sinclair

ZX Spectrum+

User Guide

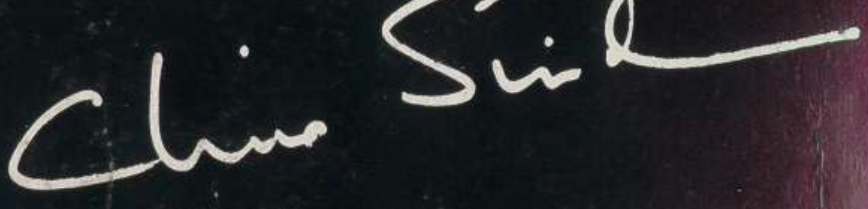
SPECTRUM SOFTWARE

The entire range of software available for Spectrum computers (including all existing titles) is completely compatible with your new ZX Spectrum+.

INTRODUCING THE ZX SPECTRUM+

Sinclair Research has long led the advance in microchip technology that has brought computing to all. Following the advent of the world's first low-cost microcomputer, the ZX80, we have combined ever greater computing power with ever greater value in its successors – the ZX81, ZX Spectrum and QL computers. Ease of use has also been our watchword, both in the design of these products and in the ways in which they work.

The ZX Spectrum + takes Sinclair Research one further step along this road. In it, you have a machine with all the best features of the Spectrum in an upgraded version that makes this most powerful and popular of all microcomputers even simpler to use. It is our hope that you will take full advantage of the many possibilities that your new computer will provide.



CONTENTS

GET GOING 3

START PROGRAMMING 17

LEARN ABOUT YOUR ZX SPECTRUM+ 41

LEARN ABOUT SINCLAIR BASIC 49

Written by Neil Ardley
Published by Dorling Kindersley Ltd
in association with
Sinclair Research Ltd

HOW TO USE THIS BOOK

This guide to your ZX Spectrum + contains four colour-coded chapters. To turn to a chapter, simply open the book at the section with the right colour.

1 GET GOING

Setting up your ZX Spectrum + ■ Tuning in your TV ■ Setting up troubleshooter ■ What your ZX Spectrum + can do ■ How to use ready-to-run software ■ How to load a program ■ Software loading troubleshooter

2 START PROGRAMMING

The keyboard – your computer's control panel ■ How to operate the keys ■ The television calculator ■ Colour and how to use it ■ Simple DIY graphics ■ The on-screen sketchpad ■ Design your own patterns and pictures ■ How to create your own computer characters ■ Animation ■ How to make music and sound effects ■ How to save your own programs ■ Software saving troubleshooter

3 LEARN ABOUT YOUR ZX SPECTRUM+

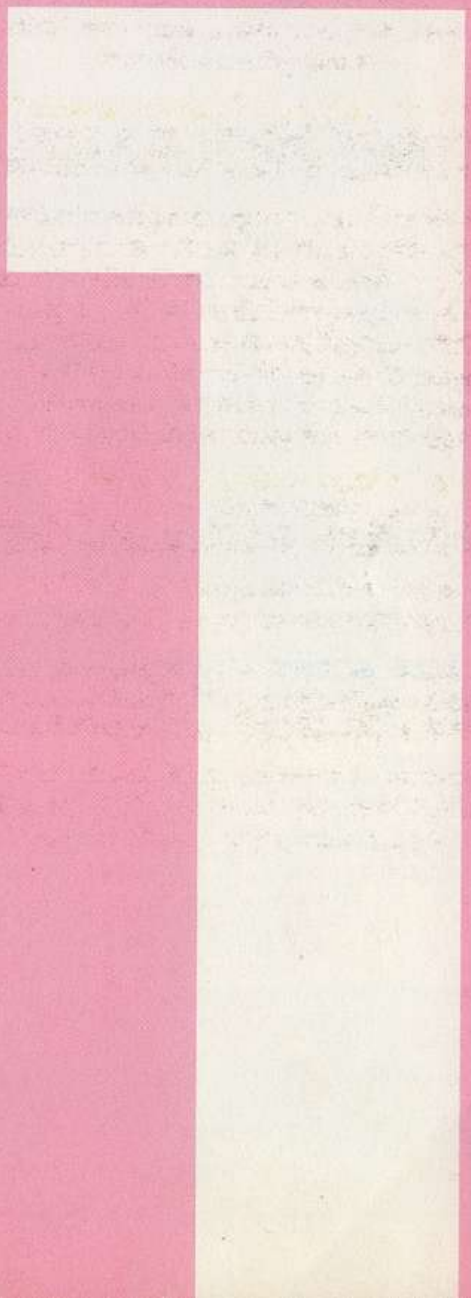
What's inside? ■ How does your ZX Spectrum + work? ■ How to connect peripherals ■ ZX Spectrum + memory map

4 LEARN ABOUT SINCLAIR BASIC

Programmer's reference guide to Sinclair BASIC keywords ■ Spectrum screen reports ■ Beyond BASIC ■ Computer jargon – what it means

GET GOING

This chapter shows you how to start exploring the potential of your ZX Spectrum +. You'll find out here how to set up the computer so that it is ready to spring into action whenever you wish. Then you have a choice. You can key in several programs that put the Spectrum through its paces and show off its colour graphics and sound, or you can find out how to use ready-to-run software, such as computer games. Either way, you'll soon be enjoying operating your new computer.



SETTING UP YOUR ZX SPECTRUM+

To get your Spectrum ready for action, first go through the checklist below to make sure that you have all the parts you need, then follow the instructions on the opposite page for connecting and powering up. You do not need to use a cassette player at this stage.

Connect everything firmly. If you accidentally disconnect or switch off the power supply when the Spectrum is in use, you will then lose your program and all your information or results.

When you have finished with the computer, switch off at the power point on the wall if it has a switch and pull the mains plug out of the power point.

Checklist: Have you got everything you need?

On unpacking, you will find:

1 Your ZX Spectrum+.

2 The ZX Power Supply – this produces the 9-volt DC supply required by the Spectrum.

3 The aerial lead - this connects your Spectrum to a television set.

4 The cassette lead - this connects your Spectrum to a cassette player.

5 A guarantee card - which you should complete and return.

6 The User Guide Companion Cassette.

7 This manual.

You will need to supply:

1 A television set.

2 A cassette player.

3 A mains plug.

Setting up questions and answers

Must I have a colour TV?
No. However, you will not be able to see the colours produced by the Spectrum on a black-and-white set.

Will any TV do?
Your Spectrum should give a picture with any television set that you own. If it does not, the reason may be that the television set and computer have different picture systems. This could happen if the television set is very old, or if the television set and your Spectrum were purchased in different countries. If in doubt, consult your television dealer.

Can I use a monitor instead of a TV set?
Yes. Your dealer may be able to supply monitors giving a superior picture for the Spectrum.

What mains supply is required?
The Spectrum requires a mains current of 1.4A at 240V/50Hz, the standard UK supply.

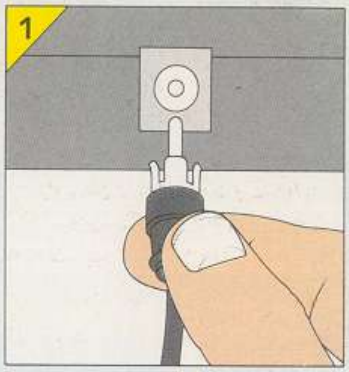
Does the Spectrum produce interference?
The Spectrum may interfere with a radio that is near the computer. This will not harm the radio or the computer.

Can I use the ZX 16K RAM?
No. This RAM pack is only to be used with the Sinclair ZX81 computer.

Powering up your ZX Spectrum +

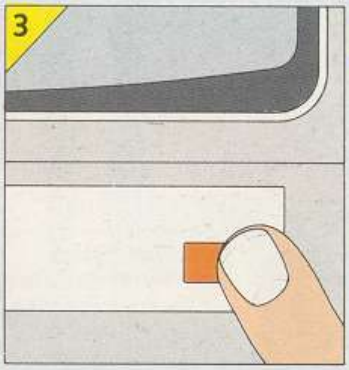
Begin by fitting a mains plug to the two bared ends of the wire from the power supply. You will then need to fit a 3A fuse to the plug. Note that your Spectrum does not require an earth connection even if the mains plug that you fit has three pins.

Next follow the sequence of illustrations here to connect your Spectrum to the mains electricity supply and your television set. Once you have made all the connections and powered up the system, turn over the page to find out about tuning in.

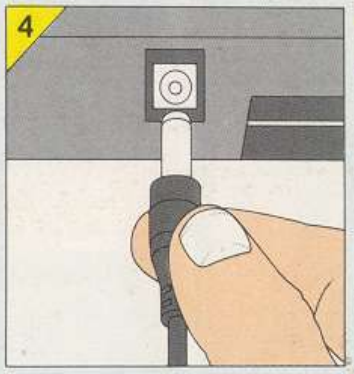


Insert the aerial lead into the socket marked TV on your Spectrum. Only one of the plugs on the aerial lead will fit.

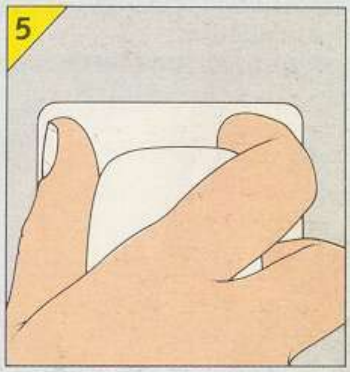
Detach the aerial cable from your television set. Insert the other plug on the Spectrum's aerial lead into the aerial socket.



Switch on the television set and turn the volume control right down.



Insert the small plug on the power supply wire into the socket marked 9VDC on your Spectrum.



Push the mains plug into a power point and switch on if it has a switch. Your Spectrum does not have its own switch.

The Spectrum's sockets and connectors

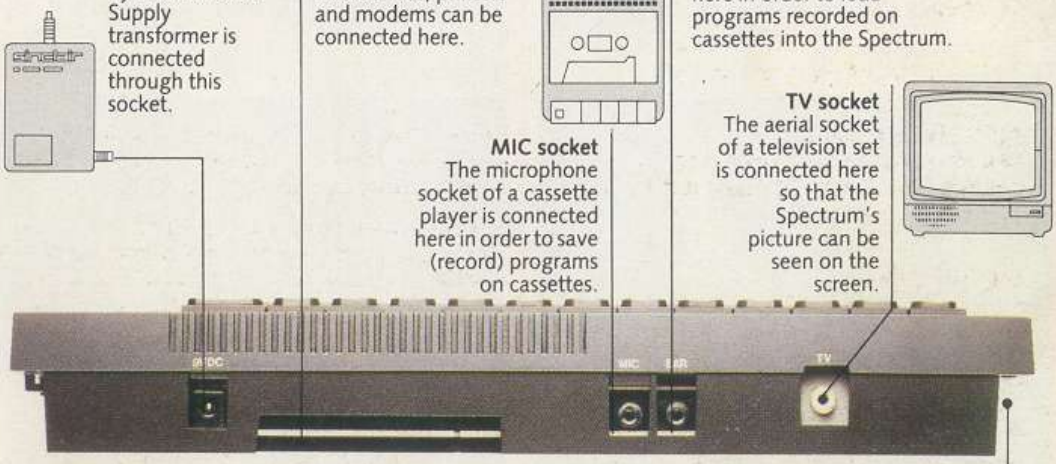
Power socket
The 9-volt DC supply produced by the ZX Power Supply transformer is connected through this socket.

Edge connector
A wide range of hardware, including Microdrives, printers and modems can be connected here.

EAR socket
The earphone socket of a cassette player is connected here in order to load programs recorded on cassettes into the Spectrum.

MIC socket
The microphone socket of a cassette player is connected here in order to save (record) programs on cassettes.

TV socket
The aerial socket of a television set is connected here so that the Spectrum's picture can be seen on the screen.



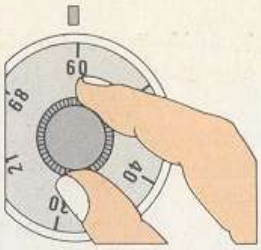
Reset button

TUNING IN YOUR TV

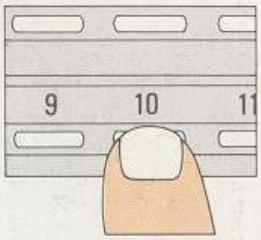
Your Spectrum puts out a colour television video signal at the frequency of channel 36 in the UHF band, so your television set must be tuned to this channel to display the computer's picture.

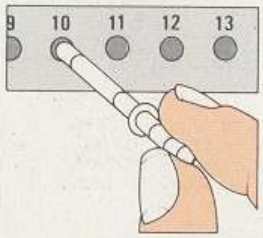
When you have powered up your Spectrum and plugged it into the television set, adjust the tuning control on the set until you get the Sinclair copyright message as in the first screen below. When you can see this, you will be ready to test the Spectrum's colours and then begin computing. If you can't get the copyright message, or if the colours don't look right, check through the chart opposite.

Tuning controls

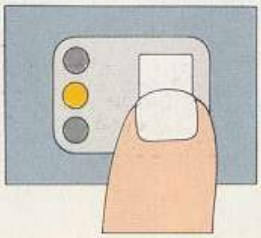


Variable tuning
A variable tuning control selects any channel. Just turn the knob until you get the copyright message.





Push-button tuning
Select a tuning button which is to be used for computing and then adjust it until the copyright message appears. If possible, use a spare button. Then you will not have to tune the set every time you want to put your Spectrum to work.

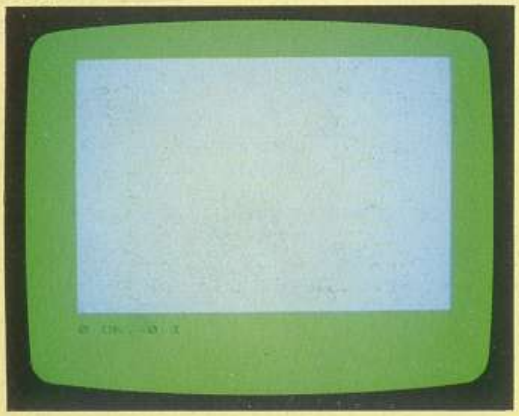


Electronic tuning
With this system, the set itself tunes in the required channel. TV sets with synthesized channel selection but no manual tuning option may not be suitable for use with this computer.

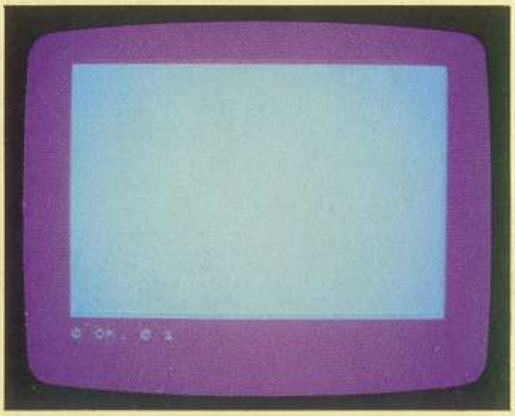
How to test the Spectrum's colours

To test the Spectrum's colours, simply press the B key and then a number from 1 to 6. The copyright message disappears; first the word BORDER appears and then the number. Now press the key marked ENTER. The 'border' area of the screen should change to the colour marked on the number key. The screens below show what happens when you key in BORDER 4 and ENTER and then BORDER 3 and ENTER. BORDER 7 will return the border to white.

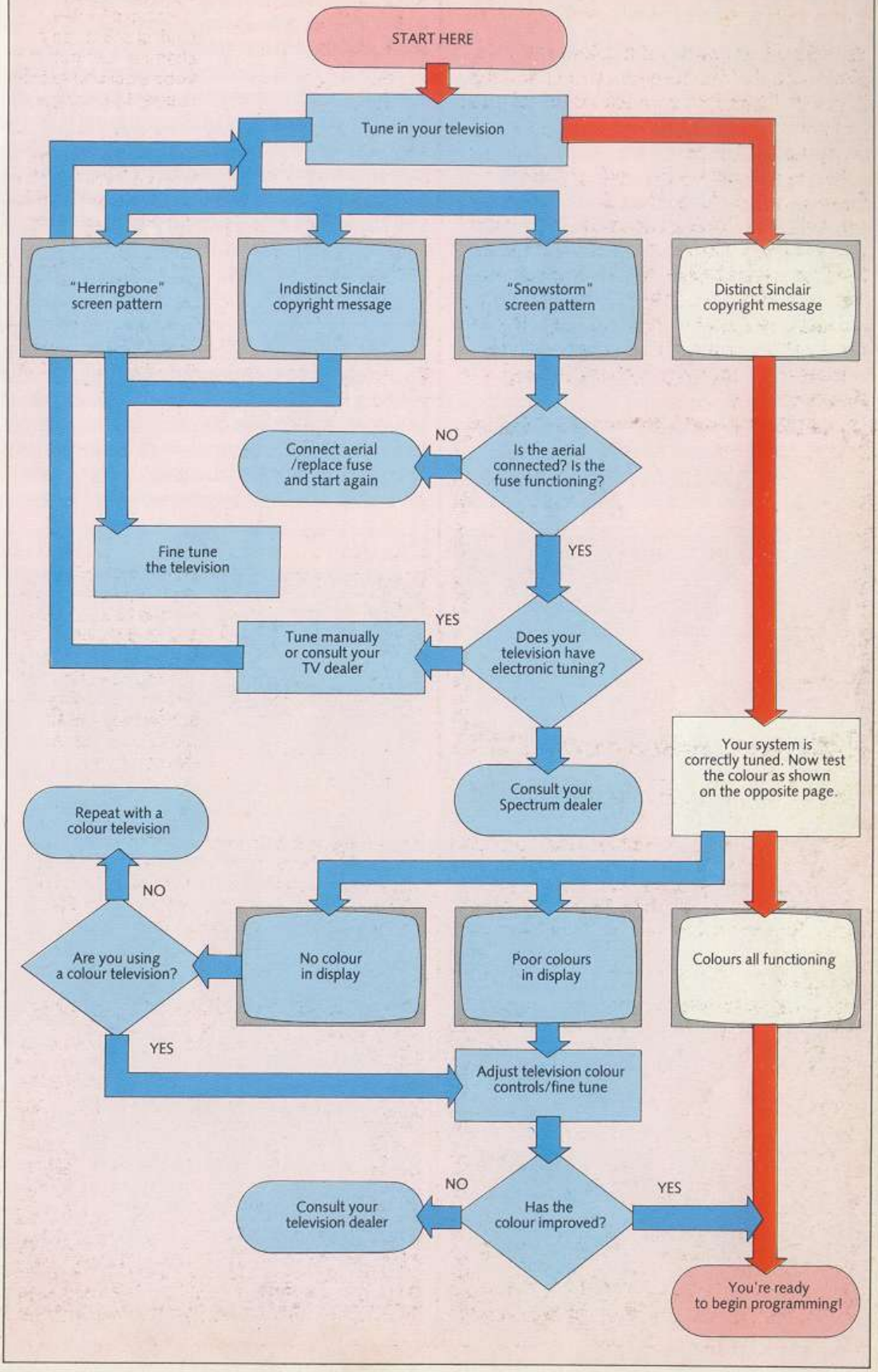
BORDER 4



BORDER 3



Setting up troubleshooter



WHAT YOUR ZX SPECTRUM+ CAN DO

First, experiment

Now that your Spectrum is powered up and your television set is tuned in, try pressing a few keys. You'll see words and letters appearing on the screen, and maybe some numbers too.

