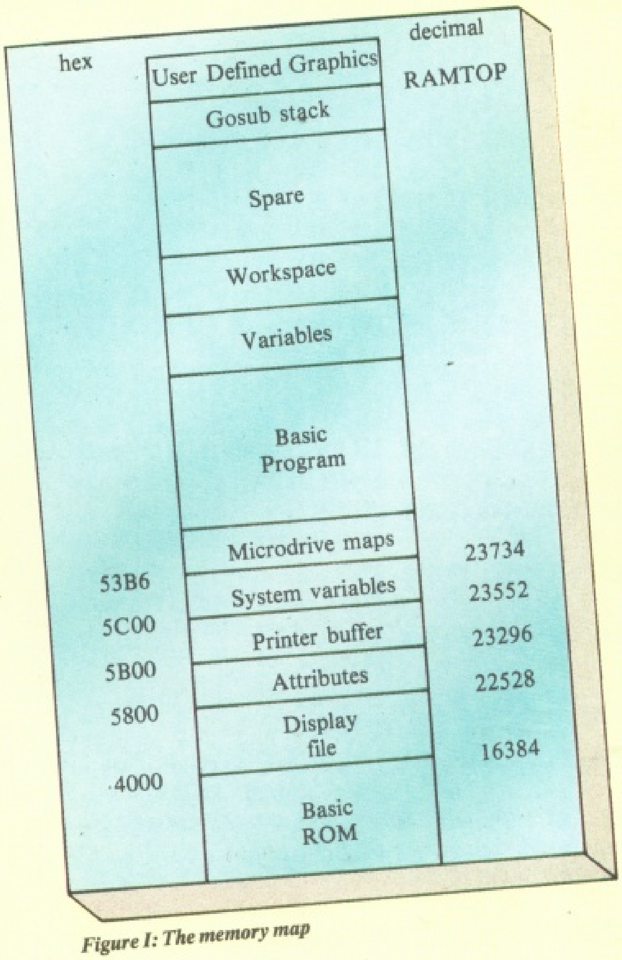


Figure I: The memory map

The screen RAM

The display file starts at &4000 (16384 decimal). It is &1800 bytes long, so if you're good at hexadecimal arithmetic you'll see that it ends at &57FF. As you'll see from Figure II this section of RAM is divided into three sections – top, middle and bottom. Program I demonstrates this quite clearly. As it runs through the display file poking consecutive bytes with 255 you'll see the screen's structure. The program waits for a keypress before poking each byte, so hold down a key to see how the screen is built up byte by byte.

Remember that a line on the screen, which is of

```
  10 REM PROGRAM I
  20 BORDER 5
  50 LET h$="0123456789ABCDEF"
 100 FOR a=16384 TO 22527
 110 POKE a,255
 130 PRINT AT 15,5;
 140 GO SUB 600
 170 IF INKEY$="" 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

course one character in depth, consists of eight rows of pixels. The top row of pixels of the first line of the screen is displayed first. This is followed by the top row of pixels of the second line, then the third and so on until the end of the seventh line, counting the first line as zero.

Though we're poking consecutive bytes, it should be obvious by now we're not necessarily filling in consecutive pixels. Only when the top row of pixels of each of the first eight lines has been poked does the program go on to fill the second row of pixels on each line. Then the third, the fourth and so on.

When the first eight lines have been painted a solid black – the attribute file hasn't been touched so there's no colour – the next block of eight lines is filled in the same manner. After the middle block comes the bottom six lines plus the two lines reserved for input.

When directly accessing the screen RAM in this way there are no restrictions, unlike Basic, which reserves the last two lines for INPUT. The bottom two lines are no different to the others, so we can poke them with impunity.

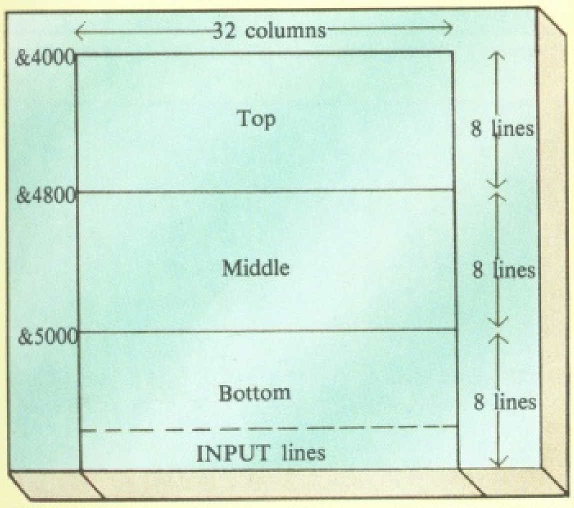


Figure II: Screen memory

As you can see, the bytes allocated to the screen correspond to what appears on that screen in a very complicated way. To recap, we deal with the screen in three separate chunks of eight lines. Each of the eight lines is built up from consecutive bytes so that all the top row pixels of these lines are defined, then all the pixels of the second row and so on. Once all the rows of all eight lines of a chunk are defined, we move on to the next chunk.

Clearing the screen

Poking each byte of the display file with 255 fills the screen, as we've seen. To clear it the display file is filled with zeros. Program II is an assembly listing of a short screen clear routine. Changing the value of *byte* allows you to fill the screen with patterns.

```
let byte=0

LD HL,&4000
LD DE,&4001
LD BC,&17FF
LD (HL),byte
LDIR
RET
```

Program II

How the display file works

Each of the three sections of the screen is &800 bytes long. The top third of the screen runs from &4000 to &47FF, making up the first eight print lines, that is, our lines, on which the characters from a PRINT statement would appear. Each line is 32 (&20) columns wide.

The second section, the middle eight lines, runs from &4800 to &4FFF. The last section runs from &5000 to &57FF. This corresponds to the bottom six lines plus the two lines reserved for input. Figure II summarises this.

Each horizontal row of pixels is &20 bytes long. There are also &20 characters per line so this means that each character must be one byte wide. As a character is eight pixels wide each byte is responsible for eight horizontal pixels.

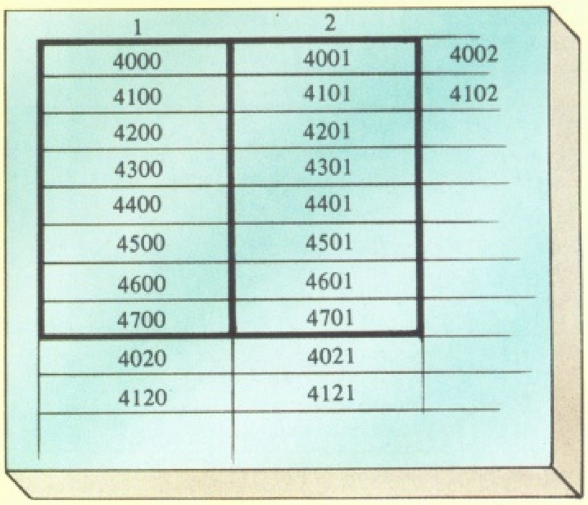


Figure III: The top left corner of the screen showing the first two character cells

This is quite convenient, as there are eight bits per byte. A single pixel on the screen corresponds to a single bit in a byte in the display file.

If a bit is set the pixel is visible – assuming of course that the pen colour is different to the paper. If a bit is reset the pixel is no longer displayed.

Since there are exactly &20 bytes corresponding to a row of pixels there are &20 bytes between the top of one character and the top of another on the line directly below it. So to move from one character cell to the one directly below &20 is added to the address. *(This isn't quite true at the borders between sections, but we'll deal with this later.)*

A single character appearing on the screen is made up of eight rows of pixels, each row being one byte wide. So we need eight bytes to define a standard screen character. We call the memory allocated to a single character a character cell.

Oddly enough, the addresses of the eight bytes of RAM making up a character cell are separated from each other by &100. You see, before we start on the second row of pixels we have to have filled in all the top rows of the section's eight lines, so we've used up 8*&20=&100 bytes before we get to the first pixel on the second row! Figure III shows the first two character cells at the top left corner of the screen.

As we've said, the bit pattern of these eight bytes

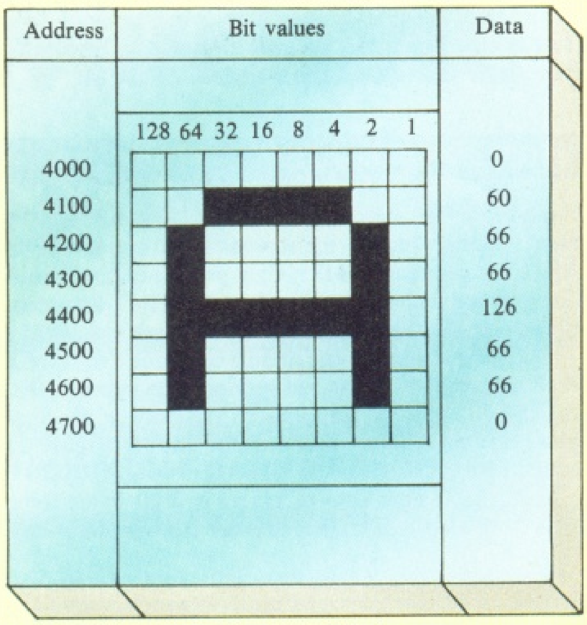


Figure IV: Data required to print a letter A in the top left corner of the screen

```
              LD HL,&4000
              LD DE,8*65+&3C00
              LD B,8
      .loop6  LD A,(DE)
              LD (HL),A
              INC DE
              INC H
              DJNZ loop6
              RET
```

Program III: Simple routine to print A

make up the characters you see on the screen. Figure IV shows a letter A printed in the top left corner of the screen.

The data for all the Ascii characters is stored in the ROM starting with Space at &3D00. The address of the data for a character is &3D00+8* (Ascii code–32). This is equivalent to &3C00+8* (Ascii code). Program III is a simple routine to print a letter A at the top left corner of the screen.

Machine code printing

The technique used in Program III to print A on the screen is fine for standard sized characters that behave themselves by staying on the print lines. However games are much better if the characters are fairly large and move smoothly a pixel or two at a time. For this a much more sophisticated routine is required.

Program IV is a print routine that is capable of printing any size character at any screen address. It can cope with any width and any height and even if the character is split over the boundary between two sections of the screen.

Each item of data is collected and XORed with the data in the display file. This prevents the background from being erased if the character is close to another. Also the character being printed can pass over others without wiping them out when it moves off.

The routine prints each row of the character, incrementing the high byte of the address in H each time. When the bottom of the character cell is reached some fiddling is necessary to move on to the top of the

```
              let address=&4000
              let data=8*65+&3C00
              let rows=8
              let width=1

              LD HL,address
              LD DE,data
              LD B,rows
      .loop4  PUSH BC
              LD C,width
              PUSH HL
      .loop5  LD A,(DE)
              XOR (HL)
              LD (HL),A
              INC HL
              INC DE
              DEC C
              JR NZ,loop5
              POP HL
```

Program IV: Printing a character of any size at any screen address

```
             let attr_p=&5C8D

              DI
              LD HL,&4020     ;scroll up
              LD DE,&4000
              LD BC,&1AE0
              LDIR
              LD HL,&47E0
              LD DE,&40E0
              LD C,&20
              LD A,16
      .loop   CALL moveu
              DEC A
              JP NZ,loop
              LD H,D          ;blank line
              LD E,&E1
              DEC BC
              LD A,8
      .loop1  LD (HL),0
              CALL moveu
```

Program V: Scrolling the screen up

cell below. The bottom is found by checking the three least significant bits of the high byte of the address. If this is equal to seven then it's at the bottom.

The data for the character can be anywhere in the memory, though it should be in the same form as the letter A in Figure IV. See if you can work out what exactly is going on. At the moment things are rather static. We'll see how to get things moving over the next few months.

Scrolling the screen

To confirm our mastery of the screen layout, and to provide you with some useful techniques, take a look at Programs V and VI. These are two routines to scroll the screen up or down. The Z80's block move instructions, LDIR and LDDR help here, though the way the screen is split into three makes life difficult.

To scroll the screen up Program V first moves the

```
          LD A,H
          AND 7
          CP 7
          JR Z,bottom
          INC H
          JP here
.bottom   LD A,H
          RRCA
          RRCA
          RRCA
          LD H,A
          LD BC,&20
          ADD HL,BC
          SLA H
          SLA H
          SLA H
.here     POP BC
          DJNZ loop4
          RET
```

```
          DEC A
          JP NZ,loop1
          LD H,&5A       ;cls to paper
          LD D,H
          LD A,(attr_p) ;attribute
          LD (HL),A
          LDIR
          EI
          RET