However, unless you know how to program the Spectrum, it's unlikely that the computer will respond by doing anything. But don't worry – nothing can go wrong with it, no matter which keys you press.

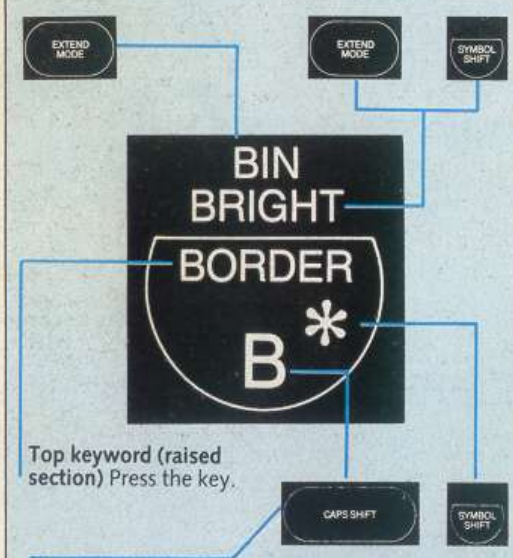
Now press the reset button on the left-hand side of the computer, and you're ready to set your Spectrum to work. The next four pages will give you some demonstrations on the TV screen of what the Spectrum can do.

How to key in

To key in any word or letter, first note its position on the key. Then use the same sequence of selector keys shown here.

Top keyword Press EXTEND MODE then the key.

Lower keyword or sign Press EXTEND MODE then *hold down* SYMBOL SHIFT and press the key.



Top keyword (raised section) Press the key.

Letter or number (raised section) Press the key. Hold down CAPS SHIFT for capitals.

Lower keyword or sign (raised section) Hold down SYMBOL SHIFT and press the key.

You can find out full details on how to operate the keys on pages 20–21.

Next, program your Spectrum

Your Spectrum can do a lot of things. But to make it work, you have to give it a set of instructions called a computer program.

Here is a collection of short programs that will put your Spectrum through its paces. All you have to do is key in the program instructions *exactly* as they appear here. The screen pictures show you what to expect. The panel marked **How to alter a program** will show you how to experiment with the programs yourself. To begin a *new* program, see the panel on page 11.

How to enter and run a program

Each set of instructions is shown in a list called a listing. You'll see that the program listings contain several sections each beginning with a number – 10, 20 and so on. Each section is called a *line* in the program (even if it takes up two lines on the screen),

NAMES

```
10 BORDER 1: INK RND*7
20 PAPER RND*7
30 PRINT " ZX Spectrum +";
40 GO TO 10
```

The name ZX Spectrum + appears in many colours all over the screen. The computer then stops and a message, **scroll?**, appears at the bottom of the screen. To make the display "scroll" up, press any key except N, SPACE, BREAK or STOP. If you do stop the scrolling, and press BREAK, then R (RUN) followed by ENTER, the names will appear in a different pattern of colours.

Try this

In line 30, change "ZX Spectrum +" to your name in quote marks (") – for example

```
30 PRINT "John ";
```

Remember to include the semicolon (;). You'll then see your name appear all over the screen.

and it contains one or more instructions for the computer.

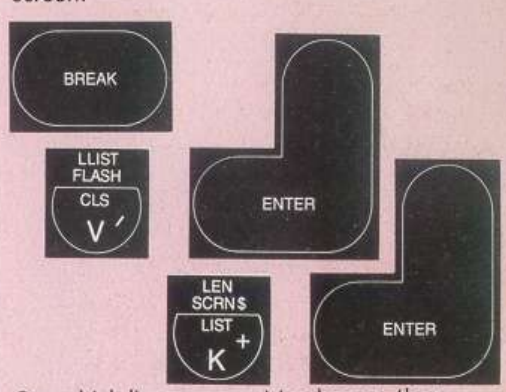
In each program line, you'll see whole words or abbreviations containing two or more letters, such as **PRINT, LET, RND, PI, PAPER** and **GOTO.** These are called *keywords* and you cannot key them in letter by letter. Instead, find the key on the keyboard bearing the keyword (**PRINT** is on the P key, for example), and then follow the instructions in the **How to key in** panel.

As you key in a line, it appears at the bottom of the screen. When you get to the end of the program line, press the ENTER key. The line now appears at the top of the screen. Then key in and enter each line in the same way. If you press a wrong key by accident, turn to the panel marked **How to correct mistakes** on the next page.

When you have entered all the lines, press R. The keyword **RUN** appears. Now press ENTER and your Spectrum will spring into action as it runs the program.

How to alter a program

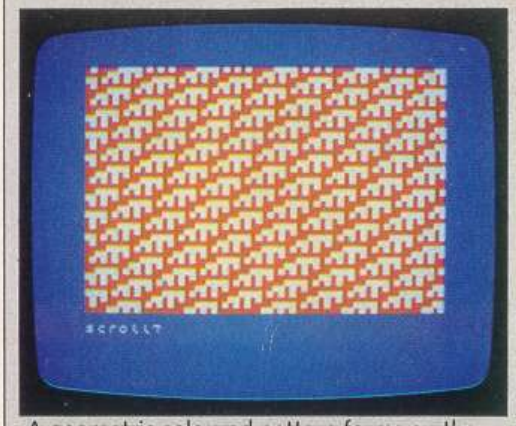
Wait until the program ends or stop it by pressing BREAK. Then press V (CLS) then ENTER followed by K (LIST) then ENTER. The program listing (list of lines) appears on the screen.



See which line you want to change, then key in the whole new line, including the line number, and press ENTER. The new line will replace the old one. Press R (RUN) and ENTER and the new program will start.

PATTERNS

```
10 LET a$=""
20 FOR x=1 TO 7
30 LET a$=a$+CHR$ (RND*14+129)
40 NEXT x
50 INK RND*7
60 BORDER RND*7
70 PRINT a$;
80 GO TO 70
```

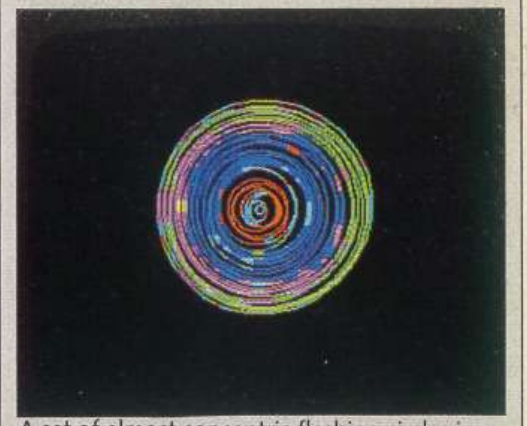


A geometric coloured pattern forms on the screen when you run the program. When the screen is full, the display stops with the scroll? message. To see more of the same display, press any key (except N, SPACE, BREAK or STOP) to make the pattern move up. To see a new kind of pattern in a different combination of colours, press N when the scroll? message appears. Then press BREAK, followed by R (RUN) then ENTER.

Try this
In line 20, change 7 to another number to get a different kind of pattern. Try 8, for example.

FLASHING CIRCLES

```
 10 BORDER 0: PAPER 0: CLS
 20 CIRCLE INK RND*6; FLASH RND
;120+RND*8,80+RND*6,RND*80
 30 BEEP 0.1,RND*60
 40 IF RND>.9 THEN GO TO 60
 50 GO TO 20
 60 FOR y=-2 TO 4
 70 FOR x=0 TO 6
 80 BORDER x
 90 BEEP .05,x*y
100 NEXT x
110 NEXT y
120 RUN
```

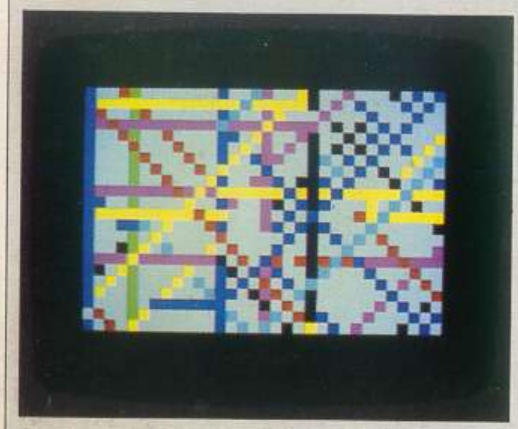


A set of almost concentric flashing circles in a range of different colours builds up on the screen. Then suddenly the border flashes, the computer produces a trilling sound, and a new set of circles appears.

Try this
Before listing the program (using the K key), key in PAPER 7 and press ENTER. Then key in line 20 again, missing out the two keywords **FLASH RND;** and the circles will no longer flash.

MANIC MOSAIC

```
  5 BORDER 0: CLS
 10 LET h=16: LET v=11
 20 LET x=INT (RND*3-1): LET y=
INT (RND*3-1)
 30 INK RND*7
 40 FOR z=1 TO 20
 50 PRINT AT v,h;CHR$ 143
 60 LET h=h+x
 70 LET v=v+y
 80 IF h<0 THEN LET h=31
 90 IF h>31 THEN LET h=0
100 IF v<0 THEN LET v=21
110 IF v>21 THEN LET v=0
120 NEXT z
130 GO TO 20
```



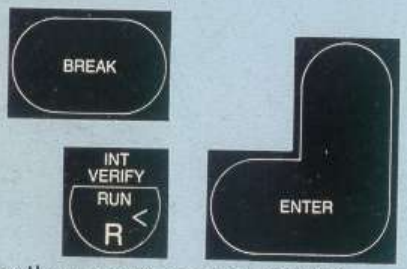
A coloured square dashes to and fro all over the screen, building up a coloured pattern. A different pattern is produced every time you restart the program.

Try this
In line 50, change 143 to 42 and you'll see stars! Try other numbers from 33 to 142. Consult the character set chart on page 51 to check what will happen.

How to restart a program

Some of these programs – like STARS AND STRIPES – come to an end and produce the report **0 OK**, and the last line number in the program. This means that the whole program has finished. To start again, simply press R (RUN) and ENTER.

Other programs either keep on running, like MANIC MOSAIC, or start again automatically, like SHIMMERING SUNRISE.



To stop these programs, press BREAK.

Hold this key down until the program stops and the BREAK report appears. To restart, just press R (RUN) and ENTER.

How to correct mistakes

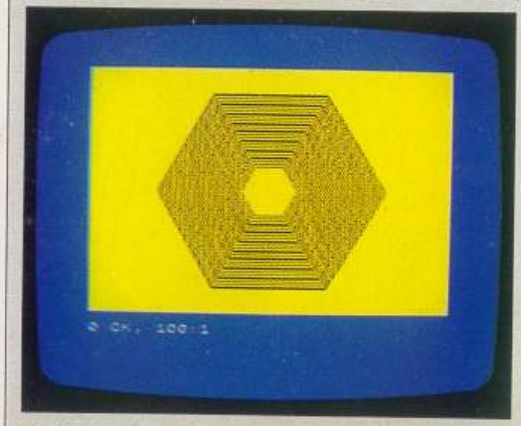
If you press a wrong key or do not press the shift or EXTEND MODE keys properly, don't worry. Press the DELETE key, and the last keyword, sign, letter or number will disappear. Hold down DELETE to delete more.

If you have made a mistake in a line and then you press ENTER, a flashing question mark may appear. This precedes the mistake. Press DELETE to delete the line up to and including the mistake, then complete the line correctly and press ENTER.

If you manage to enter an incorrect line, the program may stop working, producing a report that gives the number of the wrong line at the bottom of the screen. Key in the whole of this line correctly, then press ENTER, R (RUN) and ENTER. Now the program should work.

POLYHEDRA

```
  5 BORDER 1: PAPER 6: CLS
 10 INPUT n
 20 FOR r=20 TO 80 STEP 2
 25 LET x=128: LET y=87
 30 LET h1=x-r: LET v1=y
 35 PLOT h1,v1
 40 FOR a=0 TO 361 STEP 360/n
 50 LET h2=x-r*COS (a*PI/180)
 60 LET v2=y+r*SIN (a*PI/180)
 70 DRAW h2-h1,v2-v1
 80 LET h1=h2: LET v1=v2
 85 BEEP 0.02,r-20
 90 NEXT a
100 NEXT r
```

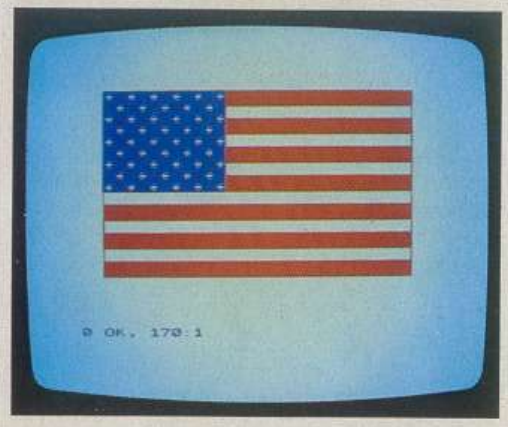


At first, you see a blank screen. Key in 6, then press ENTER. A six-sided shape builds up. Restart the program when it has finished and key in another number to get a shape with a different number of sides.

Try this
In line 20, change 2 to another number. The pattern builds more quickly if the number is bigger, and the polyhedra (many-sided figures) are farther apart.

STARS AND STRIPES

```
 10 INK 2
 20 PAPER 7
 30 CLS
 40 FOR x=28 TO 148 STEP 20
 50 FOR z=0 TO 11
 60 PLOT 16,z+x: DRAW 216,0
 70 NEXT z
 80 NEXT x
 90 PLOT 16,28: DRAW 0,131
100 PLOT 232,28: DRAW 0,131
110 PAPER 1
120 INK 7
130 FOR x=2 TO 8 STEP 2
140 PRINT AT x,2;"* * * * * *"
150 PRINT AT x+1,2;" * * * * * *
"
160 NEXT x
170 PRINT AT x,2;"* * * * * * *"
```



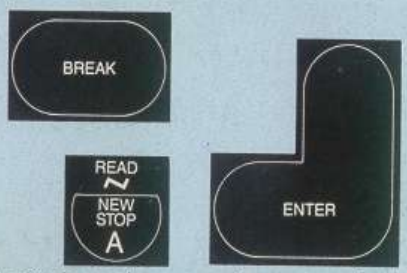
The United States flag appears on the screen.

Try this
Change the colour numbers of the flag. The stripe colour is at line 10, the stars at line 120 and the background to the stars at line 110.

How to begin a new program

When you have finished with a program and want to enter a totally new program, wait until it ends or stop it by pressing BREAK.

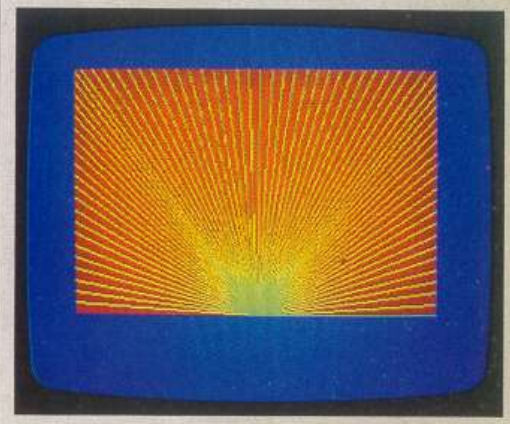
You then have a choice of two ways for erasing the old program from the computer's memory. One way is to press two keys, A (NEW) and then ENTER. The screen will go black for a moment and then the copyright message will appear.



Alternatively, and more easily, you can just press the reset button. This has the same effect as turning the Spectrum off and on at the power socket.

SHIMMERING SUNRISE

```
 10 BORDER RND*6
 20 INK RND*7
 30 PAPER RND*6
 40 CLS
 50 LET z=RND*10+2
 60 FOR x=0 TO 174 STEP z
 70 PLOT 128,0
 80 DRAW -128,x
 90 BEEP .01,x/3
100 NEXT x
110 FOR x=-127 TO 127 STEP z
120 PLOT 128,0
130 DRAW x,175
140 BEEP .01,60
150 NEXT x
160 FOR x=174 TO 0 STEP -z
170 PLOT 128,0
180 DRAW 127,x
190 BEEP .01,x/3
200 NEXT x
210 PAUSE 200
220 GO TO 10
```



A picture like a shimmering sunrise builds up in different colours every few seconds. If the screen goes blank, just wait. A new sunrise soon dawns.

Try this
In line 210, change 200 to another number in order to alter the time for which each sunrise stays on the screen. 200 is equal to 4 seconds.

What next?

Now you have a choice. If you want to keep any of these programs to run them again later, you can record them on cassette tapes. Turn to **How to save your own programs** on page 38 to find out how to do this.

If you want to carry on experimenting with your Spectrum, you can find out about programming by turning to Chapter 2 **Start programming**. So far you have just tried programs out, without necessarily understanding how they work. Chapter 2 will explain some of the features of Spectrum programming.

If you want to try out some software tapes, such as a computer game that you've bought, then turn the page to **How to use ready-to-run software**.

HOW TO USE READY-TO-RUN SOFTWARE

When you enter a program into the Spectrum, you produce a sequence of electronically coded signals as you press the keys. The codes go to the Spectrum's memory, which stores them so that the computer can use them when the program runs. The codes stay in the memory until you either remove them (by entering NEW or pressing the reset button, for example) or switch off your Spectrum.

However, it's not always necessary to key in a program when you want to use your Spectrum. Instead you can buy ready-to-run software, which contains programs that can be fed into the computer directly and automatically. Using ready-to-run software not only saves you the trouble of keying in a program every time you want to put your Spectrum to work, but it also enables you to have a library of programs ready for use that would take days or even weeks to write yourself. Software manufacturers produce programs of all kinds written by the best programmers, and a wide range is available for the Spectrum. Look at the Sinclair Spectrum Software Catalogue to get an idea of the kind of programs you can enjoy and use. Then whenever you want, you can run a particular program to suit your needs.

How programs are loaded into the Spectrum

The code signals on a software tape consist of high and low bleeps recorded at the rate of about 1500 every second. When you play back a software tape in a cassette player, the player produces the sequence of bleeps that make up the program. You just connect the cassette player to your Spectrum, and the codes go directly into the Spectrum's memory. This is called *loading* a program.

On these two pages, you can see how to connect up your cassette player. Pages 14–15 will then show you how to use it.

Software questions and answers

What does 'software' mean?
Software is the general name given to programs that are fed into computers to make them work. Hardware is the term for the actual machinery - the computers themselves and any other devices involved in computing.

Why is software produced on cassette tapes?
Cassette tapes are easy to use and do not require special equipment. An inexpensive cassette player is all that you need to load this kind of software.

What do taped programs sound like?
Play one on your cassette player without connecting it to the Spectrum. You'll hear a high-pitched screech. This is caused by the code signals going to the loudspeaker in the player instead of the computer. The signals are sent from the cassette to the Spectrum at such a high speed that it is impossible to distinguish the individual sounds.

Are there any other kinds of software?
Yes. You can get programs on ROM cartridges instead of cassette tapes. The cartridges plug into an interface which fits into the back of your Spectrum. A program on a ROM cartridge loads instantly without any waiting at all.

Software is also available on Microdrive cartridges. These contain programs recorded in magnetic form like a cassette tape. Several programs may be present on a cartridge and, unlike a cassette tape, any program can be loaded within seconds rather than minutes. Microdrive cartridges are used with the Microdrive unit (see page 46).

Which is the best cassette player?
The Spectrum is happy with an inexpensive portable cassette player, preferably connected to the mains electricity supply rather than driven by batteries. The player should have its own volume control but a tone control is not essential. Special computer cassette players are also available. These are designed to store and load programs more reliably than ordinary machines.

A cassette deck that is part of a sound system is unlikely to be easy to connect up. Furthermore, the audio (line) output sockets of cassette decks do not produce a sufficiently strong signal for the Spectrum.

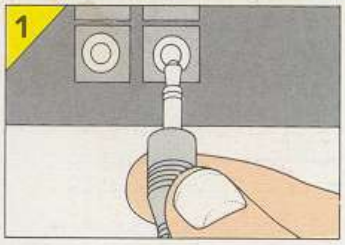
Do taped programs need any special care?
Like any form of magnetic storage, programs on cassette may be disrupted by strong magnetic fields. So don't store your cassettes near anything that uses a powerful electric current. Software cassettes also need to be kept fairly free of dust.

Does any kind of software work?
No. Only software produced for the ZX Spectrum or ZX Spectrum + will load.

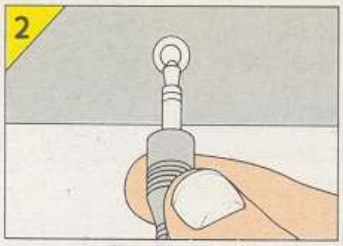
How to connect your cassette player

The cassette lead supplied with your Spectrum is intended to connect it to your cassette player. This is the lead with a pair of small plugs at each end. Place the cassette player beside your Spectrum and plug in the lead as shown. The cassette player and the Spectrum may be switched either on or off as you do this, though it's a good idea to take a cassette out of the cassette player before switching it on or off. This will safeguard programs stored on it.

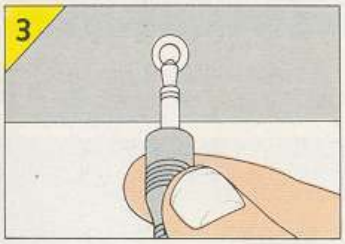
Making the right connections



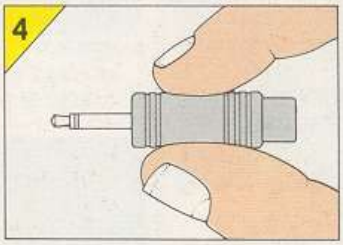
Push any one of the four plugs into the EAR socket on the back of your Spectrum.



Push the other plug of the same colour into the EAR socket on the cassette player if it has one.



If the cassette player does not have an EAR socket, connect the plug to a headphone socket if there is one. Otherwise try connecting it to an external loudspeaker socket.



If the plug on the cassette lead does not fit the socket on the cassette player, you'll need an adaptor or a special lead with the right plugs from an electrical dealer. The Spectrum EAR socket requires a 3.5 mm jack plug and an input signal of about 1 volt.

Software tips

■ The Spectrum's cassette lead has colour-coded plugs to prevent cross-connections between sockets on the computer and the cassette player. When you use a cassette player with your Spectrum, always try to keep to the same system, with one colour for the EAR sockets and the other for the MIC sockets.

■ Some cassette recorders may be affected by other electrical equipment nearby. Sometimes this can distort signals sent between the computer and cassette player with the result that programs will not load properly. If your cassette player occasionally does not seem to work, try moving it so that it is not alongside either the television or computer.

EAR and MIC Sockets
When loading programs, you can have both EAR and MIC sockets connected, as shown here. But when you are saving programs (see page 38), you *must* disconnect the EAR lead.

HOW TO LOAD A PROGRAM

Now that you've connected a cassette player to your Spectrum, you are ready to load and run a program. You can use a ready-to-run software tape or your own tape containing your programs. The procedure is exactly the same in both cases.

Switch on the cassette player. Make sure that the Spectrum is powered up, then insert the cassette into the player. If there is already a program in the computer, wait for it to end or stop it by pressing the BREAK key. You can then enter NEW or press the reset button to remove the program from the Spectrum's memory, but this is not essential as loading a new program clears the memory first. It is important to remember that if you load a program, the previous program will be erased from the memory.

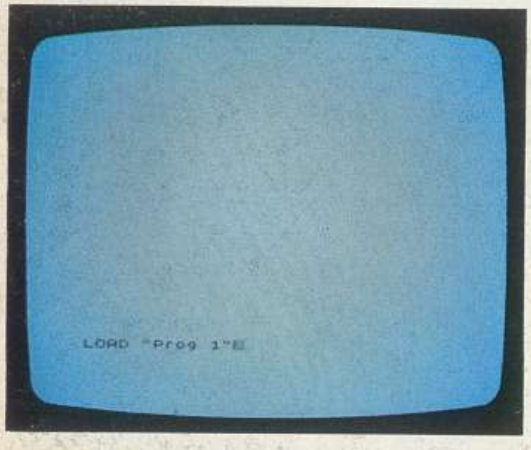
Now follow the numbered instructions. If anything goes wrong, consult the **Software loading troubleshooter** on page 16.

1 Insert the cassette and rewind it to the beginning.

2 Set the volume and tone controls on the cassette player to the required levels. Try the volume at about two-thirds maximum and if there is a tone control, set it to maximum treble.

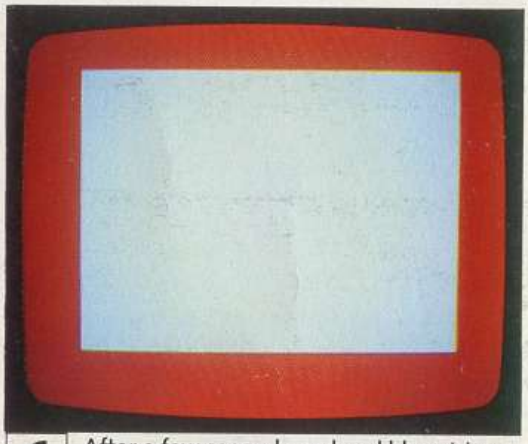
3 Press J and **LOAD** should appear on the screen. Then key in the program name in quote marks, for example

LOAD "Prog 1"



4 Press ENTER. The screen will go blank.

5 Start the tape. The border of the screen should go red or blue or flash red and blue. This indicates that the Spectrum is searching for a program.

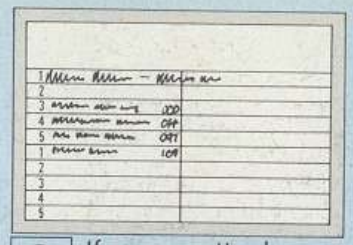


6 After a few seconds, red and blue stripes should begin to move up or down the border. This indicates that the Spectrum has begun to receive a signal.

Software loading tips

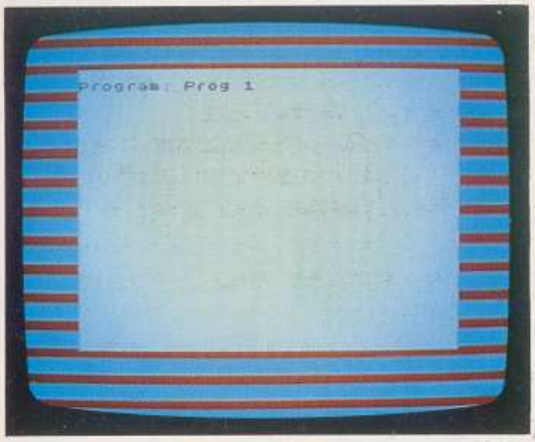
Here are some tips that will help you to save time when loading.

1 Label all tapes clearly so that you can find the programs easily. If a tape contains more than one program, write the names of the programs in order on the label. Remember to spell the program name exactly as the computer will need to use it.



2 If your cassette player has a counter, use it to find a program quickly on a tape with more than one program per side. Zero the counter at the beginning of the tape, then enter LOAD followed by any program name (in quotes) that is not on the tape. Play the tape and the Spectrum will display the name of each program that it finds without loading it. Write the counter numbers on the label beside the program names. This lets you quickly reach the program you want.

7 The word **Program:** followed by the program name, or **Bytes:** followed by a name or letter, appears on the screen. This indicates that the computer has successfully located the program.



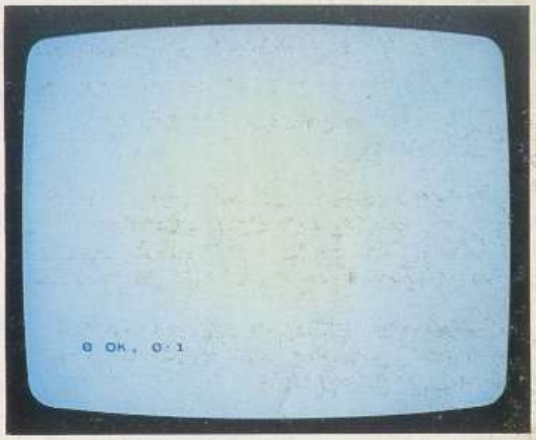
8 The red and blue stripes appear again as the computer waits to load the program.

9 A pattern of yellow and blue lines appears in the border. This indicates that the Spectrum is loading the program. Loading can take several minutes if the program is very long.

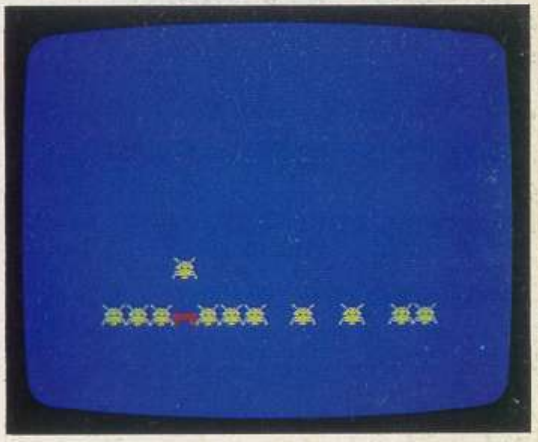
10 Operations 7, 8 and 9 may repeat once or several times if the program is split up into sections.

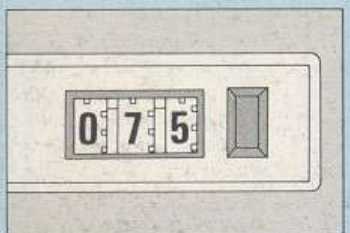
11 The program may begin automatically when it has loaded. Remember to stop the tape.

12 If the program does not begin automatically when it has loaded, the screen goes blank and the report **0 OK, 0:1** appears. Stop the tape.

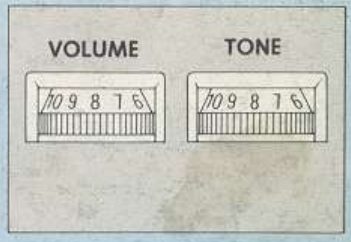


13 Press R (RUN) and ENTER, and the program will now begin.



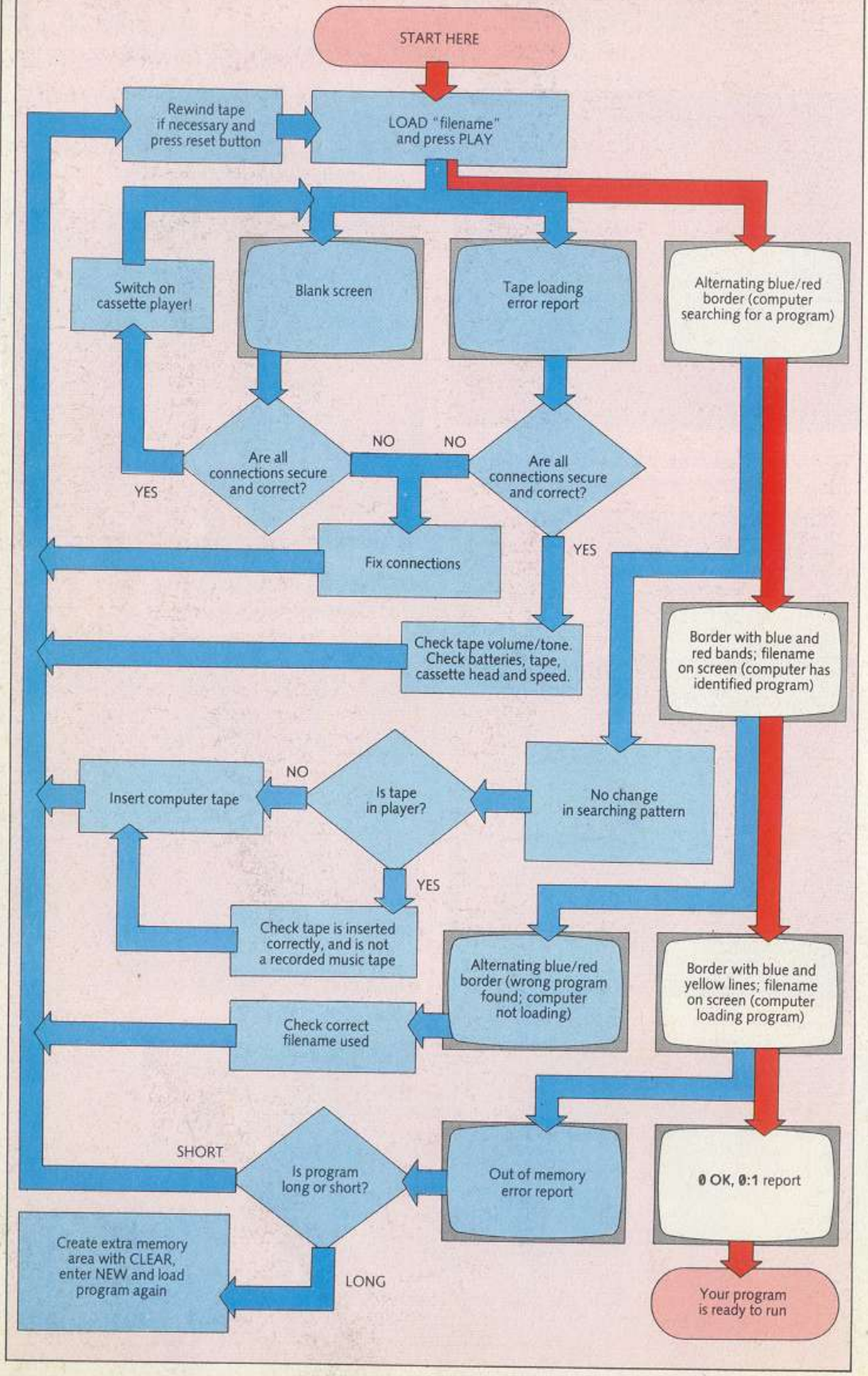


3 If the tape is at the correct program or if you do not know the name of the program, enter **LOAD " "** instead of **LOAD** followed by the name in quotes. There must be no space between the two quotes. Your Spectrum will then load the first program that it finds on the tape. If the program name that appears is not the one that you want, press BREAK, wind on and try again.



4 Note the levels of the volume and tone controls that enable your Spectrum to load. Set these levels on the cassette player before loading.

Software loading troubleshooter



START PROGRAMMING

This chapter is an introduction to program writing on the ZX Spectrum +. It tells you how to get to grips with your Spectrum by showing you how to find your way around the keyboard. Then you'll see how you can begin to put your Spectrum to work. The short programs for you to try out here concentrate on the Spectrum's special features so that when you come to writing programs of your own you'll be able to make the most of all your computer can do.

2

THE KEYBOARD – YOUR COMPUTER'S CONTROL PANEL

The ZX Spectrum + has its own language, the computer language known as BASIC. To get it to obey your instructions, you have to program the Spectrum by talking to it in BASIC. You do this by operating the Spectrum's keyboard. Furthermore, the keyboard allows you to control the computer while it is running your programs.