.moveu    PUSH BC
          PUSH DE
          PUSH HL
          LDIR
          POP HL
          POP DE
          POP BC
          INC H
          INC D
          RET
```

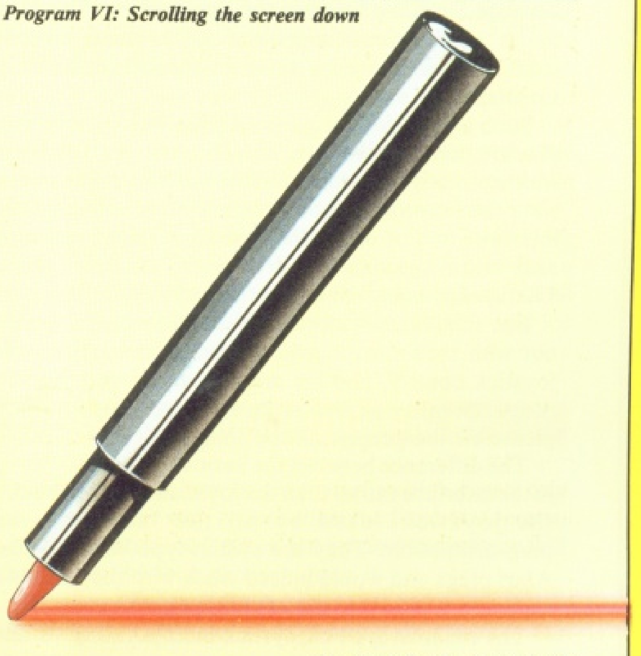
whole display and attribute files up &20 bytes using LDIR. Unfortunately this means that the top line of the top section becomes the bottom line of the top section. The top line of the middle section becomes the bottom line of the middle section and the top line of the bottom section becomes the bottom line of the bottom section.

This means that after the screen has been scrolled the bottom line of the middle section has to be moved to the bottom line of the top section and bottom line of the bottom section has to be moved to the bottom line of the middle section. The bottom line of the bottom section is then cleared. If you're confused, don't blame me, I didn't design the screen layout . . .

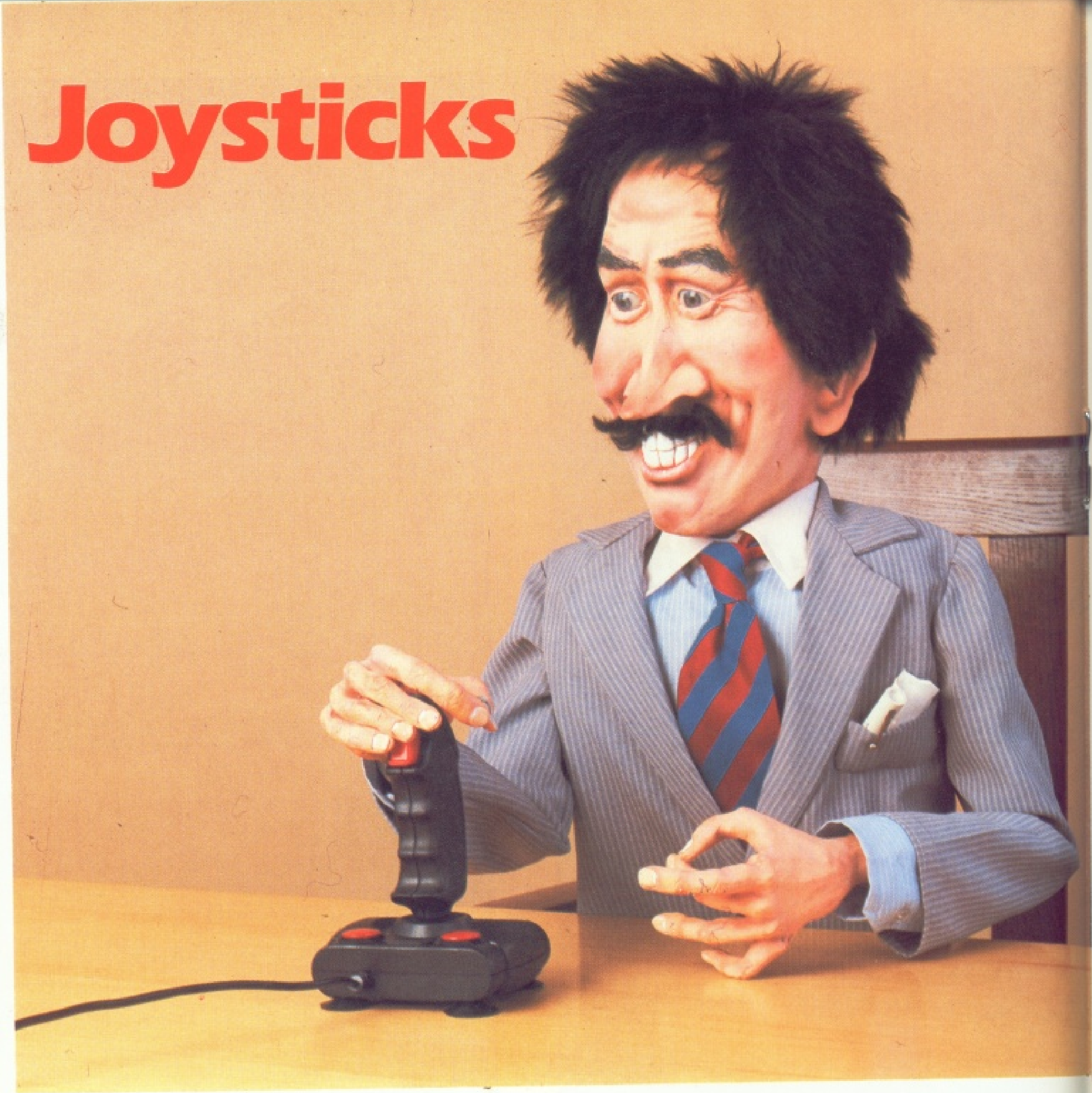
Finally, the bottom line of the attribute file is cleared to the current paper colour, and there we have it – the whole screen moved up! Moving it down is similar, but you have to take care with the order you do things in otherwise you'll copy down something you've already copied. I'll leave you to sort out exactly how Program VI manages it.

```
          let attr_p=&5C8D


          DI
          LD HL,&5ADF    ;scroll down
          LD DE,&5AFF
          LD BC,&1AE0
          LDDR
          LD HL,&501F
          LD DE,&571F
          LD C,&20
          LD A,16
.loop2    CALL moved
          DEC A
          JP NZ,loop2
          LD H,D         ;blank line
          LD E,&1E
          DEC BC
          LD A,8
.loop3    LD (HL),0
          CALL moved
          DEC A
          JP NZ,loop3
          LD H,&58       ;cls to paper
          LD D,H
          LD A,(attr_p) ;attribute
          LD (HL),A
          LDDR
          EI
          RET


.moved    PUSH BC
          PUSH DE
          PUSH HL
          LDDR
          POP HL
          POP DE
          POP BC
          DEC H
          DEC D
          RET
```

Program VI: Scrolling the screen down



Joysticks



IT is a strange anomaly that the Spectrum, which is one of the most popular game-playing machines, has no built-in joystick port. There wasn't even an official Sinclair add-on joystick interface for a year after the Spectrum first became available. This may have been because Sir Clive believed the Spectrum was destined to be used in education and as a small business machine, and its use as a games console took him by surprise.

Such a major oversight does not go long unrepaired, though, when there are a million or so customers waiting to buy a joystick. Kempston Micro Electronics were the first to supply a joystick fitted with a suitable interface to slot on to the Spectrum edge connector, and most games programmers swiftly added "Kempston compatible" to their cassette inlays.

But this did not stop other manufacturers coming out with their own ingenious and varied schemes for joystick control, and we are now faced with joystick menus on many games as the software writers strive to please all the people, all the time.

The difference between the various systems is mostly in the interface rather than the joystick itself. Almost all the joysticks used by all the interfaces are plug-compatible. They are switch-type with standard Atari plugs and would indeed work with the Atari – as well as the Amstrad and lots of other computers.

The rheostat type of joystick incorporating variable resistors – as employed by Acorn computers – is not used with any popular Spectrum interface or software. This compatibility means that you can choose your joystick and interface separately, or you can use one joystick with several different interfaces.

There are really only four basic types of joystick interface, although there are many more brand names.

The interfaces

The Kempston and compatible types allow the joystick to be tested with a single instruction reading port 31, the position of the joystick being determined from the number returned. Programmers find this fast, compact and convenient, especially in machine code.

Almost every Spectrum arcade game written since 1982 has a Kempston option, but of the few that don't, some will crash the computer if the Kempston interface is connected. This is due to the fact that they use interrupt mode 2 in a way that clashes with the Kempston interface, according to Kempston, or because the Kempston's port address decoding is not rigorous enough, according to the programmers.

It is, however, possible to use interrupt mode 2 in a way that does not require rigour on behalf of the interface circuitry. Most programmers are willing to accommodate Kempston for the sake of the large

Hat trickery

Besides the combinations of different joystick interfaces in one box, there are interfaces with an amazing range of extra features built-in, even one – the Opus Discovery – with its own disc drive!

Nidd Valley Micro Products have a range of interfaces including programmable or Kempston types with slow motion control built-in. This feature, which is also available as a separate device, allows you to adjust the speed of the game so that you can play it in slow motion.

Datel Electronics offer interfaces switchable between Kempston, cursor, or Interface 2 controls with an option to send the sound through your TV speaker.

The Multiface One from Romantic Robot is basically a piece of hardware for making back-up copies of tapes, but it includes a joystick interface and composite video output.

The Fox Electronics programmable interface has built-in memory with battery back-up, so that it retains its programming even when the computer is turned off. There is room for 16 different sets of keys to be programmed in for different games. The battery recharges automatically when the joystick is in use.

number of Kempston joystick users, even if it does use up a quarter kilobyte of RAM.

Cursor-type joysticks are designed to imitate the pressing of the keyboard cursor keys, usually with the fire button imitating the zero key. This has the advantage that the joystick will work with some early software that has no joystick option but uses the cursor keys for movement.

One of the cheapest of these cursor joysticks was not electronic but wholly mechanical, clamping on to the keyboard and actually pressing the keys as you moved the stick! Luckily most of these were broken by journalists during "reviews", and very few got into the hands of the public. Most software nowadays has a cursor key option just to accommodate cursor joysticks, as the keys themselves are not very ergonomically placed for fast action games.

When Sinclair's own joysticks arrived they were predictably incompatible with any other system. They imitated keypresses on the top row of the keyboard, but not quite the same keys as the cursor keys. For those of you with the Spectrum Plus, the cursor keys were originally keys 5, 6, 7 and 8 used in conjunction with Caps Shift, which is why some pre-Plus Spectrum software does not work as it should with the cursors. It is really testing for the numbers without the shift, but the Plus cursor keys have the shift built in. The Sinclair Interface 2 did allow two joysticks to be used at once, however, and it had a slot for ROM-based software, very little of which ever materialised. There are very few games that provide for the use of two joysticks, either.

Oddly enough, you still can't buy joysticks from Sinclair. Interface 2 comes without the joysticks themselves, and you have to get them separately from another manufacturer.

Finally, in an attempt to cope with all the different control schemes in use with all available software, the programmable joystick – or rather about a dozen different ones – came into being. These allow you to program the interface so that the joystick will imitate whatever keys the game uses for control. Sometimes this programming is done with jumper leads, but most of the programmable interfaces available now have built-in memory which is programmed by manipulating the controls.

Combinations of the above include interfaces that are switchable between Kempston, cursor, and Interface 2 types, or which have two joystick sockets each providing different interface types. Programmables are also sometimes provided with a Kempston option.

The joysticks

Although any Atari-type joystick will plug into any Spectrum joystick interface, there are still plenty of differences between them on which to make a choice. Some have multiple-fire buttons, or automatic repeat fire. Some have suction cups to stick them firmly to a table, while others have a base shaped to fit the hand.

The feel of a joystick is very important to an avid game player and it is a good idea to try out a few before you make a choice. One that seems admirably robust to one set of hands may well feel stiff and awkward to another.

It is not at all uncommon for a joystick to be torn to pieces in the heat of an exciting game, so you want one that is sturdy, but not so stiff that it makes your wrist ache after a session. Many have large handles meant to be grasped in your fist, but there are delicate fingertip types too.

The more popular models are often fitted with "leaf" switches, which can give an imprecise feel. It is worthwhile looking for ones that use microswitches. These are the type of switch used in the coin-in-the-slot arcade machines and they not only give more delicacy of control but are an indication of a higher engineering quality.

Joysticks with special features like rapid or auto firing may not work with every interface. These rely on a bit of circuitry to pulse the fire signal, and may require a power feed from the interface.

Similarly, the provision of independent fire buttons may not be of use with every interface, or indeed with every game. Independent fire controls allow you to use one button for, say, firing the lasers and another for the rockets, but obviously this only works if the game has both lasers and rockets. You need an interface – necessarily a programmable type – that allows for more than one fire input. Lots of joysticks give you a choice of fire buttons which are not independent, but all linked together. This is a different thing, and is only for convenience of handling.

Programming the interface

A programmable interface will give joystick access to a few games that won't work with the fixed Kempston, cursor, or Interface 2 types, but remember that you will have to spend time programming it.

Programming usually requires you to hold the stick in the desired position while pressing the relevant key on the keyboard and a button on the interface. Some of the cheaper programmables even require a tape to be loaded to do the programming before you can load the game.

If you get the programming wrong – by forgetting which keys you need for the game you are about to load, for instance – then it's back to square one. It is worthwhile making sure the interface you propose to buy will allow you to reprogram after the game is loaded.

Some programmables disable the keyboard when the joystick is enabled, so unless all the controls can be programmed on to the joystick you will have to keep flicking a switch back and forth to use the keyboard for the extra functions.

This can be very awkward when you need that smart bomb in a hurry. It is also annoying between games when you have to deal with menu selections.

A variation on the programmable interface is the type in which the controls are assigned to keys by shifting electrical connections. The Comcon interface from Frel provides a pegboard representation of the Spectrum keyboard into which you insert plugs to select the controls for four directions and two independent fire buttons. This has the advantage of simplicity, and you can change the control assignments during a game if you wish.

Lots of games have programmable controls themselves, which allow you to assign the various functions to any key on the keyboard. This feature can be used to program the game to match cursor or Interface 2 joysticks, but not the Kempston ones. When both the game and the joysticks are programmable, then you just have to program them both to the same set of keys.

Strange devices

There are a few weird and wonderful objects that serve as joysticks, but resemble them only to a certain extent.

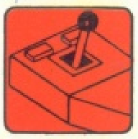
The Cheetah RAT uses an infra red link, like the remote control on a modern TV set, so there is no cord connecting the interface and the bit you hold in your hand.

Unfortunately, the actual control is not a joystick but a membrane keypad. The infra red link works perfectly, but the membrane control is horrible, so this must be regarded as a missed opportunity. The infra red link is an integral part of both interface and keypad, so there is no way to connect a proper joystick to the system. The RAT is Kempston compatible.

The Lazor from J. English Electronics is a touch control keypad that acts as a joystick but is activated by contact of your thumb or finger on five metal bosses.

This is a true touch control with no pressure required, and it should be very quick in action if you can get used to it. An obvious advantage is the lack of moving parts, which ought to make it hard to break. Buy your interface separately.

The Stick from Lightwave Leisure looks like it's already broken. It has no base at all, just a handle with a cord hanging from it. The special feature that makes this possible is the



The final analysis

Kempston or Kempston-compatible interfaces work with almost every game that has a joystick option, and you can buy both interface and joystick together at a price of about £15, if you shop around. Who would pay up to four times as much for a few extra features? Arcade game fanatics, that's who. They will not only pay out £30 or so for the most advanced, do-it-all programmable interface, but will probably end up with a collection of different joysticks with different handling characteristics as well.

Even if you have money to burn, in terms of sheer enjoyment you will probably be better off spending it on software than on sophisticated joysticks.

Suppliers

Kempston Micro Electronics
Singer Way
Kempston
Bedford
MK42 7AW
Tel: 0234 856633

Romantic Robot
77 Dyne Road
London
NW6 7DR
Tel: 0892 834783

Frel Ltd.
1 Hockeys Mill
Temeside
Ludlow
Shropshire
SY8 1PD
Tel: 0584 4894

Nidd Valley Micro Products
Freepost
Knaresborough
North Yorks
HG5 8YZ
Tel: 0423 864488

Datel Electronics
Unit 8
Fenton Industrial Estate
Dewsbury Road
Fenton
Stoke-on-Trent
Tel: 0782 273815

Lightwave Leisure
P.O. Box 33
Wallasey
Merseyside
L44 1EW
Tel: 051-639 5050

Cheetah Marketing
1 Willowbrook Science Park
Crickhowell Road
St. Mellons
Cardiff
Tel: 0222 777337

J. English Electronics
31 Lincoln Road
Southport
Merseyside
PR8 4PR

Fox Electronics
Fox House
35 Martham Road
Hemsby
Gt. Yarmouth
Norfolk
NR29 4NQ
Tel: 0493 742320

gravity-operated mercury tilt switch.

The idea is that you tilt The Stick as if it were a normal joystick and you get the same action on screen from the same sort of wrist movement, but with no base to hang on to or stick to a table with suction cups. Another one that takes some getting used to, especially as shaking it can have pretty wild effects, but it certainly reduces wrist fatigue. Plugs into any interface.

Etch a Sketch

ETCH a Sketch demonstrates how to get full eight-way control from the three most popular types of joystick interface – the Kempston, the Sinclair Interface 2, and the cursor key type.

Eight-way control means that you can have diagonal movement. Many chart topping games use only four-way control as this is certainly easier to arrange, especially in Basic. That is because the INKEY$ function, often used to read the cursor keys or Interface 2, will only work with one keypress at a time, and two keys are necessary for diagonal movement.

Using the IN function to read the appropriate ports is more flexible. However more work is required to test the individual bits of the byte read from the port. In Basic this is an awkward process because binary bit handling instructions do not exist, so we have to fake it with a lot of decimal arithmetic instead. This is one thing that would actually be easier to program in machine code!

Once you have selected your joystick type from the menu, you can use it to draw on the screen. There is a single flashing pixel to mark your position. To draw, hold the fire button down as you move. Use the stick without the fire button to move to a new position without drawing. The cursor pixel can also be used to erase by moving it over unwanted lines without pressing fire.

If you have no joystick you can still use the program. Just select the cursor-type stick and use the cursor keys instead, with the zero key as a fire button. If you have a programmable stick, you can program it to use either the cursors or the Interface 2 keys.