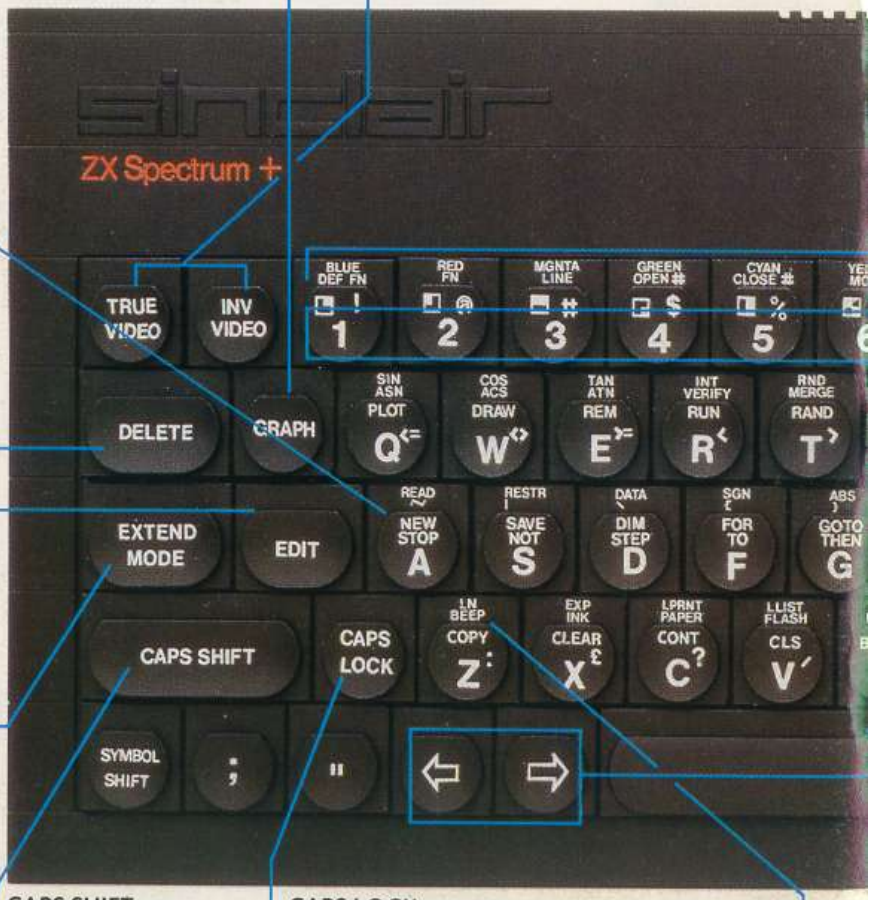
The dialect (version) of BASIC that the Spectrum understands is a simple but powerful form of this language. It's designed to be as much like English as possible to make it easy to use. In addition, the Spectrum has one great feature that makes programming much easier. This is the single-key keyword entry system.

Keys and keywords

Keywords are special words in BASIC that instruct the computer to do something – words like PRINT and INPUT. On most

GRAPH
This key is used to select the shapes or graphics characters on keys 1 to 8. If you press this key and then press a number key with or without the CAPS SHIFT key, a graphics character will appear on the screen.

TRUE VIDEO and **INV VIDEO**
These keys insert control codes into program lines to produce normal or inverse colours.



NEW
This key clears the computer's BASIC memory area, erasing any program held in it.

DELETE
This key is used if you press a wrong key and want to remove a keyword, letter, number or sign – see page 10.

EDIT
This key is used to change a line in a program without completely rewriting it – see page 21.

EXTEND MODE
This key selects the upper keyword above the raised section of any key. When followed by SYMBOL SHIFT and a key, it selects the sign or keywork immediately above the raised section of the key – see pages 20-21.

CAPS SHIFT
Press this key with a letter key to produce a capital letter. If you want a number of letters to appear in capitals, use CAPS LOCK.

CAPS LOCK
Use this key if you want the letter keys to produce capital (upper-case) letters all the time. Press it again to get small (lower-case) letters.

BEEP
This key produces the keyword which controls the Spectrum's sound synthesizer.

computers you have to key in each letter of a keyword as you would on a typewriter, and you must spell each word absolutely correctly. But on the Spectrum, you simply press a single key to get a whole keyword on the screen.

Sinclair BASIC has over 80 keywords, accessed by a total of 36 keys (26 letter keys and 10 number keys). Because the Spectrum uses such a wide range of BASIC instructions, many keys produce not one but several keywords which the computer will recognize. Most keys actually give you keywords as well as a letter, number, sign or even a shape (graphics character) all of which can be used in programs.

Selecting keywords and signs

On the Spectrum keyboard, there are two keys which you will be using a lot. These are EXTEND MODE and SYMBOL SHIFT, and they are the keys which let you choose which of the keywords and signs on the other keys you want to appear on the screen. You have already had a brief introduction to keying in with these keys on page 8. Now, after you have familiarized yourself with the keyboard layout, the next two pages will show you exactly how to select anything that appears on the computer's keyboard. Once you know how to do this, you can begin to write your own programs.

Colour display keys
These six keys produce keywords that control colour on the screen.

Number keys
As well as producing numbers, these keys can put control codes in programs for the colours shown – see page 33. The keywords immediately above the key-tops from 4 to 0, except key 8, are used only with ZX Microdrives.

BREAK
This key stops a program running. It does not erase the program from the computer's memory.

ENTER
This key tells the computer that the information just entered is complete and that it can go ahead.

SYMBOL SHIFT
Hold down and press a letter or number to select the lower keyword or sign on the raised section of the key. When used after EXTEND MODE, it selects the symbol or keyword immediately above the raised section – see pages 20-21.

Space bar
This produces a space like the space bar on a typewriter.

Cursor controls
Pressing these keys makes the cursor move in the same direction as the arrows. These keys are often used by programs to control the movement of shapes on the screen. They are also used when editing programs.

HOW TO OPERATE THE KEYS

You can get as many as six different keywords, letters, numbers or signs from most keys on your ZX Spectrum +. However, selecting a character or keyword on the board is not complicated once you become familiar with one of the Spectrum's special features. If you press a key, the result that appears on the screen depends upon the *mode* that the computer is in at that moment. The different modes each let you key in different types of information, like keywords, letters or graphics characters. The advantage of this is that as you operate the keyboard, the Spectrum actually helps you in choosing keyboard modes so that you enter instructions and information in the right order. On these two pages you will find out exactly what the modes do.

Keyword mode

Switch on or reset your Spectrum so that the copyright message appears. Now press ENTER. A flashing K appears in the bottom left-hand corner. The flashing square is called the *cursor*. It shows you where something is going to appear on the screen and the K indicates that the computer is in *keyword* mode. Press any letter key and the *top* keyword on the raised section of the key appears on the screen. Try Q, for example, and the keyword **PLOT** appears. Press the DELETE key to remove the keyword and try other keys. Number keys will give numbers, but as soon as you press a letter key, the top keyword on the raised section appears.

Use DELETE again so that the K cursor reappears. Now press either SYMBOL SHIFT key, hold it down and press any letter key. This time, the keyword or sign just above the letter on the raised section appears. With a number key, the sign to the right on the raised section appears. So, keyword mode relates to the *raised section* of the key.

How to select a keyword, symbol or character

You can see here how to select any keyword, sign or character on either a letter or number key. When selecting a key function, note where it is on the key, and then by using the two example keys here, decide which other keys – if any – you will need to change to the correct mode. Always look at the cursor on the screen first to see which mode the computer is in.

Letter key

Keyword (K) mode

Key only	**BORDER**
SYMBOL SHIFT and key	*

Extended (E) mode

EXTEND MODE then key only	**BIN**
EXTEND MODE then SYMBOL SHIFT and key	**BRIGHT**

Letter (L) mode

Key only	b
CAPS SHIFT and key	B
SYMBOL SHIFT and key	*

Capitals (C) mode

CAPS LOCK then key only	B
CAPS LOCK then SYMBOL SHIFT and key	*

Graphics (G) mode

GRAPH	then keys A to U only user-defined graphic

Number key

Keyword (K) mode

Key only	3
SYMBOL SHIFT and key	#

Extended (E) mode

EXTEND MODE then key only	magenta paper
EXTEND MODE then SYMBOL SHIFT and key	**LINE**

Letter (L) mode

Key only	3
SYMBOL SHIFT and key	#

Capitals (C) mode

CAPS LOCK then key only	3
CAPS LOCK then SYMBOL SHIFT and key	#

Graphics (G) mode

GRAPH then key only	▄
GRAPH then CAPS SHIFT and key	▀

Letter and capitals modes

Having produced a keyword or sign in keyword mode, the computer automatically changes the cursor to L. It is now in *letter* mode. Press any letter key and the lower-case (small) letter appears. Press a number and the number appears. If you want to get a capital letter, hold down CAPS SHIFT and then press the letter key.

If you want all capital letters, then press CAPS LOCK first. The cursor changes to C. Your Spectrum is now in *capitals* mode and you get a capital letter every time you press a letter key. You still get numbers in capitals mode. To return to letter mode (L), press CAPS LOCK again.

Extended mode

The next mode is called *extended* mode and it is produced by pressing the EXTEND MODE key. The cursor now changes to E. Press any letter key, and the top keyword of the pair of keywords *above* the raised section is given. For example, press B and you get **BIN.** To get the bottom keyword or sign above the raised section, press either SYMBOL SHIFT key first and hold it down, then press the letter key. On key B, for example, you now get **BRIGHT.** So in extended mode, you get the pair of keywords *above* the raised section of the key. After pressing a key (or EXTEND MODE) in extended mode, the computer automatically returns to letter or capitals mode.

Graphics mode

The fifth mode is called *graphics* mode and it is produced by pressing the GRAPH key. The cursor changes to G. Press keys 1 to 8 and see that the graphics characters marked on these keys appear. Now press CAPS SHIFT and any number from 1 to 8. The graphics appear again, but this time black and white are reversed. To leave graphics mode, you must always press GRAPH again, as the computer does not leave it automatically.

Editing on the Spectrum

When you give commands or when you come to writing programs for your Spectrum, you will want to correct mistakes in commands or program lines or to alter them. You can easily do this by editing.

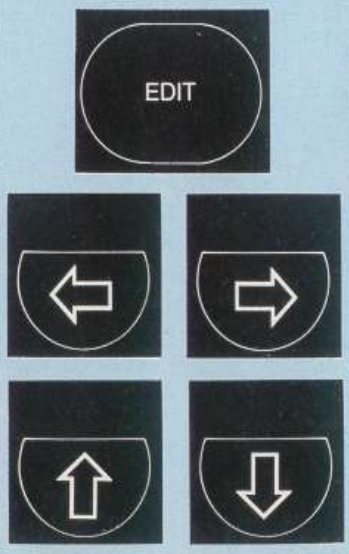
How to correct a mistake

If you try to enter a line or a command that is wrong in BASIC, the Spectrum will display a flashing ? before the error. To correct the mistake, hold down the left or right cursor control key to move the cursor to the right of the error. Then either delete the mistake by pressing DELETE or add whatever keyword, letter, number or sign is required. Then press ENTER.

For example, suppose you want the computer to multiply 7 by 8 and you do not press SYMBOL SHIFT to get the * sign. You would in fact key in

PRINT 7b8

instead. The Spectrum cannot obey this command, so on pressing ENTER it displays a flashing question mark before the b, which is where the mistake has occurred. All you need to do is move the cursor just to the *right* of the mistake, then press DELETE to remove b. Then press SYMBOL SHIFT and B to get * and press ENTER to make the computer obey the correct command; you don't need to move the cursor back to the end of the line first. The Spectrum carries out the command and displays the result.



How to edit a program line

When you write a program, you build up a sequence of numbered lines of instructions called a listing. If, after writing a program, you 'list' it by pressing K (LIST) and ENTER, you may see a > sign against one of the program's lines. If not, press and hold either the up or down key until the cursor appears. If you then press EDIT, the line is duplicated at the bottom of the screen and can then be changed as before with the cursor and DELETE keys. Press ENTER to place the new line in the program. If you want to edit another line, move the > sign with the up or down cursor control key to the line you wish to change and then press EDIT. If this takes too long, enter LIST followed by the line number and then press EDIT. In each case, the line you require will appear at the bottom of the screen and can be changed.

To delete a complete line from a program, simply key in the line number alone and then press ENTER. If you do run a program that contains an error, you will see a screen report. These are explained on page 74.

THE TELEVISION CALCULATOR

The ZX Spectrum + can make calculations extremely quickly and with great accuracy. All it needs are some numbers to work on and signs such as + and – that tell it what to do with the numbers.

First key in this instruction (you'll find the + sign on the K key):

```
PRINT 6+2
```

This is a *command*. When you press ENTER, the command disappears and the answer, the number 8, is printed on the screen.

Your Spectrum uses five signs known as *arithmetic operators* for calculations. You can see what they each do in the panel at the bottom of this page. You can use them all in just the same way with PRINT.

Entering commands such as PRINT 6+2 turns your Spectrum into a calculator. But it can do many things that an ordinary calculator cannot. For a start, it can display calculations and their results together. Enter this command:

```
PRINT "6+2=";6+2
```

The computer responds by displaying

```
6+2=8
```

What happens is that PRINT causes *everything* between double quote marks (") to be displayed on the screen, so 6+2= appears. The characters between the quote marks make up a *string*. The semicolon instructs the Spectrum to display the result immediately after the equals sign.

The Spectrum's calculation signs

The following signs or 'arithmetic operators' are used by the Spectrum to carry out mathematical operations. Note that the computer does not use × or ÷ signs.

Symbol	Key	Function	Example
+	K	Add two numbers	8+2=1Ø
–	J	Subtract two numbers	8–2=6
*	B	Multiply two numbers	8*2=16
/	V	Divide two numbers	8/2=4
↑	H	Raise first number to the power of the second	8↑2=64

Your first program

When a command has been carried out, your Spectrum forgets it. If you want the computer to repeat the calculation, you can write it as a program. Key in this instruction and then press ENTER.

```
1Ø PRINT 6+2
```

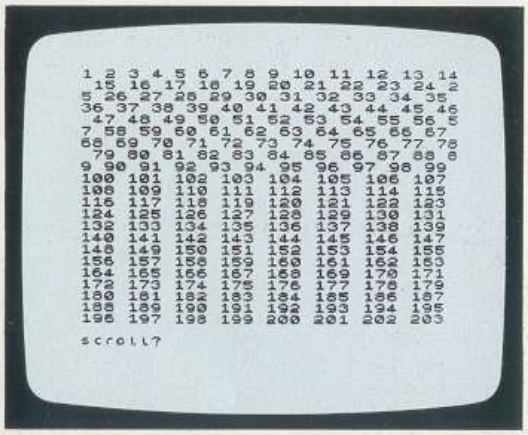
This time, it is not obeyed straightaway. The computer displays the instruction on the screen instead. Next press R (RUN) and ENTER. The result 8 now appears.

The whole instruction is now a computer program. Putting a number at the beginning makes your Spectrum place the instruction in its memory, but not carry it out until told to do so. Whenever you run a program by pressing R (RUN) and then ENTER, the instruction is carried out. The instruction is now called a *statement* instead of a command, and it forms a numbered line in a program. Program statements are always carried out in order of their line numbers, and these usually go in tens so that extra lines can be inserted later if necessary.

Next, get the Spectrum really working. Enter this program. Remember to press ENTER after keying in each line, and then when you have finished, press R (RUN) and ENTER. When you have run the program, this is what you should see.

NUMBER CHART

```
10 LET n=1
20 PRINT n;" ";
30 LET n=n+1
40 GO TO 20
```



All the numbers from 1 to 2Ø3 are displayed. Now press any key except N, the space bar, STOP or BREAK. A whole new set of numbers appears.

This program uses a *variable*. In this case, the variable is called n. Any letter or word could be used – n here simply stands for

number. A variable is given a value that changes as the program runs. In line 10, the keyword LET is used to set the value to 1. Line 20 displays the value followed by a space. Then in line 30, LET is used again, this time to increase the value by 1, so n becomes 2. Line 40 uses the (single) keyword GOTO to send the program back to line 20, which now displays 2. This is repeated over and over again until the numbers fill the screen.

How to make a program ask for a number

Stop the program by pressing BREAK. Now key in a new line

10 **INPUT** n

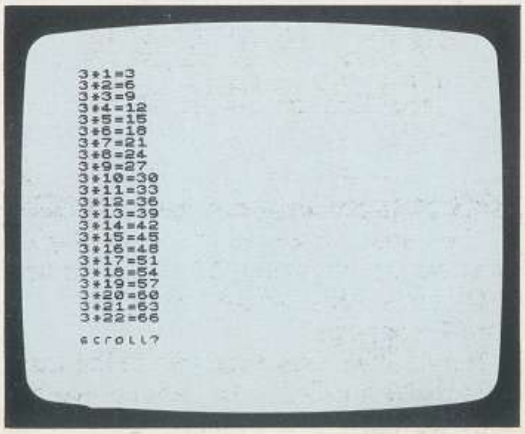
This line replaces the old line 10 in the program. When you run it, the computer now waits for you to enter a number. Key in any number and press ENTER. Now the numbers begin at the number you entered. This is because INPUT n makes the value of n equal to the number you enter. INPUT instructs the computer to ask for information during a program.

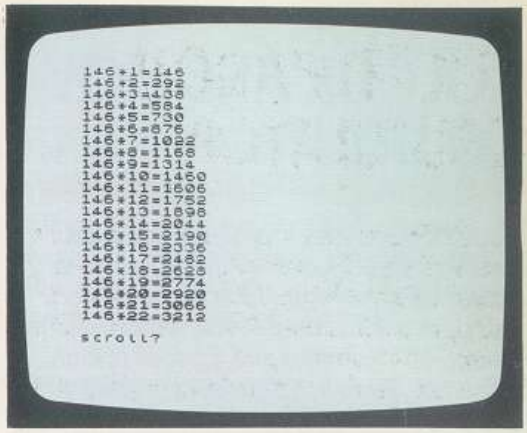
Programming a multiplication table

Press the reset button to remove the old program and enter the next one. This program gets the Spectrum multiplying. Key in any number and a multiplication table for the number will flash up on the screen. Press any key except N, BREAK or the space bar, and the table continues. Press BREAK and then run the program again to create a new table. Here is the program and what you should see if you key in 3 and next 146.

MULTIPLICATION TABLE

```
10 LET x=1
20 INPUT n
30 PRINT n;"*";x;"=";n*x
40 LET x=x+1
50 GO TO 30
```





Why you need to use brackets

You will sometimes need to use brackets in a calculation. Enter these two commands and compare the results:

PRINT 6+2 /4
PRINT (6+2)/4

The first gives 6.5 and the second gives 2. The reason for these different results is that the computer has an in-built system of priorities which it uses in calculations. It carries out ↑ first, then * or /, and finally + or −, but it *always* carries out any calculations in brackets first. So, in the first command above, it first divides 2 by 4 and then adds the result (0.5) to 6. In the second command, the computer adds 6 and 2, and then divides by 4.

How to punctuate with your Spectrum

The Spectrum uses a range of punctuation signs. They are very important because many of them double up as instructions to the computer, affecting the way it understands a program line or produces a display.

;	**Semicolon** When used with PRINT, tells the computer to display the two items on either side of it next to each other on the screen.
:	**Colon** Signals the end of one statement in a program line and the beginning of the next.
"	**Quote mark** Any characters within quotes are treated not as numbers or variables but just as text. Quotes begin and end a *string*.
,	**Comma** When used with PRINT, tells the computer to display the following item either in the centre of the line or at the beginning of the next line. Do *not* use to indicate thousands or millions.
.	**Point** Either a decimal point or full stop.

COLOUR AND HOW TO USE IT

Your ZX Spectrum + can produce eight different colours, and each colour has a colour code number. You can use each colour in three different ways – as a border colour, an ink colour and a paper colour.

ZX Spectrum + colour codes

This chart shows the colours and codes used by the Spectrum. You don't have to remember these codes; the number keys that produce them are also marked with the colour names. (These names are *not* keywords.)

Number	Colour
0	Black
1	Blue
2	Red
3	Magenta
4	Green
5	Cyan
6	Yellow
7	White

The actual shades you get on your television set will depend on the set and the adjustment of the colour, contrast and brightness controls. Remember that you need a colour set.

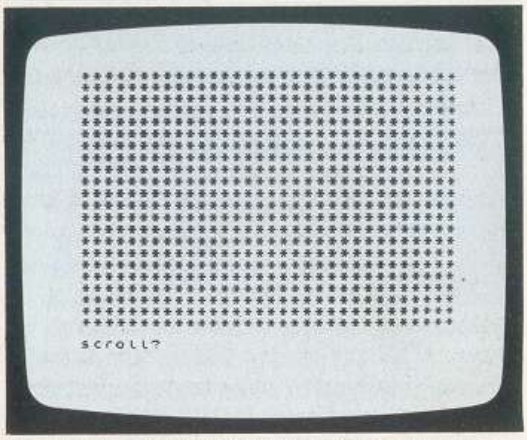
The Spectrum's three ways of using colour

You can control colour in three different ways. The *border* colour is the colour of the border around the central display area. The *ink* colour is the colour in which characters (letters, numbers, signs and graphics shapes) and points or lines appear. The *paper* colour is the colour of the background, either over the whole display area or in a square just around each character.

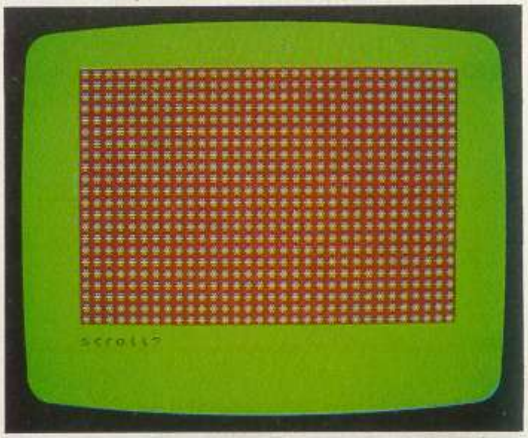
When you turn the Spectrum on, it uses the preset colours. The ink colour is black, and the border and paper colours are white. You can change these colours instantly by entering direct commands from the keyboard. You have already seen this at work on pages 6–7, where the BORDER command was used to check that your television and Spectrum were both properly set up for colour. Now press the reset button and key in and run this simple program.

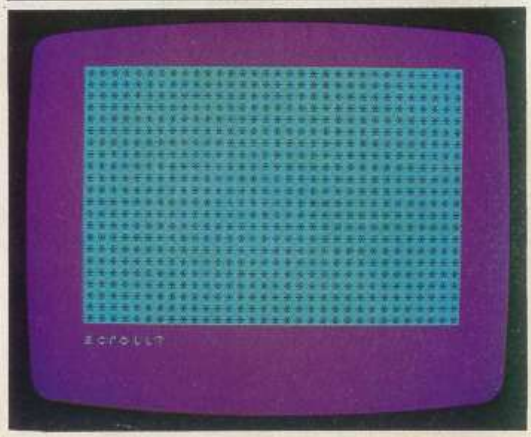
COLOUR TESTER

```
10 PRINT "*";
20 GO TO 10
```



A pattern of stars builds up in black and white. Now press BREAK and enter some colour commands. Key in the keywords BORDER, INK and PAPER each followed by a number from 0 to 7, pressing ENTER after each, and then run the program again. Here are two displays, the first with BORDER 4, PAPER 2 and INK 7, and the second with BORDER 3, PAPER 5 and INK 1.





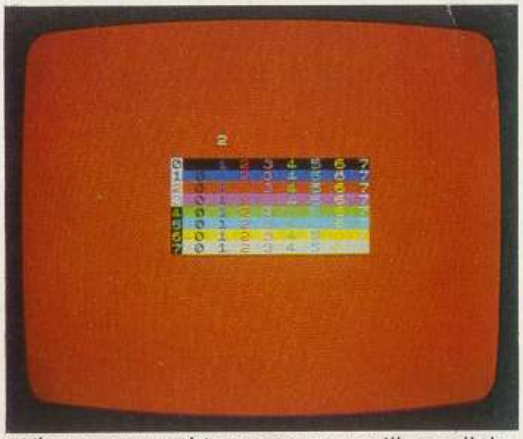
How to write programs with colour

You can use the BORDER, PAPER and INK keywords in a program to make text, tables, patterns and pictures appear in all kinds of colours. Using BORDER in a program line makes the border colour change as soon as the Spectrum reaches this line. INK in a line on its own gives a new ink colour when any characters or lines next appear on the screen. PAPER in a line on its own changes the paper colour but only around any characters (this includes any points or lines as well). If you want the whole background of the display area to be a certain colour, you must follow PAPER with CLS.

You can also use INK and PAPER after PRINT. In this case, only the particular characters displayed by PRINT have these INK and PAPER colours. The next program demonstrates all the border, ink and paper colours. It also shows you how to use INK and PAPER after PRINT.

COLOUR COMBINATIONS

```
  10 FOR b=0 TO 7
  20 BORDER b: PAPER b: CLS
  30 PRINT AT 6,12; INK 9;b
  40 FOR p=0 TO 7
  50 PRINT AT p+8,8; INK p; PAPE
R 9;p;
  60 BEEP 0.5,b*p-20+p
  70 FOR i=0 TO 7
  80 PRINT INK i; PAPER p;" ";i;
  90 BEEP 0.01,i*5
 100 NEXT i
 110 NEXT p
 120 NEXT b
```



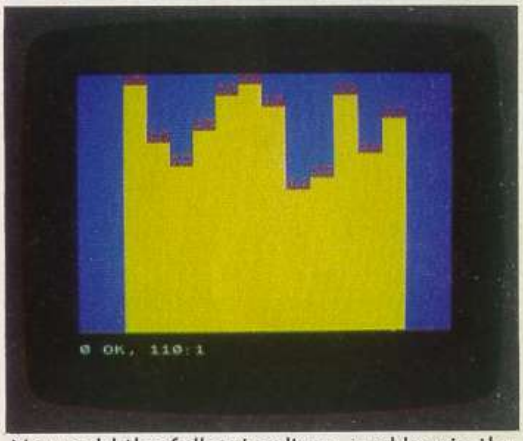
When you run this program, you'll see all the combinations of border, paper and ink. The program has three variables, b for the border number, i for the ink number and p for the paper number. BEEP produces the sound, and the lines beginning with FOR and NEXT mark the beginning and end of a program loop that changes all the colour numbers from 0 through to 7 in order. You'll find out more about using FOR and NEXT in program loops on page 27. Note that INK and PAPER can both have a value of 9. This makes the ink or paper colour either black or white so that it shows up against the background or a character.

Programming coloured bar charts

The next program uses the Spectrum's colours to produce a bar chart. It shows twelve daytime temperatures as yellow columns with numbers. In line 60, enter two spaces between the quote marks.

BAR CHART

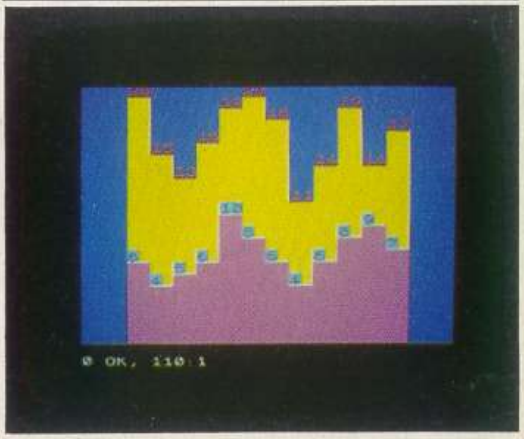
```
  10 BORDER 0: PAPER 1: CLS
  20 LET c=4
  30 FOR x=1 TO 12
  40 READ t
  50 FOR l=21 TO 21-t STEP -1
  60 PRINT PAPER 6;AT l,c;"  "
  70 NEXT l
  80 PRINT INK 2;AT 20-t,c;t
  90 LET c=c+2
 100 NEXT x
 110 DATA 20,15,13,16,19,20,18,1
1,12,19,14,17
```



Now add the following lines, and key in the new line 110 as shown. The chart now appears in two colours. To find out about READ and DATA, turn to page 33.

DOUBLE BAR CHART

```
  85 READ t
  86 FOR l=21 TO 21-t STEP -1
  87 PRINT PAPER 3;AT l,c;"  "
  88 NEXT l
  89 PRINT INK 1; PAPER 5;AT 20-
t,c;t
 110 DATA 20,6,15,4,13,5,16,6,19
,10,20,8,18,6,11,4,14,6,19,8,14,
9,17,7
```



SIMPLE DIY GRAPHICS

Your ZX Spectrum + can give you low-resolution and high-resolution graphics. Both kinds can appear on the screen at the same time. Low-resolution graphics displays are made up of blocks of colour. On these two pages, you will see how to produce these blocks from the keyboard, and how to position them on the screen.

The low-resolution screen

On the low-resolution screen there are 32 positions in which characters can be placed across the screen and 22 positions in which they can be placed down it. Each screen position has a pair of numbers to identify it. First comes the line number, which is the number of lines *down* the screen to reach the position. The top line is line 0 and the bottom line is line 21. Next comes the column number, which is the number of columns *across* the screen to reach the position. The left-hand column is column 0 and the right-hand column is column 31. (On page 80 you can see the low-resolution grid laid out.) The next program fills these character positions with colours. The keyword RND (on the R key) chooses a random ink colour.

RANDOM SQUARES

```
10 BORDER 1: INK RND*7
20 PRINT "■";
30 GO TO 10
```

Here squares appear all over the screen. To make a character appear at a particular position, you need to use the keyword AT together with PRINT. AT is placed after PRINT and followed by the line number, a comma, the column number and a semicolon. This command

PRINT AT 11,16;"*"

for example, displays a star at line 11, column 16, which is the centre of the screen.

How to draw rainbow patterns

A good way of producing coloured patterns is to use FOR NEXT loops in your graphics programs. FOR NEXT loops are parts of a program that repeat themselves a certain

How to select graphics characters

Your ZX Spectrum + has a set of keyboard graphics characters that make low-resolution graphics easy to program. You can see them on keys 1 to 8.

To produce the graphics characters on the screen, press the GRAPH key and then press keys 1 to 8, using the space bar between each one. The graphics characters appear at the bottom of the screen. The white part of each character on the key is the ink colour and the black part the paper colour. Now press the keys again, holding down CAPS SHIFT at the same time. This time the characters appear with the ink and paper colours reversed.

This is exactly how you put graphics characters into program lines. To leave graphics selection and return the number keys to normal, simply press GRAPH again.

GRAPH
This key is used to switch the Spectrum to graphics mode.

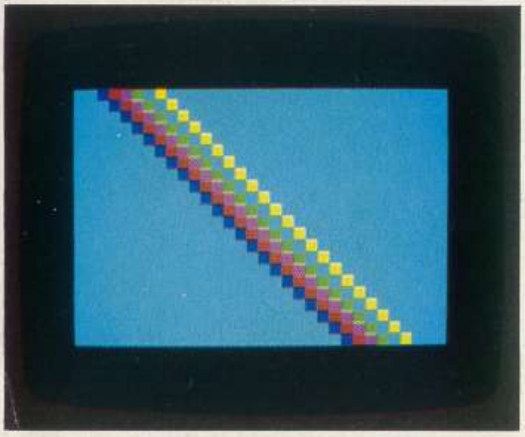
Key 8
This key is often used with GRAPH and CAPS SHIFT to produce a solid square of colour.

number of times. In the line that begins a loop, you can tell the computer how many times you want the loop to be carried out. As it does so, it can be used to place characters on the screen, for example.

You aren't limited to programming just one loop at a time. You can put one loop inside another, often with very useful results. The next program shows you how two FOR NEXT loops (one 'nested' inside the other) can be used to change the colours and positions produced by INK and AT. You can see how to program these loops in the panel at the end of this page.

RAINBOW

```
  5 BORDER 0: PAPER 5: CLS
 10 LET x=1
 20 FOR l=0 TO 21
 30 FOR c=1 TO 6
 40 PRINT INK c;AT l,c+x;"■"
 50 NEXT c
 60 LET x=x+1
 70 NEXT l
```



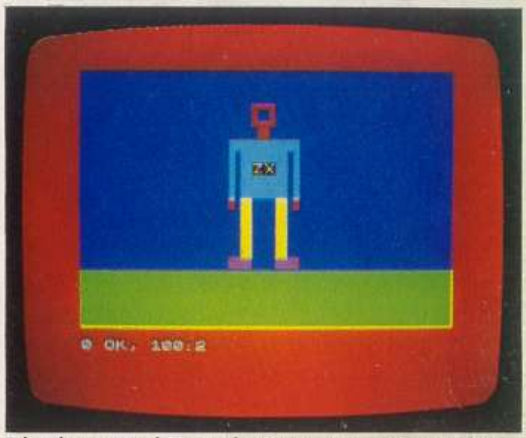
Programming pictures

In low-resolution graphics, you can 'paint' pictures by working out the positions and colours of the graphics characters. You can plan your own picture by using the low-resolution grid on page 80. Then, by selecting the graphics characters in the way shown on the opposite page, enter the program lines one by one to build up the picture.

The next program shows you the sort of results you can produce. All the shapes in it can be found on the number keys. You can either wait until you have keyed in all the lines before running the program, but if instead you run it after entering each line, you'll see how the different parts of the robot are put together. (Remember that if you key in the wrong graphics characters, you can edit them just as you would edit incorrect numbers or letters.)

ZX ROBOT

```
  5 BORDER 2: PAPER 1: CLS
 10 PRINT INK 2;AT 3,15;"▐▀▌"
 15 PRINT INK 2;AT 4,15;"▐▄▌"
 20 PRINT INK 2;AT 5,15;"▝▘"
 30 PRINT INK 5;AT 6,13;"▟▀▀▀▙
"
 40 FOR l=7 TO 10: PRINT INK 5;
AT l,13;"▌▀▀▀▐": NEXT l
 45 PRINT INK 6; PAPER 0;AT 8,1
5;"ZX"
 50 PRINT INK 2;AT 11,13;"▙
▟"
 60 FOR l=11 TO 15: PRINT INK 6
;AT l,14;"▌ ▐": NEXT l
 70 PRINT INK 3;AT 16,13;"▀▀";T
AB 17;"▀▀"
 80 FOR l=17 TO 21: FOR c=0 TO
31
 90 PRINT INK 4;AT l,c;"■"
100 NEXT c: NEXT l
```



The keyword TAB that appears after PRINT in line 70 is used to position a character along the line that the computer is currently working at. TAB is followed by one number from 0 to 31, specifying a column position.

How to use FOR NEXT loops

A FOR NEXT loop always begins with a line containing the keywords FOR and TO, together with a variable and its beginning and end value, for example

30 **FOR** c=1 **TO** 6

Here the variable is c. The loop that this begins would then contain line(s) that make the computer repeat an operation. They might also use the variable c themselves. FOR NEXT loops always end with the keyword NEXT and the variable, for example

50 **NEXT** c

When the program is run, the whole loop from FOR to NEXT repeats a set number of times. The variable begins at the first value before TO and increases by 1 each time until it reaches the limit after TO. In this case, the loop repeats six times, with c starting at 1 and then becoming 2, 3, 4, 5 and finally 6.

In the first program on page 25, three loops are used in a 'nest'. This means that for every cycle of the 'outside' loop, the 'middle' one goes through all its cycles. The 'inside' loop goes around all its cycles most often, every time the 'middle' loop cycles once.

THE ON-SCREEN SKETCHPAD

Graphics on the ZX Spectrum + are not limited to chunky low-resolution patterns and pictures. With its high-resolution capability, you can use your Spectrum to create detailed images with sharp outlines and straight or curved lines and edges.

High-resolution graphics are made up of many dots placed one after another to form a line or to fill out a shape in solid colour. Each dot is a sixty-fourth of the size of the squares you use in low-resolution graphics. If you enter this command

PLOT 128,87

you'll see one in the centre of the screen.

The dots used in high-resolution graphics are called *pixels*, which is short for picture cells. Like a low-resolution character, each pixel requires two numbers to specify its position. These are not the same numbers as those used in low-resolution displays.

The high-resolution grid

The high-resolution grid consists of 256 pixels across the screen and 176 down. However, unlike in low-resolution displays, the first number is the horizontal coordinate – its position *across* the screen. These position numbers go from 0 at the left-hand edge to 255 at the right-hand edge. The second number is the vertical coordinate, but the numbers go from 0 at the bottom to 175 at the top. Position 0,0 is the bottom left-hand corner, not the top left-hand corner as in low resolution. See page 80 for a chart of the high-resolution grid.