```
  10 CLS : PRINT "Select Joystick Typ
e"
  20 PRINT '''"Press K for Kempston t
ype"
  30 PRINT '"Press S for Sinclair Int
. II"
  40 PRINT '"Press C for Cursor Key t
ype"
  50 IF INKEY$<>"" THEN  GO TO 50
  60 LET i$=INKEY$: IF i$="" THEN  GO
 TO 60
  70 IF i$="k" OR i$="K" THEN  LET st
ick=1000: GO TO 150
  80 IF i$="s" OR i$="S" THEN  LET st
ick=2000: GO TO 150
  90 IF i$="c" OR i$="C" THEN  LET st
ick=3000: GO TO 150
 100 GO TO 50
 150 LET x=127: LET y=87: CLS
 160 PRINT #0;"Use stick to move - Fi
re to Draw"
 200 PLOT x,y: LET h=0: LET v=0: GO S
UB stick
 210 IF fire=1 THEN  PLOT  INVERSE 1;
x,y
 220 LET x=x+h: LET x=x+(x<0)-(x>255)
 230 LET y=y+v: LET y=y+(y<0)-(y>175)
 250 GO TO 200
1000 REM Kempston
1010 LET i=IN 31
1020 GO SUB 5000
1030 LET h=bit0-bit1: LET v=bit3-bit2
 : LET fire=1-bit4
1040 RETURN
2000 REM Sinclair Interface II
2010 LET i=IN 61438
2020 GO SUB 5000
2030 LET v=bit2-bit1: LET h=bit4-bit3
 : LET fire=bit0
2100 RETURN
3000 REM Cursor Key
3010 LET i=IN 61438
3020 GO SUB 5000
3030 LET h=bit2: LET v=bit4-bit3: LET
 fire=bit0
3040 LET i=IN 63486
3050 GO SUB 5000
3060 LET h=bit4-h
3100 RETURN
5000 REM test bits
5010 LET bit0=0: LET bit1=0: LET bit2
=0: LET bit3=0: LET bit4=0
5020 IF i>127 THEN  LET i=i-128
5030 IF i>63 THEN  LET i=i-64
5040 IF i>31 THEN  LET i=i-32
5050 IF i>15 THEN  LET i=i-16: LET bi
t4=1
5060 IF i>7 THEN  LET i=i-8: LET bit3
=1
5070 IF i>3 THEN  LET i=i-4: LET bit2
=1
5080 IF i>1 THEN  LET i=i-2: LET bit1
=1
5090 LET bit0=i
5100 RETURN
9999 SAVE "etch": GO TO 9999
```

An example of what can be achieved with Etch a Sketch

Etch a Sketch program

Pictures for all

We look at five graphics packages that can help you produce some amazing designs

NEARLY all the programs produced for the Spectrum nowadays have quite amazing title pages and all but a very few have some form of graphics within the program. So how do you go about producing these sort of effects for yourself?

The Spectrum in its basic form has a number of excellent graphics commands and is a darn sight easier to program than most of the other home computers on the market. But even with these advantages it is not easy to create the sort of pictures we are talking about.

There are principally two ways of building up a picture on the screen. One is to use a series of specially designed characters that may be used just as one would type in text, but in this case build up into a predetermined picture.

The other way is to create your design on the screen by switching on individual pixels, so producing a freeform drawing that can then be coloured in to give the final masterpiece.

The first method is normally referred to as low resolution graphics and the second as high resolution graphics. This is a bit of a misnomer, for although you only have 32 columns across the Spectrum's display and can therefore only have 32 separate character blocks, each of these blocks can individually incorporate hi-res detail.

A combination of the two methods will often prove to be the answer with the use of user-defined graphics (UDG) characters for any repetitive designs within the picture.

Although the design and use of UDGs is not complicated it can be laborious, and the Spectrum as it stands only supports 22 such characters. Using PLOT, DRAW, MOVE and CIRCLE can be a minefield for the programmer. There has to be an easier way.

The answer is to use one or more of the graphics packages available. Over the years there have been quite a number of these. Some have stood the test of time, some have been improved and recently we have seen the introduction of a couple of interesting icon driven graphics design programs where all functions are operated by a joystick or a mouse.

Here we're concerned with the joystick packages – the mouse packages will be dealt with in a separate article.

Not only do these program utilise all the inherent graphics commands of the Spectrum, they also offer many other easy to use facilities. They do not require you to have any programming knowledge and will enable you to create and save a picture of your own design.

This datafile containing your picture may then be used on its own as a title page to a program or, with a little ingenuity, included within another program.

In each case you start with a blank screen and build up your picture using a number of drawing options. These options vary slightly with each package and are called up either by single key commands or by using a joystick to direct a pointer to on-screen icons and pressing the fire button to initiate the command.

Lines may be drawn either by joining two specified points or freehand. Also a range of shapes may be available such as circles, boxes, triangles or arcs. Using these it is possible to draw almost anything. Of course nothing is perfect and although having 256 separate pixels across the screen may seem an awful lot, it isn't really.

You will find that some lines that should be a smooth curve look somewhat stepped. The answer is to plan ahead. Get yourself some graph paper or one of the specialised pixel pads and take a bit of time with your designs before switching on your computer. A little time spent at this stage creating a workable design, will pay handsome dividends later when it actually comes to creating the real thing.

There is one other problem you have to think about – colours. Or more to the point, the actual meeting of different colours on the screen. As you are probably aware, the Spectrum will only permit two colours to exist in any one character block. This is no real

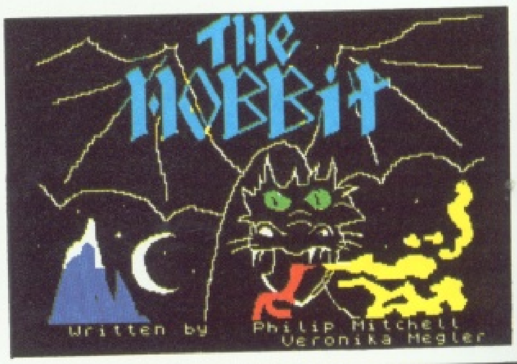
Melbourne Draw

One of the original graphics programs for the Spectrum. Although some of the newer titles offer more options, it is still a very useful utility. You are limited to the 22 UDG characters permitted by the Spectrum but once drawn on to your work area you may redefine them and create some more.

As you can grab UDGs from this work area, you can easily simulate the cut and paste operation offered in other programs. No predesigned textures are provided but again use of the UDG editor will supply an alternative approach.

The instructions are clear and informative, with the necessary information to include either the UDGs or complete screens in your own programs. Microdrive users will also find sufficient data to enable them to think about transferring the program.

● *Produced by Melbourne House Software.*



Paint Plus

An upgraded version of the very successful Paintbox of a year or two back. P'n'P are really into graphics and have a suite of four programs that will enable you to do almost anything with your Spectrum's screen.

Paint Plus has a large number of facilities and provides access to a number of banks of UDGs. The instruction booklet covers all aspects clearly and has a number of basic routines that will help incorporate either UDG or full screen graphics into the user's programs – even to showing methods for simple but effective animation.

The basic drawing aids are generally extended over those in Melbourne Draw, offering a degree of convenience rather than anything that is remarkably different. The ability to draw radii from a fixed centre and to be able to draw arcs without having to draw a circle first is a definite plus.

Undo is also a useful addition. Make a mistake, and Undo will erase the last thing you did. This can save minutes of painstaking work erasing individual pixels. Full instructions are given for adaptation for use with microdrives.

Screen Machine extends the facilities of Paint Plus (the basic graphics package) together with options to compress graphics and text data to fit into less memory, flip screen animation and more besides.

Art-O-Matic is a graphics drawing program coupled to a graphics compiler that will compress graphics data by up to 90 per cent, enabling many graphics screens to be called from within a program. Without compression a normal Spectrum screen takes 6912 bytes.

This is no program for the novice. It is not a user-friendly program for creating on screen graphics as are the others mentioned here. However it is for serious programmers wishing to extend the Spectrum to its limits.

The last offering from P n'P is **Sprite Machine,** an excellent package for those who want to include sprites in their programs.

● *For more details write to Print n'Plotter, 19 Borough High Street, London SE1 9SE. Make sure you include a stamped addressed envelope. By the way, they also produce some nice pixel pads.*

drawback as proved by many of the title screens you will have seen, but, you *must* plan ahead.

This is where the specialised pixel pads come into their own. In addition to having small squares representing each pixel, they also have each character block of 8 × 8 pixels outlined with a slightly thicker line. Practice makes perfect and you will soon learn to juggle the colours of your designs to suit the Spectrum. Most of the instruction books included with graphics packages give a few hints on how to get the best out of the Spectrum's system – read them carefully.

A very necessary facility is to be able to call up some sort of grid on to the screen so that you can check that the attribute cells – character blocks – are where you think they are. Another extremely useful option is to have the X and Y coordinates displayed. Without these it is difficult to readily follow a previously drawn pattern from your pixel pad.

Some packages allow you to scroll – or push – your picture around the working area to enable a better match with the attribute cells, but there is really no substitute for prior planning on paper.

When it comes to filling your outline drawings with colour, you must be absolutely sure your line drawings form a continuous border around the area to be filled. The slightest gap will allow the "filling" to ooze out all over the place.

Lines may often be drawn with different pens or brushes, giving a thicker or thinner line. There may also be the option of filling or painting with different patterns or textures. Some even offer the use of a "spray can" – the more you spray, the denser the painting.

User-defined graphics characters, may be created on

Icon Graphix

A joystick or keyboard driven program that is fun to use. Sadly it does not have the facilities needed to make it a worthwhile investment if you seriously want to design and use your own graphics.

It is, however, fairly easy to use and although the instructions are brief, they cover what is there. There are no UDGs and strangely no text option – other than drawing your own. The work area is surrounded by different texture patterns or icons indicating the various options. These may be actioned by moving a pointer over them and pressing the fire button or Enter.

A nice feature is the Eraser. This provides an eraser one character block in size, that can be guided over areas you wish to be rubbed-out. Neat, especially for the fumble-fingered.

● *Produced by ASL.*

a separate editing screen for later use in the working area.

Some packages even allow UDGs to be grabbed from part of your main picture. This is very useful when you are creating repetitive designs and are unable to visualise what you want when just using a UDG editor.

Text will almost certainly be needed on title screens – here you may have the option of different founts, different sized characters and even the facility to rotate the characters in order to have text running up and down the screen instead of the normal left to right. Perhaps a font editor similar to the UDG editor will give you some ideas for your own calligraphy.

There will also be an opportunity to magnify parts of the work area so that crucial parts of the design can be cleaned up pixel by pixel. Do you want to cut out part of your picture and paste it elsewhere, perhaps changing its size – rescaling – as you do this? Many things are possible – all you need are ideas and a little patience.

Which package?

Five currently available graphics designer packages are shown in our table. There are others, so use the table as a basis of comparison. All will offer the basic requirements to produce your own pictures and although it may seem that Carton X does a lot more than Carton Y, there are many ways that the facilities overlap.

Often it is a case of convenience. You can draw a box by drawing four separate lines but it's just that bit easier to call up a box command. On the other hand X and Y coordinates are vital if you are going to work from an accurately thought-out master drawing on paper.

Two's company . . .

All the graphics toolkits, art studios, designers – call them what you will – allow you to SAVE your masterpiece on to tape in the form of a SCREEN$ file. Part of the Spectrum's memory (from address 16384 to 23296) is the information displayed on the screen. LOAD this data back into the Spectrum – with or without the graphics program – and your original picture will be re-created.

Not only does this mean that you can use this as a title screen but it also means that as you become more ambitious, you can LOAD this picture into another graphics package and make use of facilities the first did not have.

This is a very important consideration, as none of these packages gives you all the facilities available. They are all useful – some more than others – but to get the most out of your imagination, you may well eventually consider buying two packages.

The Art Studio

This program is controlled by joystick – or keyboard, or mouse and is a very professional package indeed.

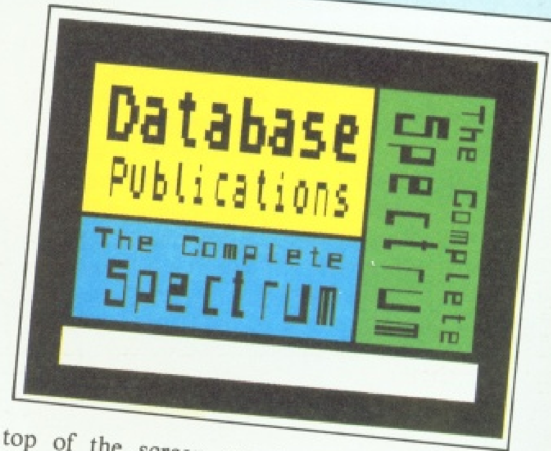
Right from the initial questions – that include whether you are using an 80-column dot-matrix printer and if so with which of 18 different interfaces – comes the feeling that this program knows what it is about. After a little more LOADing you are asked if you wish to save the configuration to tape. This allows a customised version for your printer or joystick to be SAVEd for future use.

Next comes the tricky bit. The Art Studio is protected against pirating by the Lenslok system. This uses a distorting lens system to decode two characters displayed on the screen. If you get the decoding wrong three times the computer reverts to a cold start condition. If you take too long in deciding what to enter it also drops out.

If you do not get this operation right you have to wait eight minutes to try again. It is not difficult but make sure you have that piece of plastic lens bent in the right direction – guess who didn't.

Also how many times can you bend this plastic before it breaks? Although I'm all in favour of stopping piracy, I might be somewhat upset if I could not use something I had got legitimately.

Operation using the joystick is a delight. At the top of the screen are the 11 main options. Selecting any of these pulls down a menu of further options, again selected by the joystick. As you can see from the comparison table, most requirements are catered for, with the notable exceptions of UDGs and sadly no X and Y coordinates.



Some facilities, such as the option to merge previously saved files with the current workspace and the window facility to reposition, rescale, merge or rotate a portion of the work area should be greatly appreciated.

If a microdrive is used then additional options are available, making The Art Studio a definite recommendation to any serious user.

● *Produced by Oxford Computer Publishing and distributed by Rainbird.*

Light Magic

Another easy-to-use graphics package with clear instructions. The program is not quite as versatile as Paint Plus but is probably a little easier to grasp when first starting out.

This is a useful package with some options that are not catered for in other programs. One such is to have a specified reference point, so that you can always return to exactly the same point on the screen. Another is the variable brush size and range of brush patterns – definitely worth considering.

● *Produced by New Generation Software.*

	Melbourne Draw	Paint Plus	Icon Graphix	Art Studio	Light Magic
Background grid	✓	✓		✓	✓
Lines/points	✓	✓	✓	✓	✓
Different line sizes			✓	✓	✓
Radii		✓		✓	✓
Circles	✓	✓	✓	✓	✓
Arcs		✓		✓ *	
Boxes		✓	✓	✓	
Triangles				✓	
Fill	✓	✓	✓	✓	✓
Scroll attribute screen	✓	✓			✓
Different brush sizes			✓	✓	✓
Tints/shades (spray)		✓	✓	✓	
Textures		✓	✓	✓	✓
UDG editor	✓	✓			✓
UDGs from work area	✓	✓			
Multiple UDGs (banks)		✓			✓
Font editor		✓		✓	✓
Text	✓	✓		✓	✓
Rotate (character)	✓	✓		✓	✓
Eraser			✓		
Copy (to printer)		✓	✓	✓	
Save	✓	✓	✓	✓	✓
Load	✓	✓	✓	✓	✓
Merge		✓ +		✓	
Include in Basic progs	✓	✓		✓	✓
Magnify	✓	✓	✓	✓	✓
Coordinates (X, Y)	✓	✓			✓
Undo		✓	✓	✓	✓ –
AMX Mouse compatible				✓	
Joystick cursor operation		✓	✓	✓	✓
Screen memory compression		✓ +		✓ *	✓
Cut and paste				✓	✓
Rescale	✓	✓ +		✓	✓
Microdrive compatible		✓	✓	✓	

Note: + with Print n'Plotter Screen Machine
** Extended Art Studio used with microdrive*
– Work area may be stored and returned to if an error is made

Character boosters

We look at four special commands which can enliven your screen displays

AS we've already seen, the commands INK and PAPER allow us to change the colour of the characters displayed on the screen. There are also four special commands which affect the way in which a character is displayed: BRIGHT, FLASH, INVERSE and OVER. Each command is followed by either 0 or 1 which indicates if the function is on (1) or off (0). Therefore, the following instructions turn on (enable) the corresponding display options:

```
BRIGHT 1
FLASH 1
INVERSE 1
OVER 1
```

And by changing the 1s to 0s they can be turned off (disabled):

```
BRIGHT 0
FLASH 0
INVERSE 0
OVER 0
```

As with INK and PAPER, the commands can be nested within PRINT and INPUT statements to temporarily change the display options.

BRIGHT

BRIGHT alters the brightness of the INK and PAPER colours. If BRIGHT 1 is used all subsequent symbols will appear brighter than usual. The normal brightness can be reselected with BRIGHT 0.

Program I uses BRIGHT to alter the brightness of two messages.

Lines 20 and 30 select blue INK and cyan PAPER. BRIGHT 0 is then used to reset the brightness to normal. The word NORMAL is now printed. The next message, BRIGHT, is printed following a BRIGHT 1 instruction, and as the message indicates, the foreground and background colours are now brighter.

You should note that the brightness of black will be no different following either BRIGHT 0 or BRIGHT 1. Think about it!

```
10 REM PROGRAM I
20 INK 1
30 PAPER 5
40 BRIGHT 0
50 PRINT AT 10,12;"NORMAL"
60 BRIGHT 1
70 PRINT AT 11,12;"BRIGHT"
```

Program I

If you wish to use BRIGHT with the graphics instructions – DRAW, PLOT and CIRCLE – the BRIGHT statement must be nested within the chosen command. Thus:

```
CIRCLE BRIGHT 1,50,50,20
```

will work correctly, but:

```
BRIGHT 1
CIRCLE 50,50,20
```

will not.

FLASH

One way of attracting someone's attention is to use FLASH. When FLASH is enabled with FLASH 1 all new symbols displayed will alternate between their foreground (INK) and background (PAPER) colours. Thus a red message on a white background will change to a white message on a red background and then back to its original state at a predefined rate. Program II shows this example in action.

```
10 REM PROGRAM II
20 INK 2
30 PAPER 7
40 FLASH 1
50 PRINT AT 10,9;"URGENT MESSAGE"
60 FLASH 0
```

Program II

The final line of the program disables the FLASH feature with FLASH 0.

As with BRIGHT, FLASH will only work correctly with the graphics commands if the FLASH statement is nested in the graphics instruction itself.

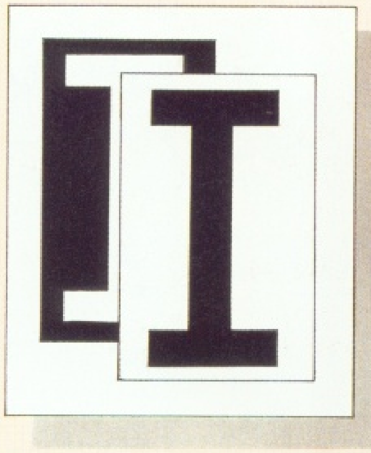
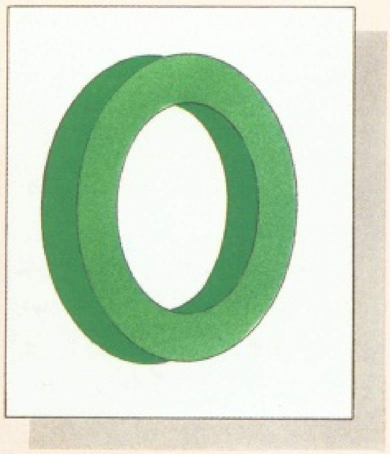
Program III demonstrates this quirk. The FLASH 1 command in line 70 doesn't affect the CIRCLE command of the next line. The program was designed to draw two circles in the centre of the screen. The outermost one is supposed to flash, whereas the other is not.

Line 50 ensures the flash option is off before drawing the inner circle. Next the flash is enabled and another circle is drawn with a larger radius than the previous one. Once again, the final line turns FLASH off.

However the outer circle doesn't flash between its INK and PAPER colours. But if you delete line 70 and incorporate the following line to the program everything will be all right.

```
80 CIRCLE FLASH 1,128,88,40
```

Sometimes when you want to add a bit of contrast to the screen without keeping track of the individual

```
10 REM PROGRAM III
20 INK 3
30 PAPER 7
40 CLS
50 FLASH 0
60 CIRCLE 128,88,25
70 FLASH 1
80 CIRCLE 128,88,40
90 FLASH 0
```

Program III

colours it is useful to swap the INK and PAPER colours. This can be done with the INVERSE command.

INVERSE

Once INVERSE 1 has been executed the INK colour is used for the background and the PAPER for the foreground. For example, if the INK is 2 (red) and the PAPER is 6 (yellow) INVERSE 1 will switch them round so that the INK becomes yellow and PAPER red.

When INVERSE 0 is encountered the INK and PAPER colours are restored to their original settings.

Program IV alters the INVERSE option, and thus

```
10 REM PROGRAM IV
20 INK 7
30 PAPER 1
40 INVERSE 1
50 PRINT CHR$ (65);
60 INVERSE 0
70 PRINT CHR$ (65);
80 GO TO 40
```

Program IV

the INK and PAPER colours, while it prints a string of letter As.

It's worth noting that the INVERSE command doesn't work with the graphics commands.

OVER

One of the most useful Basic graphics commands is OVER. When OVER 1 is used the symbol being displayed is combined with the screen position being written to. In fact the way in which they are combined is quite unusual as we'll see.

The pixels which make up the character definition are Exclusively ORed with the pixels at the screen location being accessed. The Exclusive OR operation determines whether the resulting pixel will be set or clear – that is, filled with the foreground or background colour respectively – by comparing the two corresponding pixels of the new and previous character.

Figure I lists the possible outcome for the four different conditions.

If the result bit is set the pixel appears in the current INK colour, otherwise it becomes part of the background.

From Figure I you can see that the resulting pixel will be displayed in the INK colour only if one of the

Old pixel	*New pixel*	*Result*
unset	unset	unset
unset	set	set
set	unset	set
set	set	unset

Figure I: Result pixels for Exclusive OR

two pixels is set. If both are set, or both clear, the resulting pixel will appear in the background colour.

Let's assume a character has been displayed on a blank screen with OVER 1. If the same character is printed on top of the previous one it will disappear. This is because of the effect Exclusive OR has on the pixels.

When the first character is printed the screen is clear, so the character appears in the usual way. However when the character is printed again the set and unset pixels of the old and new symbol overlap exactly.

Since the old and new are being Exclusive ORed, the matching set pixels combine to give a resulting unset pixel. And, of course, the unset pixels also combine to give unset pixels so the old image disappears.

To put it another way, the resulting pixel will always be unset if the old and new pixels being Exclusive ORed are identical – see Figure I. If you still don't believe me type in Program V and see.

Lines 20 to 40 select the INK, PAPER and OVER options. Line 50 prints the word HELLO near the centre of the screen – remember, we're using OVER 1. The next lines acts as a short delay. Line 80 causes the program to restart from line 50. HELLO is printed once more – directly on top of its previous position. Since we're Exclusive ORing it disappears. Next time round, HELLO is printed at the same position on a blank screen, thus reappearing, and so on.

The OVER command can be put to good use in animation and games. This is because objects can be moved over background data without destroying the display.

The next program, Program VI, moves a letter X across the screen without destroying any objects – text in this example – along its path.

The "magic" is all due to Exclusive OR. As we've seen already, OVER 1 allows a shape to be displayed or erased with one simple PRINT statement. This principle goes further, because the character can be displayed and then erased while on top of another

```
10 REM PROGRAM V
20 PAPER 7
30 INK 2
40 OVER 1
50 PRINT AT 10,12;"HELLO"
60 FOR I=1 TO 100
70 NEXT I
80 GO TO 50
```

Program V

character, leaving the original character still in place.

It's important to remember that the outcome is the result of combining the two characters in Program VI. In other words, when we first Exclusive OR X on to one of the background characters we see the X combined with the character underneath, according to the Exclusive OR rules. This can lead to side-effects.

In Program VI the background message is "IT'S AMAZING!". When, for instance, the X moves on to the !, several pixels disappear. You will notice that the pixels which have vanished are the ones which are set in both the X and !. This is the side effect. If you think about it you'll see that this is exactly what we'd expect

```
10 REM PROGRAM VI
20 INK 2
30 PAPER 7
40 CLS
50 OVER 1
60 PRINT AT 11,8;"IT'S AMAZING!"
70 FOR C=31 TO 0 STEP -1
80 PRINT AT 11,C;"X"
90 FOR L=1 TO 250
100 NEXT L
110 PRINT AT 11,C;"X"
120 NEXT C
```

Program VI

with Exclusive OR, since the resultant pixel is zero when both the old and new pixels are set. The pixels which have been removed will be replaced when the X is printed the second time – line 110.

OVER is the only command that can be used nested or un-nested with the graphics commands. Thus:

```
10 OVER 1
20 CIRCLE 70,50,20
```

and:

```
10 CIRCLE OVER 1,70,50,20
```

both have the same effect.

As you can see OVER can be most useful as proved by the fact this Exclusive OR technique is used in many commercial games.

H s & Tips Hints & Tips Hints &

THE Spectrum lacks a command to delete a block of Basic lines. You can remove a whole program with the NEW command or delete a single line by entering just the line number on its own, but to get rid of a number of unwanted lines can be a tedious affair.

However there is a quick way to delete a large chunk of program by poking the system variables. Because of the way it works you don't have a great deal of choice in the block of lines that gets deleted, but it can save a lot of typing in the appropriate circumstances.

You can only delete a block of lines from the end of a program with this trick. That is, you can delete from any chosen line to the end of the program, but you cannot remove a block from the front or out of the middle of a program.

The system variables used for this are called NXTLIN and VARS in the original Spectrum manual. NXTLIN holds the address of the next line in the Basic program (next from the one that the Basic is running at any given moment), and VARS holds the address of the Basic variables storage area, which also happens to be the end of the program.

All we have to do is insert a line just before the one where we want our block delete to start, to PEEK the address of the next line and POKE it into VARS, and then do a CLEAR.

The CLEAR causes Basic to clear the Basic variables (not to be confused with the system variables) from the beginning marker (which we have just changed with our POKEs) to the end marker, and start a new variable area at the new beginning marker.

What we have done is to fool it into clearing out a lot of Basic lines as well as variables.

The whole thing has to be done in one line to prevent scrambling the NXTLIN address, but it is not very long. Try this:

```
  10 REM
  20 REM
  30 REM
  40 POKE 23627,PEEK 23637: POKE
23628,PEEK 23638: CLEAR
  50 PRINT "DONE!"
  60 REM
  70 REM any number of lines
  80 REM will be removed
  90 REM from 50 onwards
 100 REM
```

When you run this, the PRINT instruction and the following lines will disappear before the program finishes with line 40, so the word "DONE!" never gets printed.

Line 40 is all you need for this trick, but of course you must type it in with an appropriate line number to insert it just before the first line you wish to remove in your own program.

EAST

SAI COMBAT

Set against a series of subtle backgrounds, Sai Combat has 8 skill levels and can be played by one or two players, with a choice of 16 moves. The combatant sprites are large and easy to control via keyboard or joystick.

Spectrum 48K £6.95

OR WEST

A Crash! Smash

Rev up your engine and take to the skies to save your country. Hunt down the enemy and build up your log of kills as you rise through the ranks to the elite of the RAF.

Spitfire 40

Spectrum 48K	**£9.95**
Amstrad CPC	**£9.95**
Commodore 64	**£9.95 tape, £12.95 disk**

PLUS

New expanded 128K Spectrum version with a full mission mode, improved sound, and more ground detail. Still only £9.95!

ONLY THE BEST

FROM

MIRRORSOFT

Purnell Book Centre, Paulton, Bristol BS18 5LQ

QUICK TO LEAR

THAT'S...

FINALIST
BRITISH MICRO COMPUTING AWARDS
1985

MINI OFFICE

JUST LOOK WHAT THIS PACKAGE CAN DO!

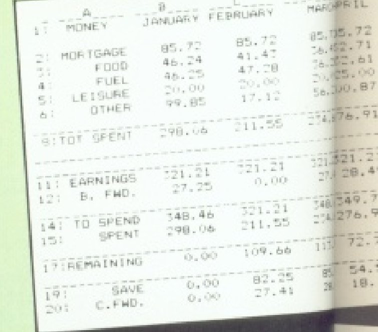
WORD PROCESSOR – Ideal for writing letters or reports! *Features:* Constant time display ● Constant word count (even shows words per minute) ● Normal or double-height text on screen or printout.

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.

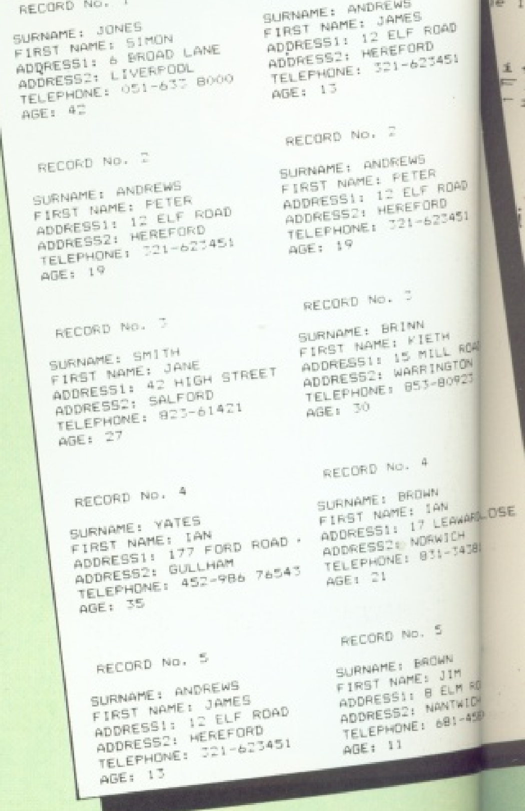
DATABASE – Use it like an office filing cabinet! *Features:* Retrieve files at a keystroke ● Sort ● Replace ● Save ● Print ● Search.

SPREADSHEET



```
       A            B        C          D
 1:  MONEY       JANUARY  FEBRUARY    MARCAPRIL

 2:  MORTGAGE      85.72     85.72
 3:      FOOD      46.24     41.47
 4:      FUEL      46.25     47.28
 5:  LEISURE       20.00     20.00
 6:     OTHER      99.85     17.12

 8:TOT SPENT      298.06    211.55

11: EARNINGS      321.21    321.21
12:   B. FWD.      27.25      0.00

14: TO SPEND      348.46    321.21
15:     SPENT     298.06    211.55

17:REMAINING        0.00    109.66

19:      SAVE       0.00     82.25
20:   C.FWD.        0.00     27.41
```

DATABASE



```
RECORD No. 1                     RECORD No. 1

SURNAME: JONES                   SURNAME: ANDREWS
FIRST NAME: SIMON                FIRST NAME: JAMES
ADDRESS1: 6 BROAD LANE           ADDRESS1: 12 ELF ROAD
ADDRESS2: LIVERPOOL              ADDRESS2: HEREFORD
TELEPHONE: 051-637 8000          TELEPHONE: 321-623451
AGE: 42                          AGE: 13

RECORD No. 2                     RECORD No. 2

SURNAME: ANDREWS                 SURNAME: ANDREWS
FIRST NAME: PETER                FIRST NAME: PETER
ADDRESS1: 12 ELF ROAD            ADDRESS1: 12 ELF ROAD
ADDRESS2: HEREFORD               ADDRESS2: HEREFORD
TELEPHONE: 321-623451            TELEPHONE: 321-623451
AGE: 19                          AGE: 19

RECORD No. 3                     RECORD No. 3

SURNAME: SMITH                   SURNAME: BRINN
FIRST NAME: JANE                 FIRST NAME: KIETH
ADDRESS1: 42 HIGH STREET         ADDRESS1: 15 MILL ROA
ADDRESS2: SALFORD                ADDRESS2: WARRINGTON
TELEPHONE: 823-61421             TELEPHONE: 853-80923
AGE: 27                          AGE: 30

RECORD No. 4                     RECORD No. 4

SURNAME: YATES                   SURNAME: BROWN
FIRST NAME: IAN                  FIRST NAME: IAN
ADDRESS1: 177 FORD ROAD          ADDRESS1: 17 LEAWARD CLOSE
ADDRESS2: GULLHAM                ADDRESS2: NORWICH
TELEPHONE: 452-986 76543         TELEPHONE: 831-3438
AGE: 35                          AGE: 21