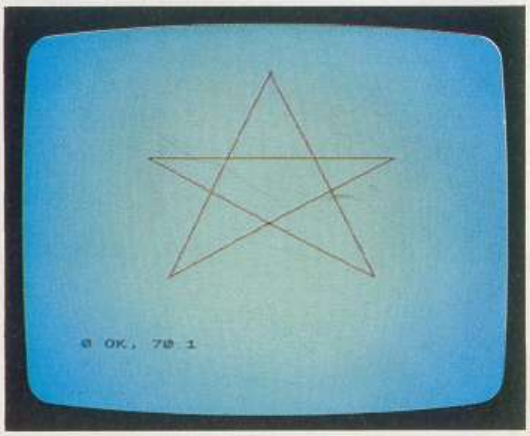
Plotting and drawing

You need only three keywords for producing high-resolution graphics – PLOT, DRAW and CIRCLE. PLOT is followed by the horizontal and vertical coordinates separated by a comma, and it places a pixel at this position. DRAW is also followed by two numbers separated by a comma, but these are *not* the coordinates of a position. Instead they are the distances in pixels from one position to another position across and up or down the screen, and DRAW then draws a line between the two positions.

The first position is 0,0 if PLOT or DRAW have not already been used in the program. If they have been used, then this position is the last PLOT position or the last position reached by DRAW, whichever is most recent. The DRAW statement then draws the line to the new position. If the line is to go to the left or down the screen, then the horizontal or vertical distances must be negative (minus) values. Try this program.

STAR

```
10 INK 2
20 PLOT 128,174
30 DRAW 70,-140
40 DRAW -152,80
50 DRAW 164,0
60 DRAW -150,-80
70 DRAW 70,140
```

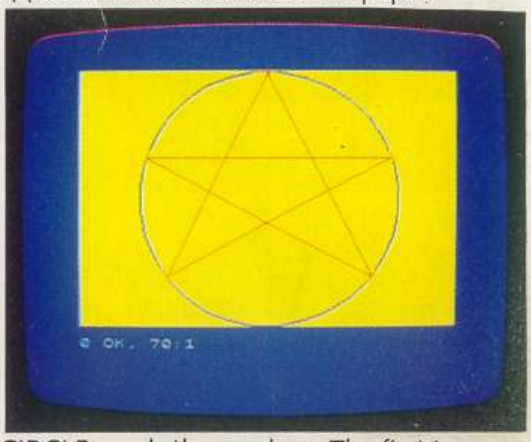


PLOT moves the start position to the top of the screen. Then the five DRAW statements draw the five red lines.

Now add these lines to the program.

```
4 BORDER 1:PAPER 6:INK 1:CLS
5 CIRCLE 128,87,87
```

Run the program again, and the red star appears in a circle on coloured paper.



CIRCLE needs three values. The first two give the position of the centre of the circle, and the third number is its radius. You can also add a third value to DRAW statements.

Try values between 2 and −2 with the program and see what happens.

How to fill in shapes

You can easily produce solid shapes in high resolution by drawing many lines close together. This can be done with a FOR NEXT loop that changes the DRAW positions so that they increase by 1 each time.

SOLID TRIANGLE

```
  10 BORDER 1: PAPER 6: INK 2: C
LS
  20 FOR x=-100 TO 100
  30 PLOT 128,150
  40 DRAW x,-120
  50 NEXT x
```

You get an interesting effect if you draw lines slightly apart. You can do this by adding the keyword STEP and a number to the FOR statement. This technique is used in the Shimmering sunrise program on page 11. It will work on the triangle program in the same way. Enter a different line 20 and run the program again

```
20 FOR x=-100 TO 100 STEP 4
```

This time the fan-like shape shown below emerges. The reason for this is that STEP makes x increase in jumps of 4 instead of increasing by 1 each time a line is drawn.

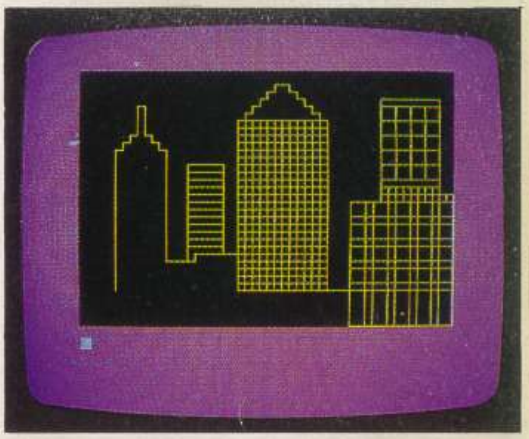
Your screen sketchpad

Your Spectrum is very versatile. You don't need to write a program every time you want to produce a picture or pattern. Instead, you can use a program that allows *you* to build up a picture directly on the screen. Here's a very simple program that allows you to do this.

It starts by using the keyword INPUT to ask you for an ink number. Then, using INPUT again (this time with a $ sign to label a string), it makes the computer draw short lines every time you press one of four specified keys – u, d, l and r.

SKETCHPAD AND EXAMPLE

```
  10 INPUT "INK ";i
  20 BORDER 3: PAPER 0: INK i: C
LS
  30 PLOT 25,25
  40 LET x=5
  50 INPUT k$
  60 IF k$="u" THEN DRAW 0,x
  70 IF k$="d" THEN DRAW 0,-x
  80 IF k$="r" THEN DRAW x,0
  90 IF k$="l" THEN DRAW -x,0
 100 GO TO 50
```



Making decisions with IF and THEN

Lines 60 to 90 in the Sketchpad program contain IF THEN statements. These allow your Spectrum to make a decision. In this case, the computer checks to see if the key you have pressed is either u, d, l or r. IF any of these are pressed, THEN the computer is told to draw a line. It won't draw a line if a capital letter is entered.

IF is always followed by something that the Spectrum tests to see if it is true or if it is happening – like certain keys being pressed. If it is true or happening, then the action following THEN is carried out. If not, then the program goes to the next line.

Everything following THEN in a line is subject to the decision. In this line

```
110 IF b=5 THEN PRINT "*" : GOTO 200
```

the computer will only go to line 200 if b is 5.

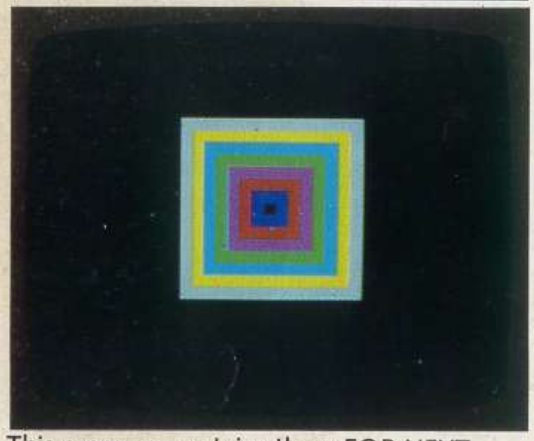
DESIGNING PATTERNS AND PICTURES

You can produce all kinds of patterns and pictures with your ZX Spectrum +, using either low-resolution graphics, high-resolution graphics or both. The best way to tackle graphics is first to draw out your design on a copy of the grids on page 80. Then work out the program that will produce the lines and shapes at the right positions.

To draw patterns and pictures, you can often use FOR NEXT loops that repeat part of a program a set number of times. Each time, the positions and colours of the characters or lines can change, usually in a regular way. Here is a program which uses this technique.

SQUARES

```
10 BORDER 0: PAPER 0: CLS
20 FOR x=7 TO 0 STEP -1
30 INK x
40 FOR l=11-x TO 11+x
50 FOR c=16-x TO 16+x
60 PRINT AT l,c;"■"
70 NEXT c
80 NEXT l
90 NEXT x
```



This program contains three FOR NEXT loops. The x loop changes the colour and also the size of the big squares that are produced, while the l loop and the c loop change the line and column position of the little square every time it is printed. Try changing the square in line 60 to a star or some other character on the keyboard.

Random effects and subroutines

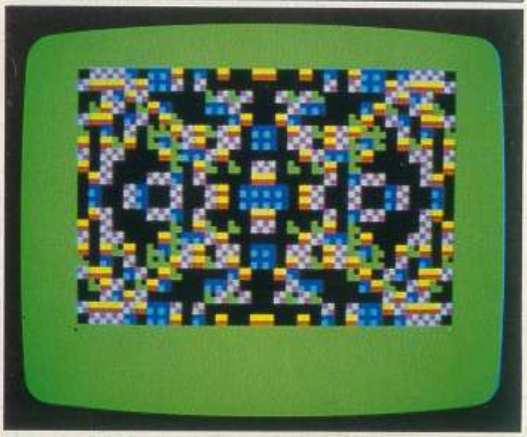
Using loops need not give identical patterns each time a graphics program is run. By using the keyword RND (short for RaNDom) in loops, you can make colours, positions and other display features different every time. Look at the mosaic program on page 10. It works because the ink colour is RND*7, which means any number with a decimal point from 0 to 7. INK changes this to the nearest whole number. So each time a square is displayed, its colour is any colour from INK 0 to INK 7.

The next program draws symmetrical patterns of graphics characters on the screen. It uses RND to change these characters and their positions. The variables i and p give the ink and paper colours, and a indicates how many patterns are drawn (in this case four). The variable n gives the number of characters in each pattern, while x is a random number from 129 to 142.

The statement GOSUB 1000 in line 50 sends the computer to a subroutine.

SYMMETRICAL PATTERNS

```
  10 BORDER 4: PAPER 0: CLS
  20 LET i=4: LET p=0
  30 FOR a=1 TO 4
  40 LET x=RND*13+129
  50 FOR n=1 TO 40: GO SUB 1000:
NEXT n
  60 LET i=i+1: LET p=p+1
  70 PAUSE 100
  80 NEXT a
  90 STOP
1000 LET l=INT (RND*11)
1010 LET c=INT (RND*16)
1020 INK i: PAPER p
1030 PRINT AT l,c;CHR$ x
1040 PRINT AT l,31-c;CHR$ x
1050 PRINT AT 21-l,c;CHR$ x
1060 PRINT AT 21-l,31-c;CHR$ x
1070 BEEP 0.01,l+c/3
1080 RETURN
```



A subroutine is a group of lines that acts like a program-within-a-program. In this program the subroutine is at line 1000. It displays a graphics character in four quarters of the screen so that each one is the same distance from the centre (position 11,16). This distance is given by lines 1000 and 1010, l giving the distance in lines and c the distance in columns. INT changes the random number to a whole number so that l is any whole number from 0 to 10 and c any whole number from 0 to 15. Then lines 1030 to 1060 display the graphics character

whose code is x (see the character set table on page 51). BEEP makes a sound whose pitch is related to the position, and then RETURN in line 1080 sends the program right back to the next statement after GOSUB in line 50.

Line 60 changes the ink and paper colours, then PAUSE 100 in line 70 delays the program for 2 seconds before it loops back to begin again. STOP is needed in line 90 to stop the program running straight into the subroutine after the fourth loop.

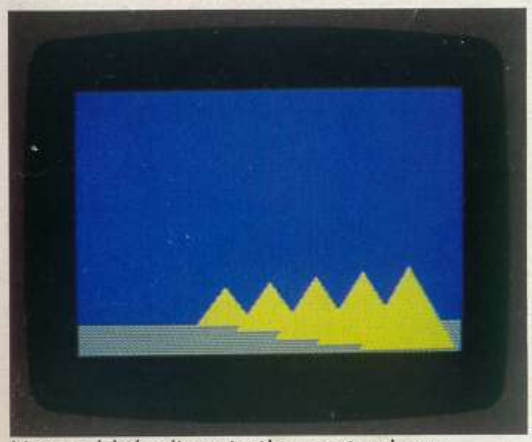
You can change this program by altering 4 in line 30, and 40 in line 50 to other numbers. If you make the range of x wider in line 40, you'll get other characters appearing on the screen. Do not allow i and p to be greater than 7.

Using FOR NEXT loops in graphics

FOR NEXT loops can be used very effectively in high-resolution graphics to create pictures made up of regular shapes and lines. Key in the following program and run it. Using only PLOT and DRAW, the two FOR NEXT loops first draw lines on the ground and then five solid triangles or pyramids.

PYRAMIDS

```
 10 BORDER 0: PAPER 1: INK 6
 20 CLS
 30 FOR y=0 TO 20 STEP 2
 40 PLOT 0,y
 50 DRAW 255,0
 60 NEXT y
 70 FOR n=100 TO 220 STEP 30
 80 FOR x=-10-n/10 TO 10+n/10
 90 PLOT n,35+n/10
100 DRAW x,-n/4
110 NEXT x: NEXT n
```



Now add the lines in the next column. When you run it again, you'll find that a laser beam continually shoots up into the night sky, creating bursts of stars. It is drawn from the corner of the screen to position x,y, the variables x and y being random numbers. These are then converted to low-resolution star position numbers.

```
120 LET x=RND*255
130 LET y=RND*104+71
140 LET l=INT ((175-y)/8)
150 LET c=INT (x/8)
160 PLOT 0,0: DRAW OVER 1;x,y
170 BEEP 0.01,x/4
180 PLOT 0,0: DRAW OVER 1;x,y
190 PRINT AT l,c;"*"
200 GO TO 120
```

OVER 1 in lines 160 and 180 allows the first line to draw the laser beam and the second line to remove it *without* changing the rest of the picture. Save this program (see page 38) as you will need it later.

FLASH, BRIGHT and INVERSE

These three keywords can really make the colours of the Spectrum work for you. Each keyword is followed by either 0 or 1, and you can put them in PRINT statements provided you put a *semicolon* after the 0 or 1. FLASH 1 makes character positions flash between the ink and paper colours, while BRIGHT 1 makes the colours brighter. INVERSE 1 changes the ink colour to the paper colour and vice versa. Using 0 after these keywords restores the display to normal.

Try making these changes to the programs on these two pages to see how the keywords work. In the Squares program, change the square in line 60 to a star and then add

```
15 INVERSE 1
```

Now the stars appear in black (the paper colour) against coloured bands (the changing ink colours). Enter **INVERSE** 0 before continuing.

In the Symmetrical Patterns program, add these lines to see how BRIGHT and FLASH work.

```
15 BRIGHT 1
16 FLASH 1
```

Note how FLASH makes the pattern appear to move to and fro. Enter **FLASH** 0:**CLS** to stop the display flashing.

All these changes affect the whole display produced by each program. Using FLASH, BRIGHT or INVERSE within a PRINT statement restricts the three keywords to whatever is printed by that line.

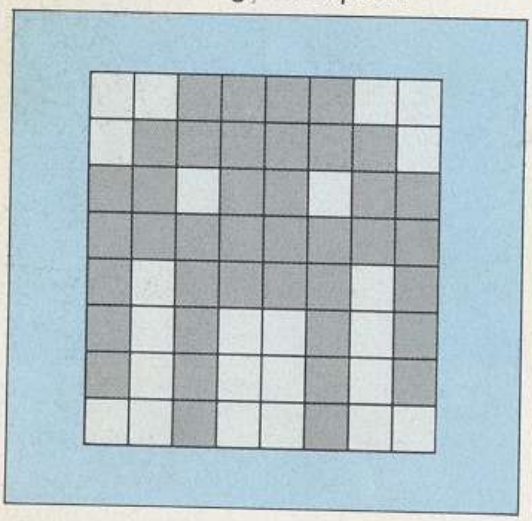
HOW TO CREATE COMPUTER CHARACTERS

Your ZX Spectrum + is not limited just to the graphics characters that you can key in at the keyboard. In a special section of its memory, it can store other characters that you design yourself. These are called user-defined graphics characters and each program can have a maximum of 21 of them.

Each character is made of up to 64 little dots or pixels of ink colour. These are arranged in eight rows of eight pixels each, and each character occupies one character position on the low-resolution grid – just like the standard graphics characters on the keyboard.

Designing a graphics character

First draw an 8×8 grid as shown below. Then fill in some of the squares to create the character. These squares represent the ink-coloured pixels. Then draw in or think of each full square as 1 and each empty square as Ø. Here is a design for a spider.



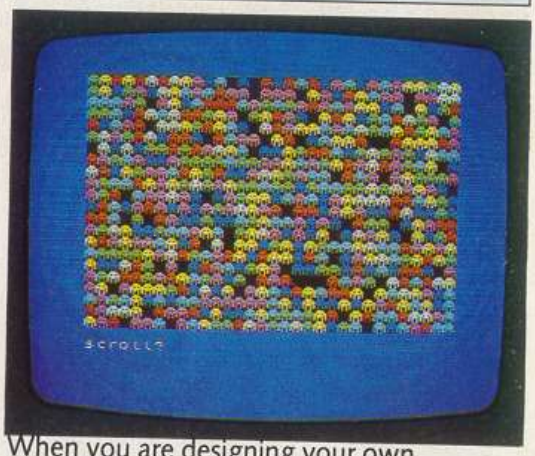
Each of the user-defined graphics characters is identified by a letter from a to u (or A to U – it makes no difference). To program the character, you enter eight POKE USR statements each ending with BIN followed by a binary number consisting of the eight Øs and 1s in each row of the grid. Let's call the spider character s.

```
10 POKE USR "s",BIN 00111100
11 POKE USR "s"+1,BIN 01111110
12 POKE USR "s"+2,BIN 11011011
13 POKE USR "s"+3,BIN 11111111
14 POKE USR "s"+4,BIN 10111101
15 POKE USR "s"+5,BIN 10100101
16 POKE USR "s"+6,BIN 10100101
17 POKE USR "s"+7,BIN 00100100
```

Now run this program and then press GRAPH and S. Instead of an S, the spider appears! Next add the following lines, obtaining the spider in line 30 in this way, and run the program. Spiders appear across the whole of the screen.

SPIDERS

```
20 BORDER 1: PAPER 0: CLS
30 PRINT INK RND*7;"s";
40 GO TO 30
```



When you are designing your own characters, remember that you won't be able to see them on the screen until you have run a program that defines them. Until then they will just appear in listings as letters.