RECORD No. 5                     RECORD No. 5

SURNAME: ANDREWS                 SURNAME: BROWN
FIRST NAME: JAMES                FIRST NAME: JIM
ADDRESS1: 12 ELF ROAD            ADDRESS1: 8 ELM RO
ADDRESS2: HEREFORD               ADDRESS2: NANTWICH
TELEPHONE: 321-623451            TELEPHONE: 681-458
AGE: 13                          AGE: 11
```

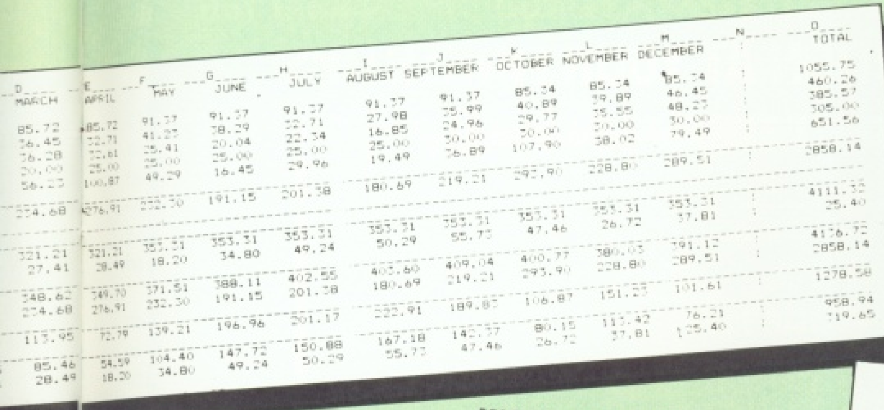
Page

Th
MI
sh
op

...and it's all at t
price of just £

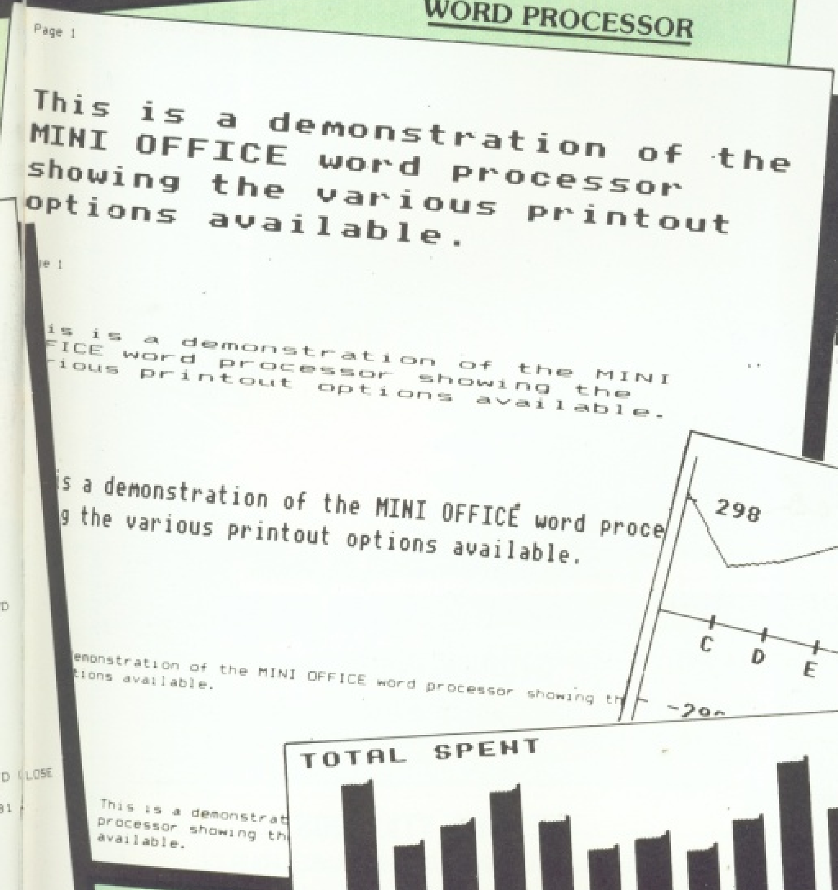
RN, EASY TO USE

Spectrum 48k and Spectrum+



D MARCH	E APRIL	F MAY	G JUNE	H JULY	I AUGUST	J SEPTEMBER	K OCTOBER	L NOVEMBER	M DECEMBER	N	O TOTAL
85.72	85.72	91.37	91.37	91.37	91.37	91.37	85.34	85.34	85.34		1055.75
36.45	32.71	41.27	38.29	32.71	27.98	35.99	40.89	39.89	46.45		460.26
36.28	32.61	25.41	20.04	22.34	16.85	24.96	29.77	35.55	48.27		385.57
20.00	25.00	25.00	25.00	25.00	25.00	30.00	30.00	30.00	30.00		305.00
56.23	100.87	49.29	16.45	29.96	19.49	36.89	107.90	38.02	79.49		651.56
234.68	276.91	232.30	191.15	201.38	180.69	219.21	292.90	228.80	289.51		2858.14
321.21	321.21	353.31	353.31	353.31	353.31	353.31	353.31	353.31	353.31		4111.32
27.41	28.49	18.20	34.80	49.24	50.29	55.73	47.46	26.72	37.81		25.40
348.62	349.70	371.51	388.11	402.55	403.60	409.04	400.77	380.03	391.12		4136.72
234.68	276.91	232.30	191.15	201.38	180.69	219.21	293.90	228.80	289.51		2858.14
113.95	72.79	139.21	196.96	201.17	222.91	189.83	106.87	151.23	101.61		1278.58
85.46	54.59	104.40	147.72	150.88	167.18	142.37	80.15	113.42	76.21		958.94
28.49	18.20	34.80	49.24	50.29	55.73	47.46	26.72	37.81	125.40		319.65

WORD PROCESSOR



Page 1

This is a demonstration of the MINI OFFICE word processor showing the various printout options available.

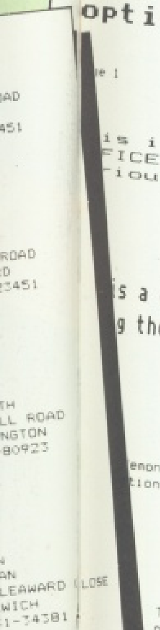
e 1

is is a demonstration of the MINI
FICE word processor showing the
rious printout options available.

is a demonstration of the MINI OFFICE word proce
g the various printout options available.

emonstration of the MINI OFFICE word processor showing th
tions available.

This is a demonstrat
processor showing th
available.



AD
451

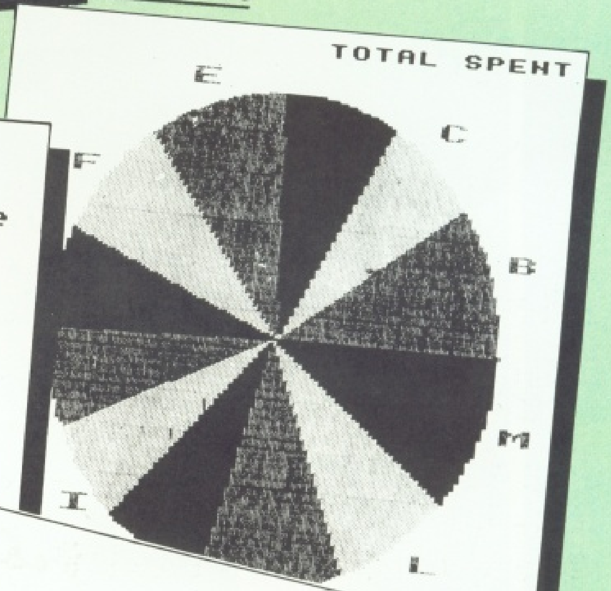
ROAD
D
23451

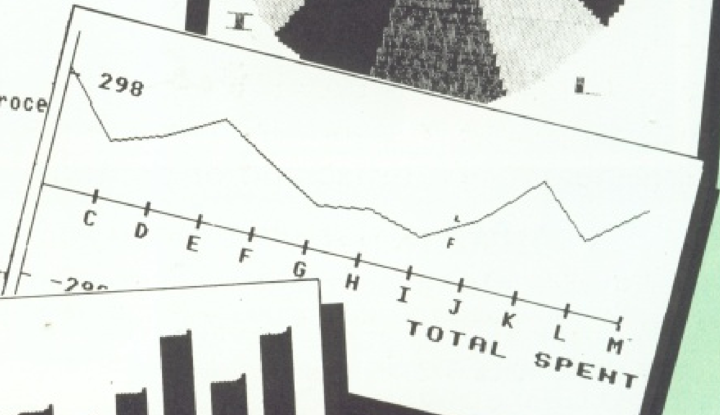
TH
LL ROAD
NGTON
80923

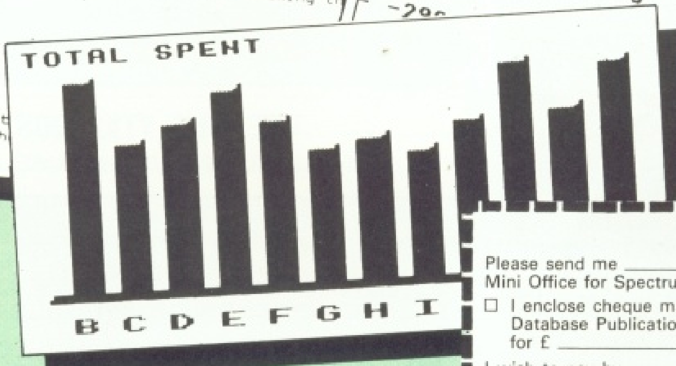
N
AN
LEAWARD CLOSE
WICH
1-34381

5
OWN
JIM
ELM ROAD
ANTWICH
681-45800

GRAPHICS







at the unbelievable
st £5.95 CASSETTE

DATABASE SOFTWARE

Only £5.95

Please send me ________ copy/copies of Mini Office for Spectrum 48k/Spectrum+	
☐ I enclose cheque made payable to Database Publications Ltd. for £ ________	
I wish to pay by ☐ Access ☐ Visa No. ________	Expiry date ________
Signed ________	
Name ________	
Address ________	

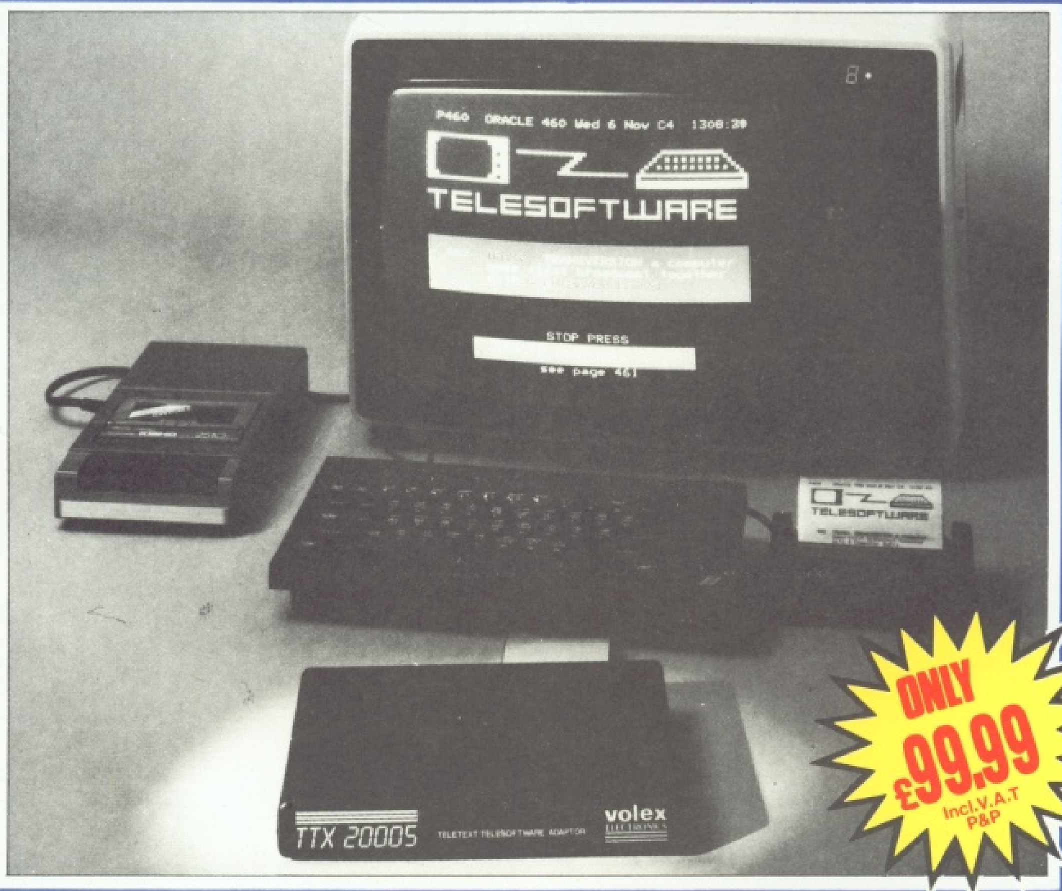
Post to: Mini Office Offer, Database Publications, 68 Chester Road, Hazel Grove, Stockport SK7 5NY.

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


P460 ORACLE 460 Wed 6 Nov C4 1308:28
TELESOFTWARE
STOP PRESS
see page 461
TTX 2000S
TELETEXT TELESOFTWARE ADAPTOR
volex ELECTRONICS

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 ______________________

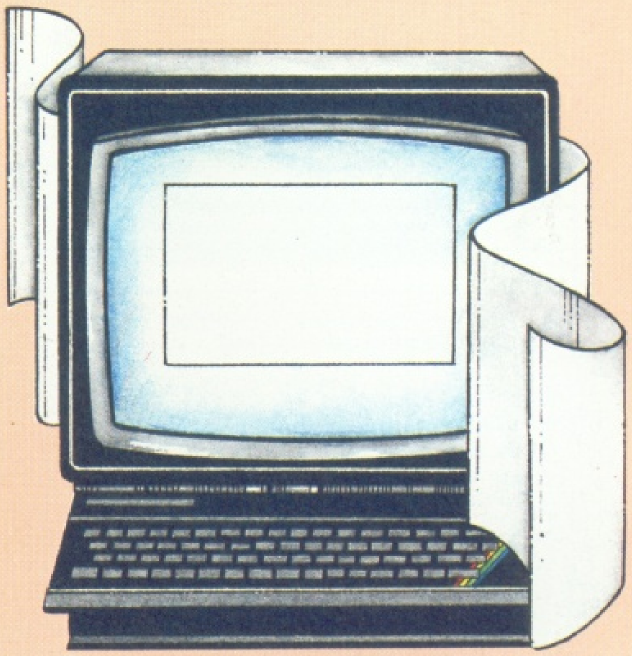
____

TTX2000S

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

Spreadsheets

What they are and how to get the best results out of them



The Spectrum is best known as a games machine but computers, as we all should know, can be very versatile tools as well. One of the most versatile tools to be found on nearly all computers is a form of spreadsheet program. Visually this presents the user with a grid of boxes something like those shown in Figure I.

Depending on your requirements – and the size of the computer's memory – this grid could contain several thousand boxes. It follows that if you are to insert any readable data into these boxes, each individual box must be several characters wide. So, what will be seen on the screen, at any one time, will only be a portion of the total picture.

If you prepared a similar grid on paper, you would have to *spread* the *sheet* of paper over a very large table to view it all.

The boxes are arranged in rows (horizontally) and columns (vertically). The rows are labelled alphabetically and the columns numerically, so that every box has a unique code of its own. Looking at Figure I you will see that box X is at position A1 and box Y is at position K5. If you were to look further down you would find that after row Z would be rows AA, AB, AC and so on. After AZ would follow BA, BB, BC and so on.

Little boxes

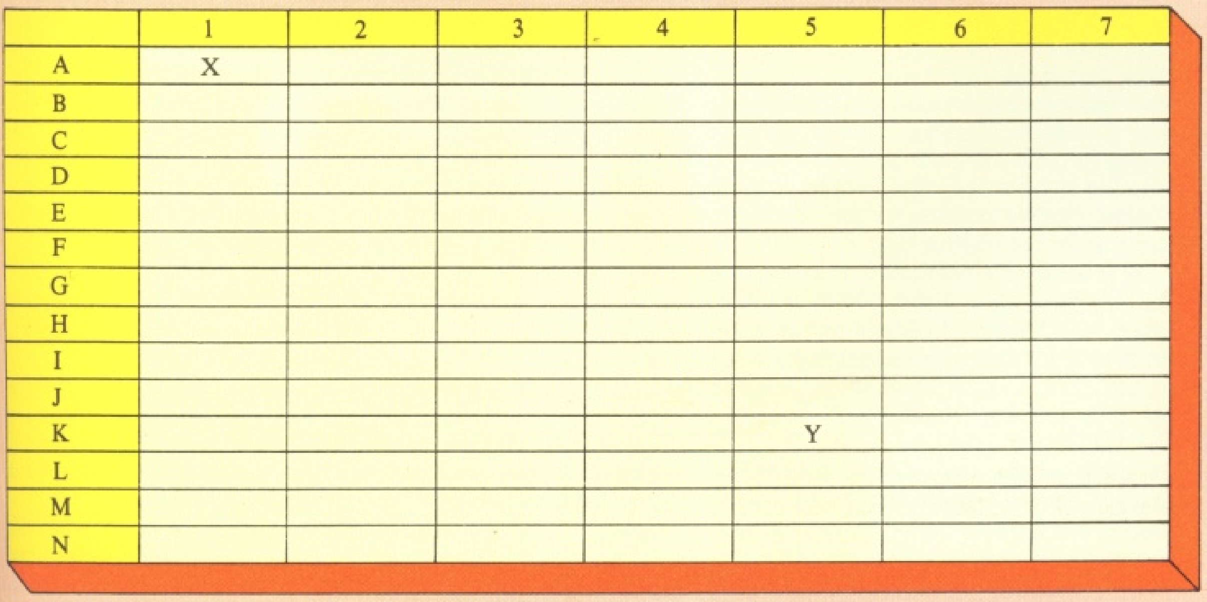
Having got ourselves several hundred empty boxes, what can we put in them and what will the program do with these inputs? The answers to these questions are interlinked, so let us look at what a few entries in some of these boxes will achieve.

The pocket calculator has caused a revolution, not only for the student striving to finish an evening's homework but also among all those who deal in figures, however simple or complex.

But the calculator normally only produces one answer at a time and there are many problems that require a range of answers to be displayed for them to make sense. The spreadsheet is the solution to these criticisms and can very well be looked on as a super calculator.

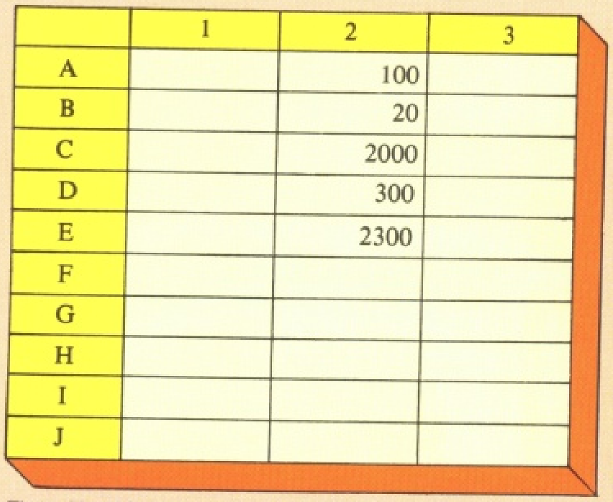
Figure IIa shows just a small window of our spreadsheet. There will be some form of cursor visible to indicate the box which the program is currently either looking at, or expecting you to give an input.

This cursor can be moved around as you choose. Let us enter the numeric value 100 in box A2. Now move the cursor down to position B2. Here we will enter a value of 20. Next a simple formula – A2×B2 –



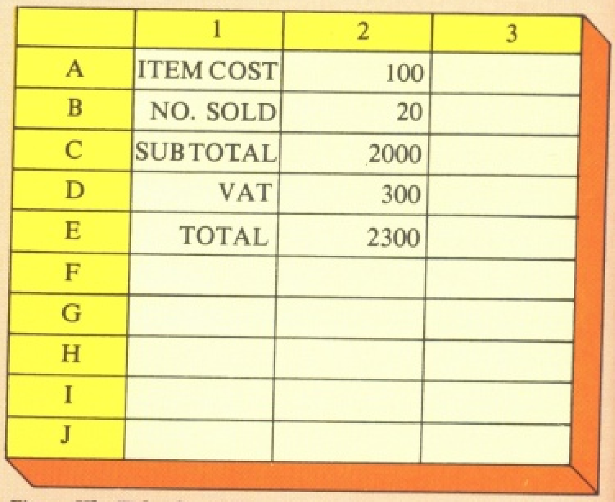
	1	2	3	4	5	6	7
A	X						
B							
C							
D							
E							
F							
G							
H							
I							
J							
K					Y		
L							
M							
N							

Figure I: Spreadsheet grid



	1	2	3
A		100	
B		20	
C		2000	
D		300	
E		2300	
F			
G			
H			
I			
J			

Figure IIa: What you can put in the boxes



	1	2	3
A	ITEM COST	100	
B	NO. SOLD	20	
C	SUBTOTAL	2000	
D	VAT	300	
E	TOTAL	2300	
F			
G			
H			
I			
J			

Figure IIb: Titles for the boxes

causes the answer 2000 to appear in box C2. The spreadsheet has acted just like a calculator.

If we now move the cursor to box D2 and enter the formula C2×0.15, D2 will display 3000. Then in E2 enter C2+D2 and we will have 2300. So far so good – just a calculator working out the cost of 20 items costing £100, the VAT on those items and the total including VAT. (You'd guessed that, hadn't you?)

Perhaps we could now use the first column to enter text to tell us what the boxes mean, so we get Figure IIb. But what if we now change the number of items sold in box B2 to 23? Nothing happens – until you tell the spreadsheet to recalculate. You will then find that all the boxes have been updated with the correct answers.

We now know the answer to our first question – you can enter numbers, formulae or text. Already you should be able to think of some of the possibilities of working with a spreadsheet. The above example could be immediately improved, as in Figure III, where one can enter and alter individual items and costs – but there is more yet.

Encore . . . encore . . .

One facility that all spreadsheets have is the ability to repeat the contents of a box – often called a cell – along a row, down a column or even over a rectangular block of cells. Enter the value 10 in B3, repeat this down the column to cell K3 and the value 10 will appear in all cells between and including B3 and K3.

You may also repeat formulae using this same facility. But spreadsheets work on a relative position basis – this means that unless told otherwise, when you repeat a formula involving another cell or cells, the repeated formula will have the same form but the cells referred to in the new formula will be those that are in the same relative position to the new cell as those referred to in the original cell.

This is a bit of a mouthful but an example should make this quite clear. Enter 100 in cell A3, then the formula A3×1.1 in cell B3 and repeat this down the column to K3. The answers appearing in column 3 will run: 110, 121, 133, 146 and so on. The formulae in these cells starting with B3 will now be A3×1.1, B3×1.1, C3×1.1, D3×1.1.

In other words, the original formula in B3 was "Multiply the cell immediately above by 1.1". So in repeating this formula down the column, each cell has the same form but instead of them all having A3×1.1, they have (the cell above)×1.1.

This enables a whole series of interlinked calculations to be performed very easily and makes the spreadsheet a very powerful tool. Although the example given above is very simple, it gives some idea of what can be done. It also tells you what would happen to £100 invested in a building society with an interest rate of 10 per cent for the next 10 years!

Again this example can be expanded to make it more flexible. The interest rate (0.1) could be entered in a separate cell, making it easier to change. If this were done, say at A5, then the formula at B3 would look something like (A1×kA5)+A1.

The kA5 signifies that the value in cell A5 is a constant and will appear throughout a repeat of this formula as the value in A5 and not A5, B5, C5 and so on.

Some spreadsheets allow the use of the conditional operators – AND, OR and NOT. This enables the use of formulae that effectively say such things as "If X is not greater than Y then Z=...". Couple this with the facility to repeat calculations until a certain condition is met and one can begin to see why spreadsheets form the backbone of many commercial and analytical operations.

Other commands you would expect to find would be the facility to transfer the contents of one row or column to another row or column, the choice of integer or floating point numbers, a rapid means of moving the cursor about in a large spreadsheet and, of course, the ability to save and load datafiles for the different spreadsheets you were working on.

The uses of spreadsheet programs are not always immediately obvious and only when you start working with them do you realise the potential they hold for any number of projects. Two such programs available for the Spectrum are Vu-Calc and Omnicalc 2.

Vu-Calc came out shortly after the birth of the Spectrum and is produced by Psion. Omnicalc 2 is from Microsphere Computer Services and is the latest version of this highly acclaimed program.

Vu-Calc: a firm favourite

Although Vu-Calc is now a bit long in the tooth, there are many people who swear by it. It is very easy to use and has all the commands mentioned above with the exception of the use of the conditional operators and the automatic repeat calculation mode.

The spreadsheet size using a 48k Spectrum is 60×60 cells and as it was originally produced before the introduction of microdrives, this is an option that could well persuade some to look elsewhere. Movement of the cursor is either by Caps Shift and keys 5, 6, 7, 8 on the Spectrum or just the cursor keys on the Spectrum Plus.

Entering data and text is particularly easy. The only special requirement is to start a text input with quotes, otherwise the program recognises what is being input and reacts accordingly. Should a mistake be made (by the user of course!) then the program ceases with an error message and the user has to enter G (GOTO) 9000, whereupon the area where the error occurred is displayed and the program resumes – with no data lost.

One very nice feature is that as the cursor is moved around the spreadsheet, whenever it is over a cell that contains a formula, that formula is displayed below the workspace. So, having implemented a repeat command, you can actually check what has been entered in each cell, or if altering a cell's contents, you can study what you had previously entered.

The quiescent state of the program is with the cursor keys active and an entry awaited from the keyboard. If a hash – Symbol Shift+3 – is entered then the commands available are listed above the workspace – all very sensible and straightforward.

Printing of what is on the screen is output to the ZX printer, or presumably to any other printer via a suitable interface, although this will only give a similar copy output. For the price (£8.95) Vu-Calc has got to be good value and offers the newcomer an easy entry into the world of spreadsheets.

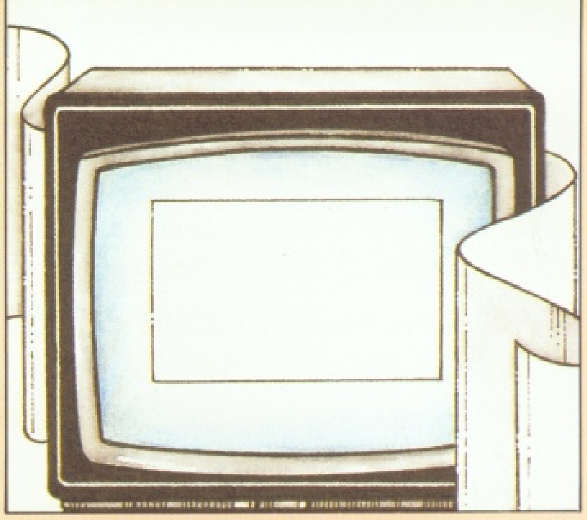
Omnicalc 2: all you need

Omnicalc 2 is microdrive-compatible and has all the functions we have mentioned so far. But in almost every case the implementation is more sophisticated, with many extra options that make the program extremely flexible in operation. As always this does mean that it will take the user a little longer to appreciate its full potential. You may design a spreadsheet with Omnicalc 2 with over 4000 cells.

In addition to text within the spreadsheet workspace, headings are permitted outside on both axes. These act as permanent reminders of what the column or row you are working on is all about and greatly assists correct entries within a large workspace.

All inputs are checked for a valid entry and in the event of an entry not being of the correct form one of a range of sensible error messages is displayed.

In addition to the ZX printer, full-size printers may be used, with the appropriate interface, and the program will use the full width of up to 132 columns. There is also the facility to draw histograms of your results and print these out – if the printer supports



graphics and there is suitable screen-copy software.

You may also extract an area from the spreadsheet workspace and save this independently. This file may then be loaded into another spreadsheet, so enabling you to readily transfer data from one program to another.

Two versions of Omnicalc 2 are included on the cassette, one of which leaves some 1500 bytes free (above 64000) in case the user requires complex printer interface software that will not fit in the 256-byte LPRINT buffer. The two programs are similar except for the smaller potential size of the latter's spreadsheet workspace.

Two further programs are included that will enable users of the earlier Omnicalc to convert their data for use with this latest version. The 30-page User Guide covers all aspects of using all these programs very clearly.

Omnicalc 2 costs £14.95 and should keep you occupied for years! There is also an expansion kit available at £6.95. For the advanced user, this has programs to allow adding customised commands to Omnicalc 2 for the use of wafa drive and disc drives, a screen-dump routine designed for a graphics printer in conjunction with the RS232 port on Interface 1 and a print out routine that may be used to print out all formulae used in the spreadsheet.

Obviously if you are a dedicated games player only, then these programs will probably not compete with the latest adventure or platform spectacular. But if you are at school, buying a house, trying to work out your year's budget or planning some investments then give one a go!

You'll be surprised at how interesting figures can be when you can see them alter in front of your very eyes.

	1	2	3	4
A	COST	SOLD	TOTALS	ITEMS
B	--------	--------	--------	--------
C	PINS	1.50	26	39.00
D	NEEDLS	3.75	16	60.00
E	THIMBLS	5.00	10	50.00
F			--------	--------
G			VAT	22.35
H				
I			TOTAL	171.35
J			--------	--------

Figure III: Greater spreadsheet detail

On the register

Making sense of machine code – Part IV of our hands-on course

THE instructions available to the machine code or assembly language programmer are very "low level" compared with the commands offered by high level languages such as Basic, Pascal, Logo, and so on.

All high level languages will, for instance, have a simple one-word command to print text on the screen, but in machine code printing even a single character may involve a long and complicated series of instructions.

In the Spectrum the screen printing routines are well over 1,000 bytes long. It is important to remember that the simple PRINT command used in Basic is only simple because someone else has written a complex machine code program to interpret and carry out the Basic command.

People sometimes get the idea that the Spectrum can run either Basic or machine code programs – but this is not quite true. The microprocessor only understands machine code. The Basic interpreter is itself a machine code program, one which converts commands that we find easy to deal with into the more demanding machine code that the computer requires.

You will probably feel that the machine code instruction set is not very powerful compared with Basic commands. The individual instructions don't seem to do very much. Well, that's true, but remember everything that Basic does is in fact done by machine code.

Basic only hides the complexity from poor simple humans, so we don't get confused. Furthermore, there are a number of things that can be done in machine code which are not possible through the Basic interpreter.

The Z80 registers

The microprocessor has a number of internal bytes of memory called registers, where it keeps numbers it is using or manipulating. We have already used the A register, also called the accumulator, and the B, C, H and L registers in our example code routines.

There are 24 registers directly accessible to the programmer, although there is usually no need to deal with more than eight or 10 of them. Almost all the machine code instructions act upon these registers and the numbers held in them.

The accumulator

The A register is the most important for the programmer. There are a number of logical and arithmetic operations that can only be performed on a number held in the A register, and the result of most binary arithmetic instructions ends up there. Associated with the A register is the F, or flag, register. This is not used to hold numbers as such, but keeps track of such things as "carries" from arithmetic performed on the A register.

Register pairs

The H and L registers are most often used together as the HL register pair. As a pair they can hold a 16-bit address, and can be used to point to any byte of memory in the Z80's 64k address space.

There are many instructions that take advantage of this and treat HL as a single 16 bit number instead of two separate bytes, but there are also instructions that use H and L separately.

H stands for high order and L for low order, reflecting the way the two registers can represent the high and low bytes of an address. This is a hangover from early microprocessors that had only one such register pair. The Z80, however, has a lot of similar registers which have necessarily had to be given less meaningful names.

The B and C registers, and the D and E registers can also be used as register pairs or as single byte registers, as above. However there are a few instructions that will only use the HL pair, or which, as with the block moving instructions, expect HL to have the "source" address and DE to have the "destination" address. The B register (and sometimes the BC pair) also has a special use with some instructions as a loop counter. The Z80 instruction set is not entirely consistent or symmetrical in the way it uses registers.

It is important to remember that the fact that you can write the assembler mnemonic does not necessarily mean the corresponding op-code actually exists. The mnemonic ADD A,(HL) has the op-code &86. This instruction will cause the byte at the address held in HL to be added to the A register.

However there is no corresponding op-code for ADD A,(BC), so if you should try to use such a construct in an assembler program, the assembler will kick it out with an error report.

The BC register has another special use in the Spectrum. When you call a code routine with LET x=USR address, you assign the variable *x* a value returned by the USR function, which will be the number held in BC at the end of the code routine. Try the hex handler (from Part Three) on this routine:

```
32000  0     7D00  00  01   LD BC,&1505
32001  0     7D01  00  05
32002  0     7D02  00  15
32003  0     7D03  00  C9   RET
```

If you now enter PRINT USR 32000, the Spectrum will print 5381, which is of course &1505. The line **LET x=USR 32000: PRINT x** will do much the same thing, except that the Spectrum also keeps the value in the variable *x* for further use. This is a handy way to pass the result of a machine code routine back to a Basic program.

Note the difference between the mnemonics using brackets around a register pair and those without. The brackets mean that the register pair is used as a pointer to a memory location. The actual operation takes place on the memory location, rather than the register. This is called indirect addressing.

The alternative register set

The Z80 has a complete duplicate set of all the above registers. There are two instructions that can be used to swap between sets – one which swaps only the A and F registers with their alternates, and another which swaps H, L, B, C, D and E.

Using the alternate registers can be confusing and dangerous, as the microprocessor has no way of telling whether the registers it is using are the originals or the alternate set. The programmer has to keep track of that. Furthermore, the Spectrum ROM uses the alternate H and L to find its way back to Basic from its calculator routines, so if you mess them up you can get a crash. They are best left alone for now.

The index registers

These are two 16- bit register pairs that are only used as pairs. You cannot access either of them as two single byte registers because the instructions don't exist. Well, maybe they do, but they are not documented officially because this is part of the chip that the chip maker could not get to work reliably – at least in the early days of Z80 manufacture.

Some people have sussed out these "illegal" instructions, and programmers do sometimes use them, but it is not a good idea as there is no guarantee they will work with every Z80 microprocessor that may find its way into a Spectrum.

Anyhow, the IX and IY registers are meant to be used for "indexed" addressing, which is a handy way of using a list or look-up table. The index register is loaded with a base address, and then instructions can act on that address plus an additional "offset". In the Spectrum, the IX register is used by the ROM cassette LOAD and SAVE routines, while IY holds the base address of a table in the system variables.

The stack

The stack pointer (SP) is another 16 bit address register with a special use. Whenever you use a CALL instruction, the computer has to store an address for the eventual RETurn. But, as we have seen, in machine code we often call routines that call other routines that call yet further routines. One register, or even half a dozen, would not be enough to keep track of the adddresses that all the calls came from.

The Z80 solves this problem by keeping a stack of addresses near the end of the RAM. Whenever you do a CALL, the address where you are is put on the stack, and then you go off to the CALLed address. Whenever you do a RETurn, the top address is taken off the stack and you go there. When there are a series of calls, you can have a stack of as many addresses as you need, up to the limits of memory.

The stack pointer holds the address of the top of the stack. When an address is placed on the stack, the stack pointer is decremented twice to point to the top of the stack again (remember, an address requires two bytes). This means that the stack grows from the far end of the memory towards the beginning. When addresses are taken off the stack they are not erased. The stack pointer is just incremented twice to point to the next one down the stack. The last address stacked is the first to be unstacked.

There are a lot of things that can go wrong with this. In fact, mixing up the stack is the commonest cause of crashes in machine code programming. The stack pointer merely points to the top of the stack and it has no idea which address goes with which RET instruction. You have to keep this in order yourself, by the way you write your program.

Obviously, you have to keep your calls and returns in pairs. If you have more calls than returns, then you won't get back to where you started from, and if you have more returns than calls, you will find yourself going off to somewhere new, an address you never stacked in the first place. But that is not the part that gives the most trouble.

CALL and RET use the stack automatically. Lots of beginners don't even realise that the stack is involved when they use these instructions. This causes trouble because you can also use the stack for temporary storage of the various register pairs. The PUSH and POP instructions allow you to place a register pair on the stack and retrieve it later, so that you can use the register for something else without losing the original contents.

The stack pointer still keeps track of the address of the last entry on the stack, but it doesn't have any idea whether it is an address from a CALL instruction or a stored register pair from a PUSH instruction. Obviously, PUSH and POP must also be kept in pairs.

Things get particularly difficult to follow when you PUSH a register and then CALL a routine that PUSHes two registers, does a calculation, POPs the two registers again (in reverse order, remember, last in, first out) and returns, (better POP that first register again before you forget).

The return address for the CALL instruction would be sandwiched between stored registers on the stack, and if anything gets out of order you will have stored register being used as a return address. Usually this will crash the program in a big way, but if you are very unlucky it may just give a weird malfunction that takes days to track down.

If you want to see the stack you can use the hex handler to step through the memory and examine it. You will find it starting at about 65300 in 48k Spectrums and 32500 in 16k models. You will notice that, like the system variables, the bytes in the "before" and "after" columns do not always agree. The stack is constantly changing whenever the computer is running. Don't enter any bytes with the hex handler or you will probably get a crash.

The stack pointer is initialised by the Spectrum on power-up or when you enter a CLEAR command. You can also use the CLEAR command with an address to move the stack to a new area of the RAM. This is normally done in order to set aside a chunk of RAM for machine code programs, so that it will not be written over inadvertently by either the stack or Basic program lines. This is best done with the CLEAR command in Basic, rather than by a machine code instruction within a code routine, or you will not be able to return to Basic since the return address will not be on the new stack.

The interrupt vector

The I register is a single byte register which is used in more complicated computers to communicate with interrupt-driven peripherals. In the Spectrum there is only one interrupt so no table of vectors is required, and the Spectrum uses a simpler interrupt mode that does not use the I register.

However, the I register is not wasted, as the Spectrum employs it in the generation of the TV signal. If you should load the I register with a number between &40 and &80, the signal will be disturbed, although higher or lower numbers don't seem to affect it. Try the following code:

```
32000  62    7D00  3E   3E     LD A,&50
32001  128   7D01  50   50
32002  237   7D02  ED   ED     LD I,A
32003  71    7D03  47   47
32004  201   7D04  C9   C9     RET
```

Run the code and you will see that the TV signal is messed up. Change the byte in location 32001 to something over &80 and run the code routine again to clear the interference from the screen.

Some programmers use a vectored interrupt in games software for continuous music or to provide "break-proof" protection, but the techniques required are too complex to go into at this point.

The refresh register

Another one best left alone, the R register is used by the computer when refreshing the dynamic memory chips. The number in this register keeps changing constantly, so it is no good trying to store something here for later retrieval!

Register loading and copying

The largest group of instructions are those that load numbers into registers or copy one register to another. The mnemonic for these almost always starts with LD, for load. LD B,E will load the B register with the contents of the E register, for instance. It is important to realise that E remains unchanged by this. The LD instructions only copy the contents of one register to another and they do not erase the register the contents were in originally.

There are LD instructions to load an eight-bit number into a single byte register or a 16-bit number into a double byte register. Almost any single byte register can be loaded into any other single byte register. There are very few instructions capable of loading a double byte register from another double byte register, so usually this has to be done a byte at a time with two separate instructions.

It is also possible to load a register with the contents of a memory location, either by specifying the address of the location in the instruction, or by having the address in another register pair. This can be done in reverse too, loading a memory location from a register. In general, any instruction which acts on a register has a counterpart which will act on a memory location, provided that the address of the memory location is in the HL pair.

Arithmetic and logic

The instructions for binary arithmetic allow you to add a number or contents of a register to the accumulator, or subtract it from the accumulator. For double byte arithmetic, BC, DE or SP (the stack pointer) may be added to or subtracted from the HL register pair. That means you must use the accumulator or HL pair when doing sums. It is not possible to add the B register to the C register, for instance. You must first load one of them into the accumulator.

There is one exception. You can add or subtract 1 to or from any register or register pair with the INC (for increment) and DEC (for decrement) instructions.

The mnemonics for addition and subtraction are ADD, ADC, SUB and SBC. In these mnemonics the C indicates that the state of the carry flag is to be added or subtracted from the result of the sum. This allows you to do multiple byte arithmetic taking a carry from earlier results. All the addition and subtraction instructions (except INC and DEC) set or reset the

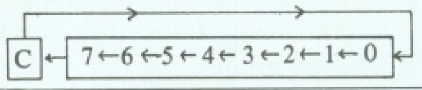
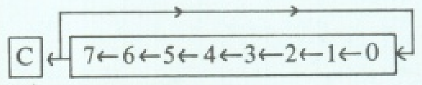
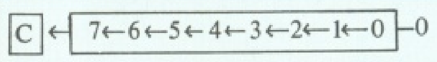
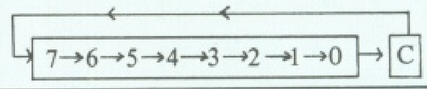
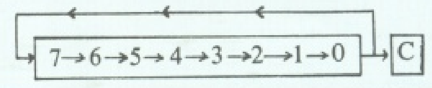
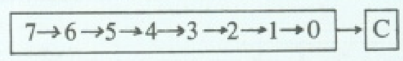
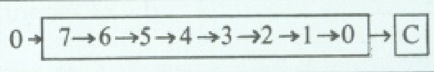
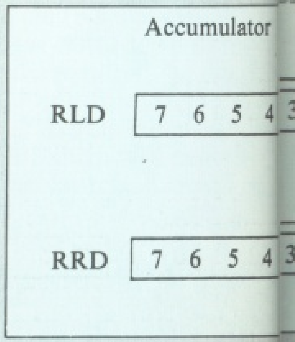
Mnemonic	Action	Diagram
RL or RLA	Rotate left through carry Bit 7 to carry, carry to bit 0	
RLC or RLCA	Rotate left and into carry Bit 7 to carry and bit 0	
SLA	Shift left Bit 7 to carry, bit 0 is reset	
RR and RRA	Rotate right through carry Bit 0 to carry, carry to bit 7	
RRC or RRCA	Rotate right and into carry Bit 0 to carry and bit 7	
SRA	Shift right Bit 0 to carry, bit 7 unchanged	
SRL	Shift right logical Bit 0 to carry, bit 7 is reset	

Figure I: Rotates and shifts



Invisible registers

There are a number of other registers within the chip that are not meant to be accessible to the programmer, but are reserved for the use of the microprocessor itself. One of these is the program counter which the computer uses to keep track of where it is in a program.

When it executes a CALL instruction, it copies the current contents of the program counter to the top of the stack and then puts the call address into the program counter, which causes it to fetch its next instruction from the new address.

When it finds a RETurn instruction, it copies the address on the top of the stack into the program counter. You can see that the instructions you put in a program affect the program counter, even though there are no mnemonics that mention it by name.

carry flag to show if there has been a carry from their operation, but only ADC and SBC take account of the state of the carry in their own sum.

Other flags affected by arithmetical or logical operations are the zero flag which is set when the result of an operation is zero, the sign flag which is used in two's complement arithmetic and the parity/overflow flag which can be used with two's complement arithmetic or communications in/out routines. These flags are bits of the flag register and they can be used for decision making by the use of conditional jumps and calls.

Not all operations affect all the flags. INC and DEC, for instance do not affect the carry flag, but do affect the zero flag when used on single – but not double – registers. A particular point to remember is that merely loading a register does not affect any flags.

Binary logic

The logic instructions allow you to AND, OR or XOR (eXclusive OR) a number or register with the accumulator. There are no double byte logical operations. These instructions perform a bit by bit logical operation on the two eight bit operands, one of which is in the A register, and return the result in the A register.

AND sets a bit in the result if the corresponding bits in both operands are set. OR sets the result bit if either or both operand bits are set. XOR sets the result bit if either but not both of the operand bits are set. The flags

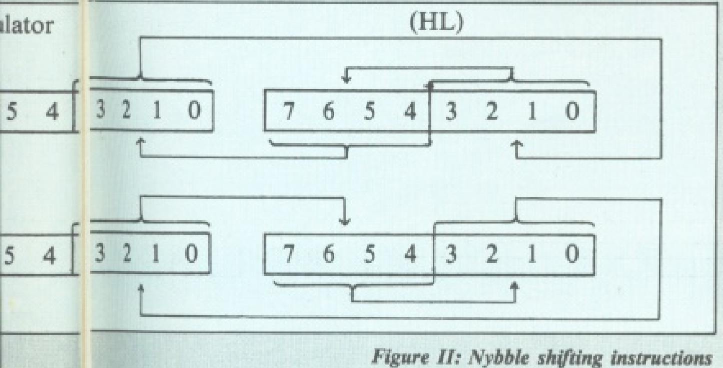


Figure II: Nybble shifting instructions

The instructions

To cover the instruction set in the kind of nit-picking detail you really need if you are to program in machine code requires a rather large reference manual. Such books exist, and if you plan to get deeply embroiled with machine code, I recommend that you get Rodney Zaks' Programming The Z80, published by Sybex ISBN 0-89588-069-5.

Other books on machine code and the Spectrum have more or less complete lists of the instruction set mnemonics and op-codes, and the original Spectrum manual lists the instruction set in Appendix A with the character set. Remember that the mnemonics are just for the convenience of humans – the machine only understands the op-codes, which are usually given in hexadecimal.

The easiest way to write in machine code (assuming you don't have an assembler program) is to write the program in mnemonics and then translate it into op-codes via one of these lists.

are affected as with the add and subtract instructions.

The compare instruction (CP) sets the flags as if it were a subtract instruction, but without actually doing the subtraction. This allows you to test for a comparison – via a conditional jump – without changing the contents of the A register.

The CPL (complement) instruction inverts the bits of the A register, making zeros into ones and vice versa. The NEG instruction is similar, but it also increments the A register. These operations are mostly used in two's complement arithmetic.

Another instruction, DAA (decimal adjust accumulator) is used in a different type of arithmetic called "binary coded decimal". These are the two main ways of programming a binary computer to do floating point arithmetic. The Spectrum ROM uses the two's complement method.

The BIT, SET and RES instructions allow you to test, set, or reset any individual bit in any one byte register. SET and RES do not affect any flags, but the result of a BIT instruction is to set the zero flag if the bit is zero, or vice versa.

There are a number of instructions for rotating registers, which can be used to shift bits along one place in either direction. This may seem an odd thing to want to do, but it is very useful in binary arithmetic, because such a shift amounts to multiplication or division by two. For example:

%01001000 = 72
%00100100 = 36
%00010010 = 18

Depending on which type of rotation you use, the bits that fall off of one end of the byte can be copied to the other end, or to the carry flag, or both. Once again the carry flag is used to transfer a carry so that you can do arithmetic on numbers larger than 255.

In the Spectrum we already have a very powerful calculator routine in the ROM which we can call if we want any tedious number crunching done, but there are many occasions when you will want to do a simple bit of binary addition or manipulation in your own routines without bothering with the calculator.

● ***In part five we'll examine the other Z80 instructions.***

Analysis

A palindrome is a word that's the same spelt backwards or forwards. Rotor is one, and so is Eve.

And while we're still around the garden of Eden, "Madam I'm Adam" is a palindrome as well – provided you ignore the capitals and the apostrophe.

This brings us to the second Analysis listing. This is called Palindrome Tester, and it gets your Spectrum to test whether or not a word is a palindrome.

Not only does it give you a decision on this vital matter, but it also illustrates some of the clever techniques used in string handling on the Spectrum.

Used to label the program. The Spectrum ignores everything after the REM.

Just sets up a string variable *d$*, making it hold the null or empty set.

The INPUT doesn't just take in a string and store it in *w$*, it also prompts you with details of what is expected. The LINE before the *w$* is there to remove the inverted commas which appear at the bottom of the screen. Leave it out and see what happens.

Checks to see if the program should end. Notice that End or eNd won't be caught by this line. It only traps the two obvious cases.

Takes one character from *w$* and puts it on the left of whatever is already in *d$*. As the loop cycles and *i* goes from 1 to the length of *w$*, so each letter in *w$* will be added to the front of *d$*. The result is that *d$* is the reverse of *w$*. A point to note is that *w$(i)* is not an array – it's short for *w$(i TO i)*.

Sends the program back to the beginning again for another go. There's no condition to fulfil – when the Spectrum reaches this line the program goes straight back to 30. It's what's known as an unconditional jump. The number 30 could have been used instead of *beginning* but wouldn't have conveyed the essence of the line as well as GOTO *beginning*.

```
10  REM Palindrome Tester
20  REM An exercise in string
    handling

30  LET palindrome=0: LET beg
    inning=30

40  LET d$=""

50  INPUT "Enter a word, 2-10
    letters, 'end' to stop", LINE
    w$

60  IF w$="end" OR w$="EN
    D" THEN STOP

70  LET length=LEN w$

80  IF length<2 OR length>10
    THEN PRINT "Invalid word":
    GO TO 50

90  FOR i=1 TO length
100 LET d$=w$(i)+d$
110 NEXT i

120 IF d$=w$ THEN LET palind
    rome=1

130 IF palindrome THEN PRINT
    w$;" is a palindrome"

140 IF NOT palindrome THEN P
    RINT w$;" is not a palindro
    me"

150 GO TO beginning
```

Sets up two variables, *palindrome* and *beginning*. Notice how they are treated in two different ways later in the program. *beginning* is used to hold a line number, and always remains at 30. Unsurprisingly it's known as a constant. *palindrome* is used as a flag to decide which of two messages is printed. The way the program is written it can only take two values 0 and 1. It's an example of a logical or Boolean variable.

Finds the length of the input word using LEN *w$* and stores it in *length*.

A mugtrap. If you have typed in a word that's too long or too short then you're told so and the program loops back to the input line. The program won't go past this line unless *w$* meets the conditions.

Make up a FOR . . . NEXT loop which cycles once for each character in *w$*. The task of the loop is to produce a reversed copy of *w$* stored in *d$*.

Checks to see if the reverse of *w$* is the same as *w$*. If it is, then the word is a palindrome and *palindrome* is set to 1 or true.

Uses *palindrome* to decide whether to print the message following the THEN. If *palindrome* is 1 then the message is printed.

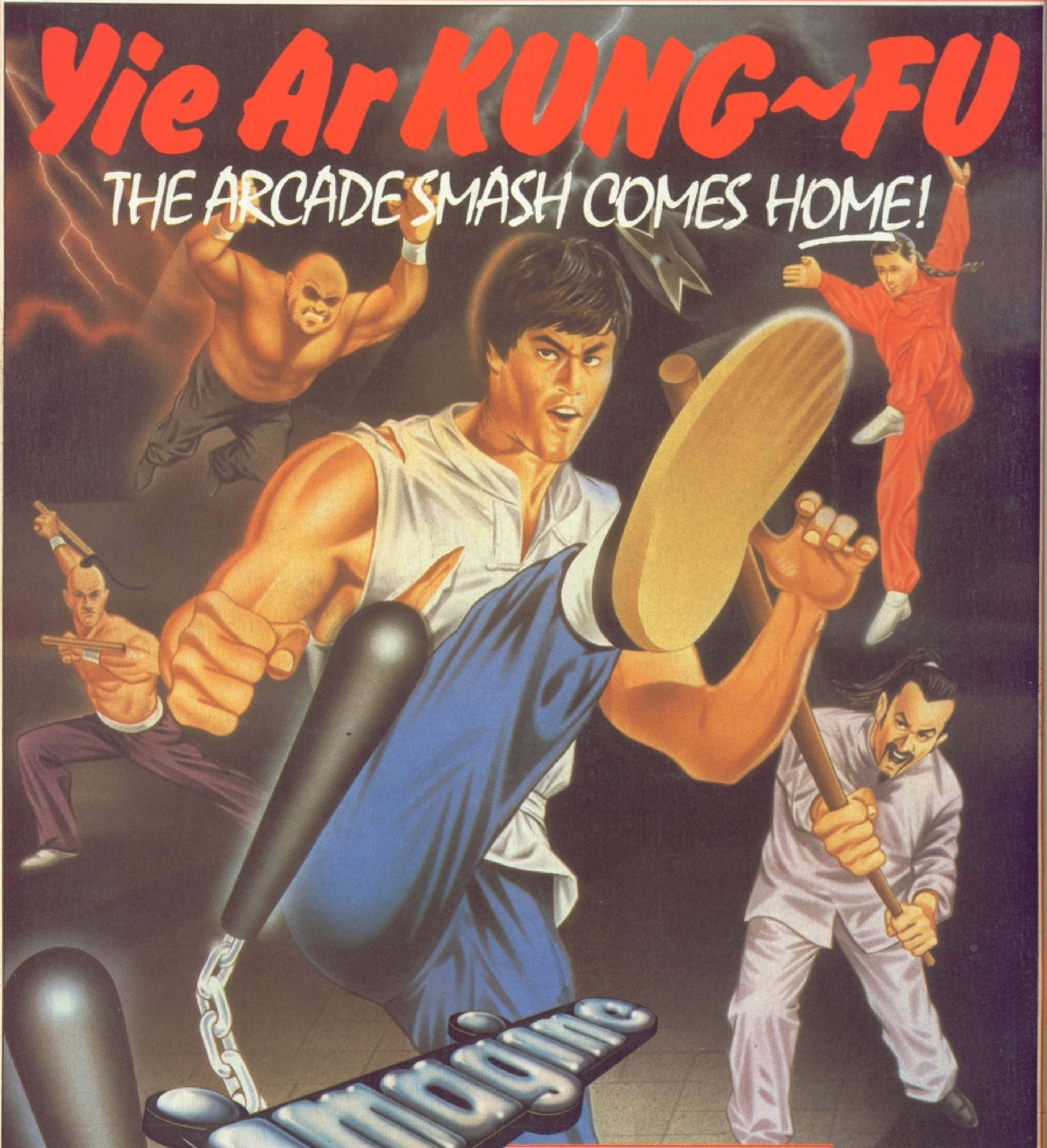
Displays the message only if *palindrome* is 0 or false.

When you've figured out how the program works, don't just sit there congratulating yourself – try to improve it.

As it stands, it won't accept Eve as a palindrome, it has to be EVE or eve. Can you get the program to ignore the difference between upper and lower case when it's testing? And what about the spaces and punctuation in "Madam I'm Adam"?

Also, couldn't the program be made more efficient? Is there a better way of testing for a palindrome? As a hint, what do the first and last letters of a palindrome have in common?

AMSTRAD £8·95
ZX SPECTRUM £7·95
COMMODORE 64 £8·95



Imagine Software is available from: Boots, WHSMITH, John Menzies, WOOLWORTH, LASKYS, Rumbelows, Greens, Spectrum Shops and all good dealers.

Imagine Software (1984) Limited • 6 Central Street • Manchester M2 5NS • Tel: 061 834 3939 • Telex: 669977