How to 'mix' colours with speckled squares

You can quite easily simulate mixed colours on your Spectrum. To do this you need to create a character which when printed gives a square that is 50 percent ink colour and 50 percent paper colour.

```
10 FOR X=0 TO 6 STEP 2
20 POKE USR "a"+x,BIN 10101010
30 POKE USR "a"+x+1,BIN 01010101
40 NEXT x
```

All you have to do is define two pixel lines and then instruct the computer to use them alternately in a character.
When you run the program, you should see a speckled square. If you use the same technique in a program that contains colour keywords, the speckled square will produce a colour that is a mixture of the program's paper and ink colours.

Simplifying graphics with READ and DATA

There is an easier way of creating your computer graphics and that is to use decimal numbers with READ and DATA. First change the eight binary numbers made up of 0s and 1s into decimal numbers. Do this by entering PRINT BIN followed by the number, for example

PRINT BIN 00111100

The Spectrum displays 60, the decimal equivalent of 00111100. In the case of the spider, the eight decimal numbers are 60, 126, 219, 255, 189, 165, 165 and 36.

Now you can use READ and DATA. These two keywords provide you with an easy way of feeding lots of values such as numbers into the variables in a program. READ is followed by a variable – one or more letters if you are dealing with numbers, or a single letter followed by $ if you are dealing with strings. As you want to READ decimal numbers here, you need a numeric variable. Call it y.

When your Spectrum encounters READ, it looks at the first DATA statement in the program. This statement contains a list of values separated by commas. The computer takes the first value from the list and this is given to the variable after READ. The next time that the computer gets to READ, the second value is given to the variable and so on, strictly in order.

Here is the new program that will produce the spider.

```
 10 FOR x=0 TO 7
 20 READ y
 30 POKE USR "s"+x,y
 40 NEXT x
 50 DATA 60,126,219,255,189,165
,165,36
```

The program will actually store any eight decimal numbers in the memory to create a character. Just change s in line 30 to the letter you want, and after DATA in line 50, key in the eight numbers *each separated by a comma*. Press GRAPH and the letter to get the character after you have run the program.

Drawing a chessboard

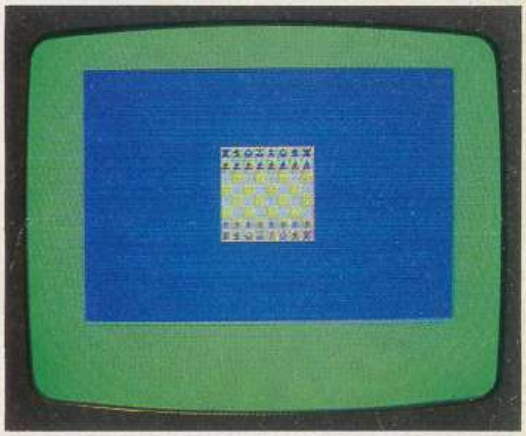
Here is a program that displays a chessboard on the screen and then lays out the pieces ready for a game. You can make colours appear in a listing by using colour control codes – see the panel below.

CHESSBOARD

```
 10 FOR x=1 TO 6
 20 READ a$
 30 GO SUB 500
 40 NEXT x
 50 BORDER 4: PAPER 1: CLS
 60 FOR l=7 TO 14 STEP 2: FOR c
=12 TO 19 STEP 2
 70 PRINT AT l,c;" ";AT l+1,c;
"  "
 80 NEXT c: NEXT l
 90 PRINT AT 7,12;"RNBQKBNR"
100 PRINT AT 8,12;"PPPPPPPP"
110 PRINT AT 13,12;"pppppppp"
120 PRINT AT 14,12;"rnbqkbnr"
130 GO TO 130
500 FOR n=0 TO 7
510 READ y
520 POKE USR a$+n,y
530 NEXT n
540 RETURN
```

The pieces are defined by a subroutine which begins at line 500.

```
 550 DATA "p",0,0,16,56,56,16,12
4,0
 560 DATA "r",0,84,124,56,56,124
,124,0
 570 DATA "n",0,16,56,120,24,56,
124,0
 580 DATA "b",0,16,40,68,108,56,
124,0
 590 DATA "k",0,16,56,16,56,68,5
6,0
 600 DATA "q",0,84,40,16,108,124
,124,0
```



Using colour control codes

Instead of using keywords such as INK and PAPER, you can put control codes in PRINT statements after the first quote mark. The characters to be displayed then change colour in the listing itself and will also appear in these colours on the screen.

To get the codes, first press EXTEND MODE and then a number key with or without CAPS SHIFT. This chart shows you how you can select any of the colours.

PRESS	1	2	3	4	5	6	7	8	9	0
EXTEND MODE + CAPS SHIFT	BLUE INK	RED INK	MGNTA INK	GREEN INK	CYAN INK	YELLW INK	WHITE INK	FLASH OFF	FLASH ON	BLACK INK
EXTEND MODE	BLUE PAPER	RED PAPER	MGNTA PAPER	GREEN PAPER	CYAN PAPER	YELLW PAPER	WHITE PAPER	BRIGHT OFF	BRIGHT ON	BLACK PAPER

ANIMATION

Computer graphics look their best when the characters or lines move about the screen, and producing animation on your Spectrum is not difficult. All you have to do is to keep on changing the position at which a character is printed or a line drawn. The best way to make this happen is to use one or more FOR NEXT loops.

Vertical and horizontal motion

Key in and run this program. If you have *not* reset or switched off your Spectrum since producing the spider graphics character on page 32, then don't bother to enter lines 10 to 50. The graphics character will still be in the memory under "s".

FALLING SPIDER

```
  5 BORDER 3: PAPER 5: CLS
 10 FOR x=0 TO 7
 20 READ y
 30 POKE USR "s"+x,y
 40 NEXT x
 50 DATA 60,126,219,255,189,165
,165,36
 60 FOR x=0 TO 7
 70 READ y
 80 POKE USR "t"+x,y
 90 NEXT x
100 DATA 16,16,16,16,16,16,16,1
6
110 FOR l=0 TO 20
120 PRINT AT l,3; INK 0;"|"
130 PRINT AT l+1,3; INK 2;"🕷"
140 NEXT l
```

You'll see the spider fall down the screen on its thread every time you run the program.

In the program, lines 60 to 100 produce another graphics character ("t") for the thread. The animation occurs in lines 110 to 140, which make up a FOR NEXT loop in which l (the line number) changes from 0 to 20. Each time the loop repeats, a length of thread is printed at one position and the spider is printed at the next position below. The next time round, the spider is replaced by another length of thread and again appears below. In this way, the spider rapidly descends on its thread until it reaches line 21, which is the bottom of the display area.

Your Spectrum can calculate the new positions very fast, so the spider drops quickly. To slow down the action, insert this line.

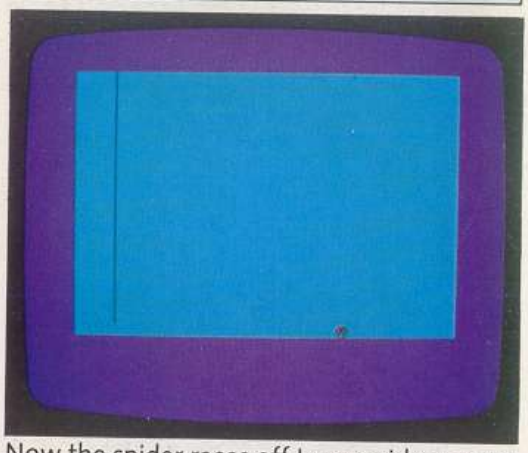
135 **PAUSE** 10

This makes the computer wait for a fifth of a second each time before printing the spider at the next position. Try changing 10 to other values and see how the speed varies.

Now add these extra lines to the program and run it again. You'll see animation but in a different direction.

SCUTTLING SPIDER

```
150 FOR c=3 TO 30
160 PRINT AT 21,c;" "
170 PRINT AT 21,c+1; INK 2;"🕷"
180 NEXT c
```



Now the spider races off to one side as soon as it touches down. The extra lines form another FOR NEXT loop that controls the column position c. Note that a space is printed first and then the spider is displayed at the next column position. This causes the spider to disappear from one position and appear at the next, moving it to the right. It is always better to delete a character before printing it at the next position, rather than to print the character first and then delete it at the old position. This helps to avoid or reduce flicker in moving graphics.

Target practice

In many computer games, action often occurs when two moving shapes collide or an object is hit by a beam. How does the computer know when a crash or explosion should happen?

Detecting collisions is not difficult. If two characters are printed at position l,c (for line and column) and position v,h (for vertical and horizontal), then if l=v and c=h, they must be at the same position. You can write this as a statement, for example

160 **IF** l=v **AND** c=h **THEN PRINT** "CRASH!"

Another way of checking for collisions is to use colour. Remove the spider program by entering NEW. Then enter the complete Pyramids program on page 31, or load the program if you have saved it on tape. You can now upgrade it and combine it with your spider (still in the memory unless you have

reset or switched off) to produce another program.

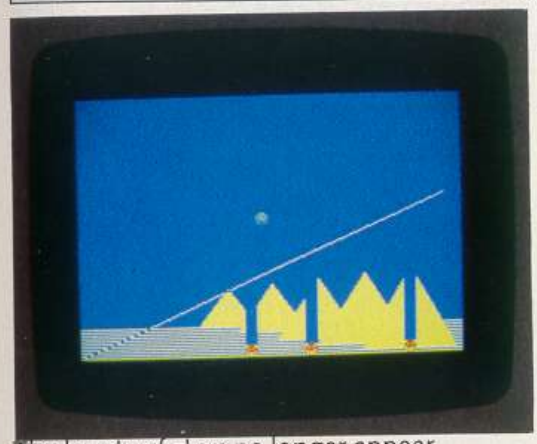
First add the following lines, which create an explosion graphic called "e".

```
    5 FOR x=0 TO 7
    6 READ y
    7 POKE USR "e"+x,y
    8 NEXT x
    9 DATA 145,82,44,121,158,52,7
4,137
```

Now delete line 190 and add or change the following lines.

SPIDERS AND PYRAMIDS

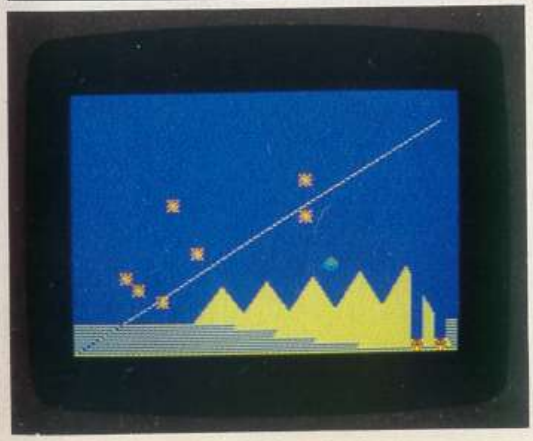
```
  114 LET h=RND*31
  115 FOR v=0 TO 20
  117 PRINT AT v,h;" ";AT v+1,h;
INK 4;"S"
  200 NEXT v
  205 PRINT AT 21,h; FLASH 1; INK
 2; PAPER 6;"S"
  210 GO TO 114
```



The bursts of stars no longer appear. Instead spiders fall through the sky and eat away the pyramids and ground. What you have done is to add a FOR NEXT loop in which v and h give the position of the spider. The variable h is random, so the spiders start their vertical fall at different places across the screen. Next add these lines.

EXPLODING SPIDERS

```
  190 IF ATTR (v+1,h)=14 THEN GO
TO 500
  500 PRINT AT v+1,h; FLASH 1; PA
PER 2;"E"
  510 PAUSE 100
  520 GO TO 114
```



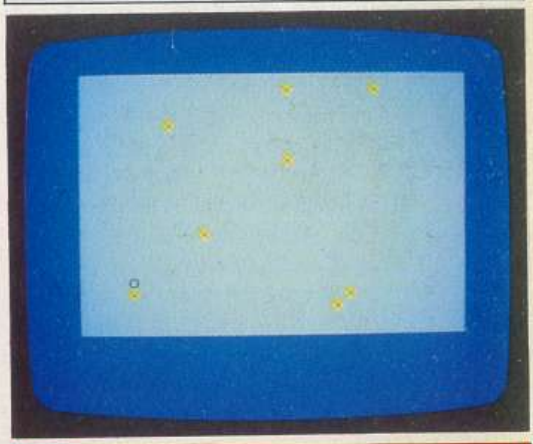
When the laser beams hit the spiders, they briefly turn yellow. When the line produced by DRAW enters a spider's character position, the ink colour changes to the same colour as the line, which is yellow. In line 190, ATTR detects if the spider goes yellow and sends the computer to the explosion subroutine at line 500.

Bouncing a ball

Many graphics programs feature shapes that bounce off the edges of the screen. This program shows you how it is done. The variables v and h work in the same way as in the exploding spiders program, but +1 or −1 is added to v or h to make the ball go down or up, and right or left. SCREEN$ checks whether there is an X at position v,h.

BOUNCING BALL

```
   10 BORDER 1
   20 FOR z=1 TO 10
   30 LET h=INT (RND*25): LET v=I
NT (RND*21)
   40 PRINT INK 2; PAPER 6; FLASH
 1;AT v,h;"X"
   50 NEXT z
   60 LET x=1: LET y=1
   70 PRINT AT v,h;" "
   80 LET v=v+y: LET h=h+x
   90 IF h=0 OR h=31 THEN LET x=-
x: BEEP .2,24
  100 IF v=0 OR v=21 THEN LET y=-
y: BEEP .2,12
  110 IF SCREEN$ (v,h)="X" THEN P
RINT INK 1; PAPER 5;AT v,h;"!*!*
!": STOP
  120 PRINT AT v,h;"O"
  130 PAUSE 2
  140 GO TO 70
```



Using attributes

The keyword ATTR detects the 'attributes' at a particular position on the screen. The attributes are the ink and paper colours and whether or not the position is flashing or bright. In the Exploding Spiders program, ATTR ensures that the spider is destroyed if it goes yellow. This is then its ink colour (number 6). The paper colour is blue (number 1) and the spider is not bright or flashing. This means that its attributes total 14. The ATTR entry in the Programmer's Reference Guide will show you how these numbers are arrived at.

HOW TO MAKE MUSIC AND SOUND EFFECTS

The ZX Spectrum + possesses a sound synthesizer that can make your programs spring to life with a great variety of musical sounds and special sound effects. It is simple to use, even if you have little or no knowledge of music. The synthesizer produces a sound signal that goes to the Spectrum's internal loudspeaker.

Programming sounds

To produce sound on your Spectrum, you use only one keyword – BEEP. It is followed by two numbers or variables representing numbers. The first tells the computer how long (in seconds) to make the sound last, and the second informs it how high or low the sound is in pitch. Pitch is measured in semitones. The pitch values are 0 for middle C, 1 for C#, −1 for B (Cb) and so on.

To hear the complete range of sounds that your Spectrum can produce, run this program.

```
10 FOR p=69 TO -60 STEP -1
20 BEEP 0.2,p
30 PRINT AT 0,0;"   ";AT 0,0;p
40 NEXT p
```

The Spectrum runs through its complete range of notes from the highest pitch (69) to the lowest (−60). You'll find that the highest notes are almost inaudible and the lowest sound like clicks. This is because these notes extend beyond the range of the human ear.

The chart at the bottom of this page shows the pitch values of a range of notes so that with it you can transform a piece of sheet music into a Spectrum program.

Sound effects

You can get all kinds of sound effects out of your Spectrum, usually by placing BEEP inside a loop that rapidly changes the pitch value. Try these programs and experiment with them to develop your own sounds. Note that the duration values are very short, being as little as a hundredth of a second. Press BREAK to end the looped programs.

BUBBLING

```
10 LET p=INT (RND*40)-30
20 BEEP 0.05,p: BEEP 0.05,p+7:
BEEP 0.05,p+4
30 GO TO 10
```

This program plays a group of three notes over and over again at random pitches. The pitch range is wide, but you can change the values in line 10 to alter the range.

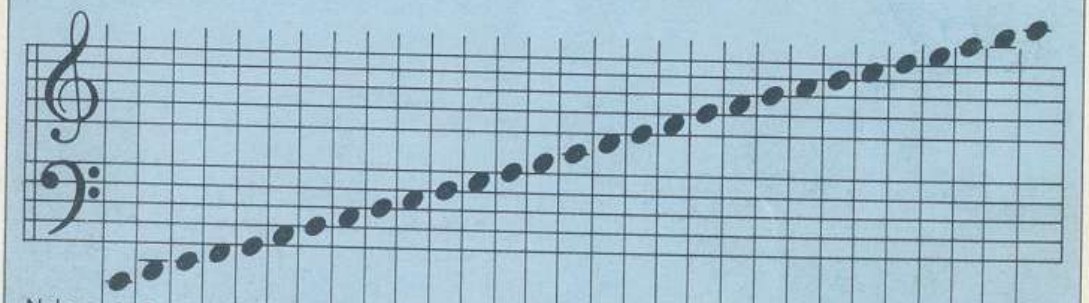
MACHINE

```
10 FOR x=12 TO 36
20 BEEP .01,x
30 BEEP .01,24-x
40 NEXT x
50 GO TO 10
```

This program produces two sounds, one going up in pitch as the other goes down. This is because the two BEEP statements make two notes sound over and over again only a hundredth of a second apart at different pitches.

Pitch values for making music

Here are the Spectrum's pitch values from the bottom to the top of the bass and treble staves. Add 1 to pitch value for a sharp note; deduct 1 for a flat note.



Note	C	D	E	F	G	A	B	C	D	E	F	G	A	B	C	D	E	F	G	A	B	C	D	E	F	G	A	B	C
Pitch value	-24	-22	-20	-19	-17	-15	-13	-12	-10	-8	-7	-5	-3	-1	0	2	4	5	7	9	11	12	14	16	17	19	21	23	24

LIFTOFF

```
10 FOR p=1 TO 48 STEP 0.2
20 BEEP .01,p: BEEP .01,p-6
30 NEXT p
```

This program is similar to the machine program, but now the two notes go up together six semitones apart. In addition, the pitch values change by 0.2 – a fifth of a semitone – each time. This makes the sound rise slowly in pitch. Try other small pitch changes by changing the STEP value.

KEYBOARD CONVERTER

```
10 LET p=CODE INKEY$
15 IF p=0 THEN GO TO 10
20 BEEP .04,(p-30)/2
30 GO TO 10
```

This program waits for you to press any key. When you do, each one gives out a different sound. Note that pressing CAPS SHIFT while holding down another key makes the sound go lower. This program works because CODE INKEY$ gives a different value to p every time a new key is pressed. The second line stops the computer from making a sound if no key is pressed. You can see the values that CODE returns in the character set table on page 51.

Sound and vision

The sound effects that your Spectrum can produce go best with action on the screen. To demonstrate how you can add sound to programs, return to the complete scuttling spider program on page 34.

Remember that you inserted a PAUSE statement at line 135 to slow down the action. Instead of delaying the program in this way, you can program a pause that produces sound. Change line 135 to

```
135 GOSUB 500
```

Now add the following lines to the program.

```
200 STOP
500 FOR p=40-l TO 38-l STEP -1
510 BEEP 0.02,p
520 NEXT p
530 RETURN
```

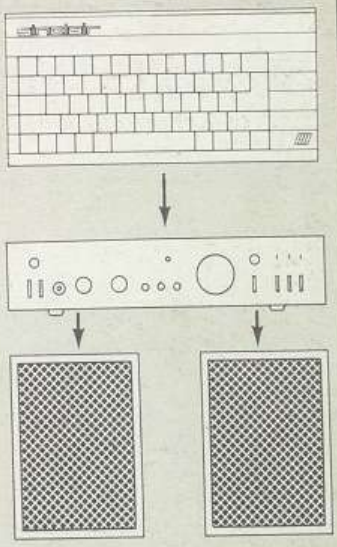
Run the program and the spider now makes a warbling sound as it falls. The subroutine plays three notes very quickly that get lower every time the spider descends to the next position on the screen. Try adding more notes by changing line 500 and speeding up or slowing down the notes by changing 0.02 in line 510.

How to amplify your Spectrum

To make the sound of your Spectrum louder, you can connect either the EAR socket to headphones or to an amplifier and loudspeaker. The amplifier will have a volume control that you can adjust to make the sounds as loud as you want.

The simplest way to do this is to use the Spectrum cassette lead to connect the EAR socket to the MIC socket of a cassette player. Take out the cassette if necessary, switch on the cassette player and then press PLAY, REWIND (REVERSE) or FAST FORWARD (CUE). Adjust the cassette player's volume control and you should hear the computer's sound coming from the loudspeaker in the player. Alternatively, you can connect headphones with the cassette player if you want.

You can also connect your Spectrum to a hi-fi or music centre if you want a really full sound. You will need a special lead with a 3.5mm jack plug to fit the Spectrum and a plug to insert into the input socket of the hi-fi amplifier or the music centre. The Spectrum produces a *line* signal similar to that output by cassette decks and tape recorders, so the REPLAY or LINE IN socket on the amplifier should work. If you have any problems, consult a shop that sells sound equipment.



HOW TO SAVE YOUR OWN PROGRAMS

Before long, you'll want to store your own programs on cassette tape. To do this, you connect a cassette player to your Spectrum and *save* the program that is in the computer. The Spectrum sends the program to the cassette player in a form that it can record on tape. Then whenever you need to use the program, you *load* it from the cassette tape back into the computer using the loading procedure described on pages 14–15. On these two pages, you can see how to save programs and also how to check that the program has been saved correctly.

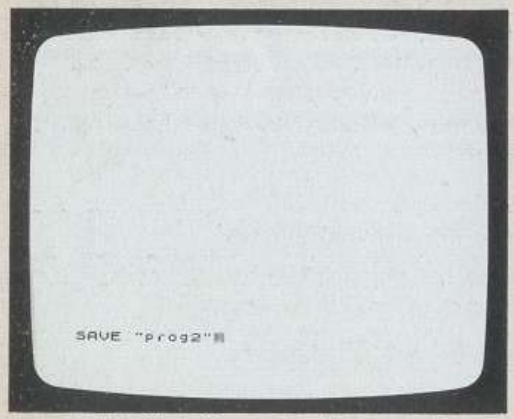
Recording your own programs

1 First connect your Spectrum to a suitable cassette player using the cassette lead as described on page 14, but make sure that *only* the Spectrum's MIC socket is connected to the cassette player.

2 If the cassette player has a record level or volume control, adjust it to about two-thirds maximum. If not, don't worry as the recording level will be set automatically.

3 Key in SAVE followed by the name of the program in quote marks, for example

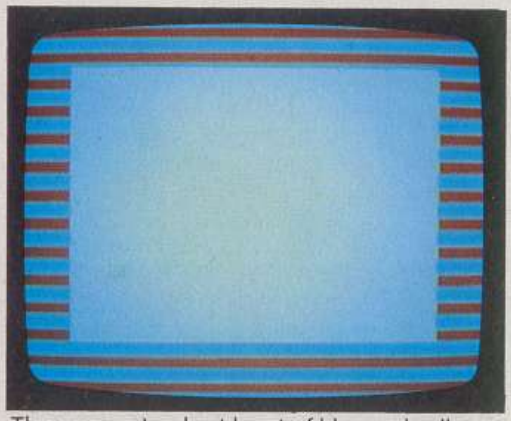
SAVE "prog2"



Any combination of up to ten letters and numbers can be used. Now press ENTER. The SAVE line will disappear and then you will see the cassette operating instruction from the Spectrum.

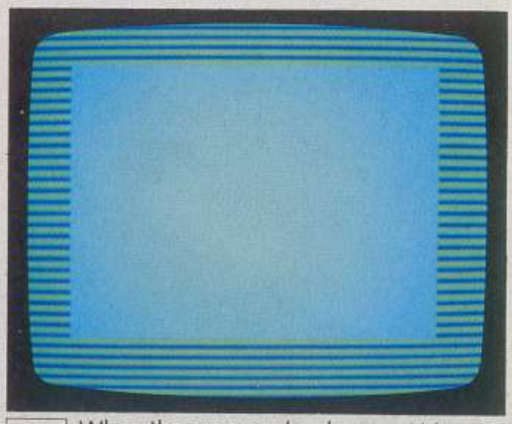
4 Set the cassette player to record, usually by pressing RECORD and PLAY together. Then press any key on the Spectrum.

5 Now wait as your Spectrum saves the program. First you should see blue and red bands moving slowly up the screen.



Then you get a short burst of blue and yellow stripes. This happens as the Spectrum sends the name of the program to the tape.

6 Next comes a short gap and then more blue and red bands. This is followed by the blue and yellow stripes again as the Spectrum now sends the program to the tape. A long program may take some time.

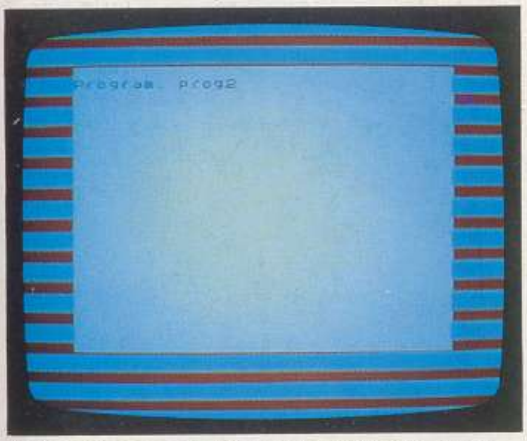


7 When the program has been sent to the tape, the report **0 OK, 0:1** appears. Stop the tape. The program has now been saved. If you want, you can now check or 'verify' it.

Now verify your program

Although the computer has sent the program to the cassette player, you cannot be sure that the program has been successfully recorded on the tape. Fortunately, your Spectrum is able to check this for you.

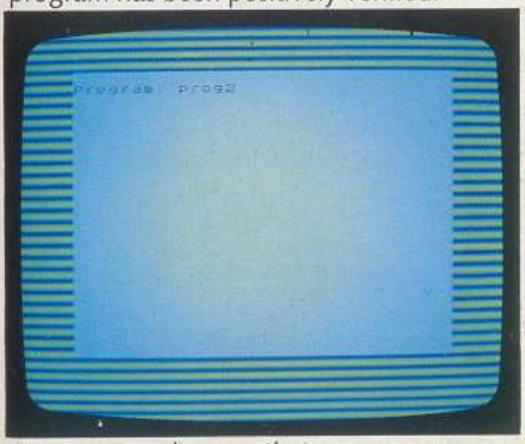
This procedure is called verification. First rewind the tape to the start of the program, then connect the Spectrum EAR socket to the cassette EAR socket. (You can leave the MIC sockets connected.) Next key in VERIFY followed by the program name in quotes. Then press ENTER and start the tape. The same sequence of blue and red bands and blue and yellow stripes should be seen. The program name will appear and remain until verification is completed.



When the second blue and yellow section shown below ends the report

0 OK, 0:1

should appear. This means that your Spectrum has checked the program on the tape with the program in its memory and found that they are exactly the same. The program has been positively verified.



You can now be sure that your program is safely on tape.

Software saving tips

1. Write the name of a program on the cassette label or card when you save it. Use the same capital or small letters as appear on the screen. If the cassette player has a counter, use it to locate the program and write the counter number by the name.

2. Before saving, place the program name in the program by using a REM statement, for example

5 **REM** SPIDER program Version 3

The computer ignores all REM statements when the program runs, and you can use REM to put remarks and reminders in the program wherever you like.

If you do not get this report, then something has gone wrong. First check the **Software loading troubleshooter** on page 16 as the fault could be that the program is safe on tape but is not loading back into the computer for verification. If you find something wrong here, correct the fault, rewind the tape and verify the program again. If the computer still does not verify the program, consult the **Software saving troubleshooter** on the next page. Don't press NEW, reset or turn off the computer because you will then lose the program in memory, without having a reliable copy on tape.

Automatic program start

You can follow SAVE by the program name and then LINE 1, for example

SAVE "SPIDER" **LINE 1**

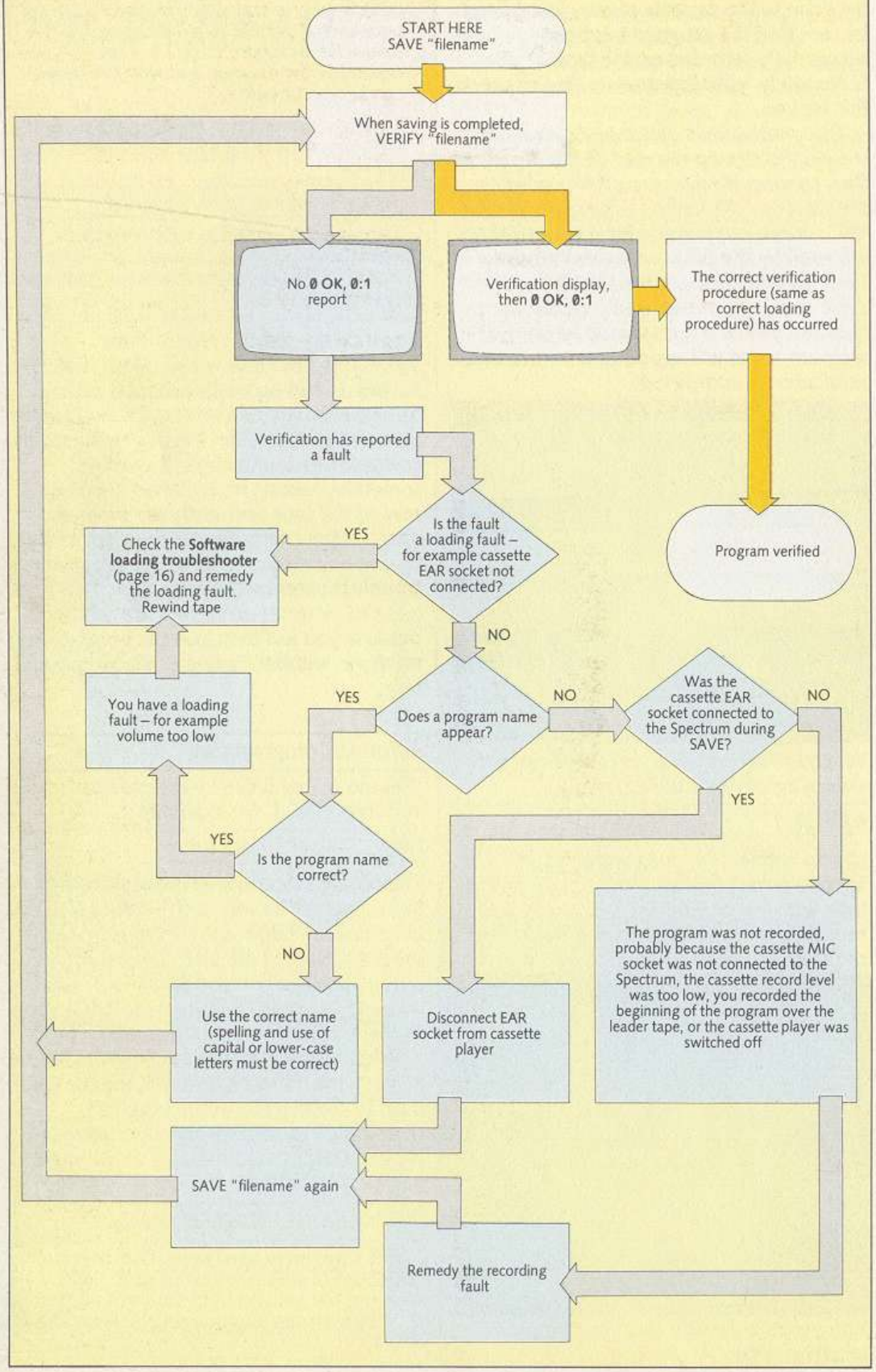
The saving procedure is no different from before but when you verify, do *not* include LINE 1 after VERIFY and the program name. Programs saved with LINE 1 will start automatically when you load them. There is no need to use RUN (but remember to stop the tape when the program starts).

What happens is that the program begins at line 1 and if there is no line 1, the computer skips to the first line in the program. Changing 1 to another number makes the program begin automatically at the line having this number.

Saving CODE, SCREEN$ and DATA

SAVE may also be used with CODE or SCREEN$ to store a section of Spectrum's memory and with DATA to store an array. See the entries in the Programmer's Reference Guide.

Software saving troubleshooter



LEARN ABOUT YOUR ZX SPECTRUM+

This chapter takes you inside your ZX Spectrum +, explaining how the various components beneath the keyboard work and how they link together to make the computer function. It also shows you how you can use 'peripherals' – add-on devices that let you upgrade your Spectrum into a full computer system. Finally, you'll find out here more about the technical side of your computer – including the way in which memory is organized up, together with the Spectrum's technical specification.

WHAT'S INSIDE?

Read on to find out – and do *not* try to open up your ZX Spectrum + in order to find out how it works. You will invalidate your guarantee if you do and you could do serious damage.

Inside the case are two ribbon connectors that link the keyboard to the rest of the Spectrum's components. These are all mounted on a single printed circuit board. The board carries standard electrical components such as resistors and capacitors, but the most prominent items are the black rectangular microchips, arranged either singly or in blocks.

Inside a chip

The functioning part of a microchip is actually much smaller than the plastic package that contains it. The casing is primarily designed to support all the connections that the chip requires, allowing it to be plugged into sockets on the circuit board. The chip itself is a thin silicon wafer that contains many thousands of electrical junctions. Each junction acts as a switch to stop, pass or store electric signals reaching it. Although this is a simple procedure, there are so many junctions acting together that they can produce signals that store or process information with astonishing speed and accuracy. The ZX Spectrum + contains a number of different chips, each designed to play a particular part in the running of the computer.

How the chips link up

So overall, your Spectrum is an electric circuit of enormous complexity. Code signals consisting of pulses of electricity constantly flash along the pathways inside and between the chips and components to make the computer work.

How is everything kept in order so that the right signal arrives in the right place at the right time? Hidden away inside one of the chips is the computer's clock. It ticks by emitting pulses of electricity – 3.5 million of them every second. These pulses move regularly through the circuits to produce the code signals that control the action of each part and keep everything in step.

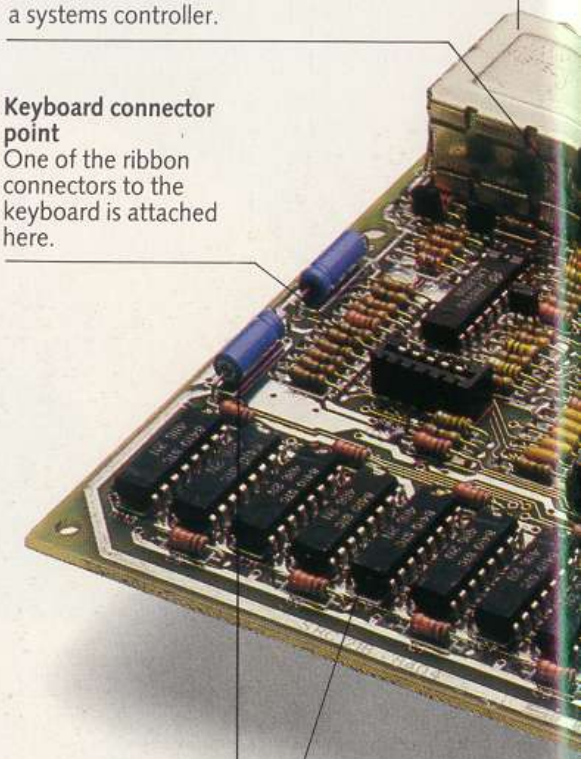
The interior of your ZX Spectrum +
In this view of the Spectrum's circuit board, the two ribbon connectors to the keyboard have been removed.

When the Spectrum is in use, pressing a key brings a pair of wires under the keyboard into contact. This sends a code signal to the CPU.

Uncommitted Logic Array (ULA)
This chip generates the display from information held in RAM and also acts as a systems controller.

TV output
This produces the signal that goes to the television set.

Keyboard connector point
One of the ribbon connectors to the keyboard is attached here.



TV encoder
This changes signals produced by the computer's circuits into colour television signals.

Random Access Memory (RAM)
These chips contain the program that is fed into the computer and any particular information needed by the program, such as values held by variables. The contents of the 48K of RAM can be changed from the keyboard, and can be erased altogether by resetting or turning off the computer.

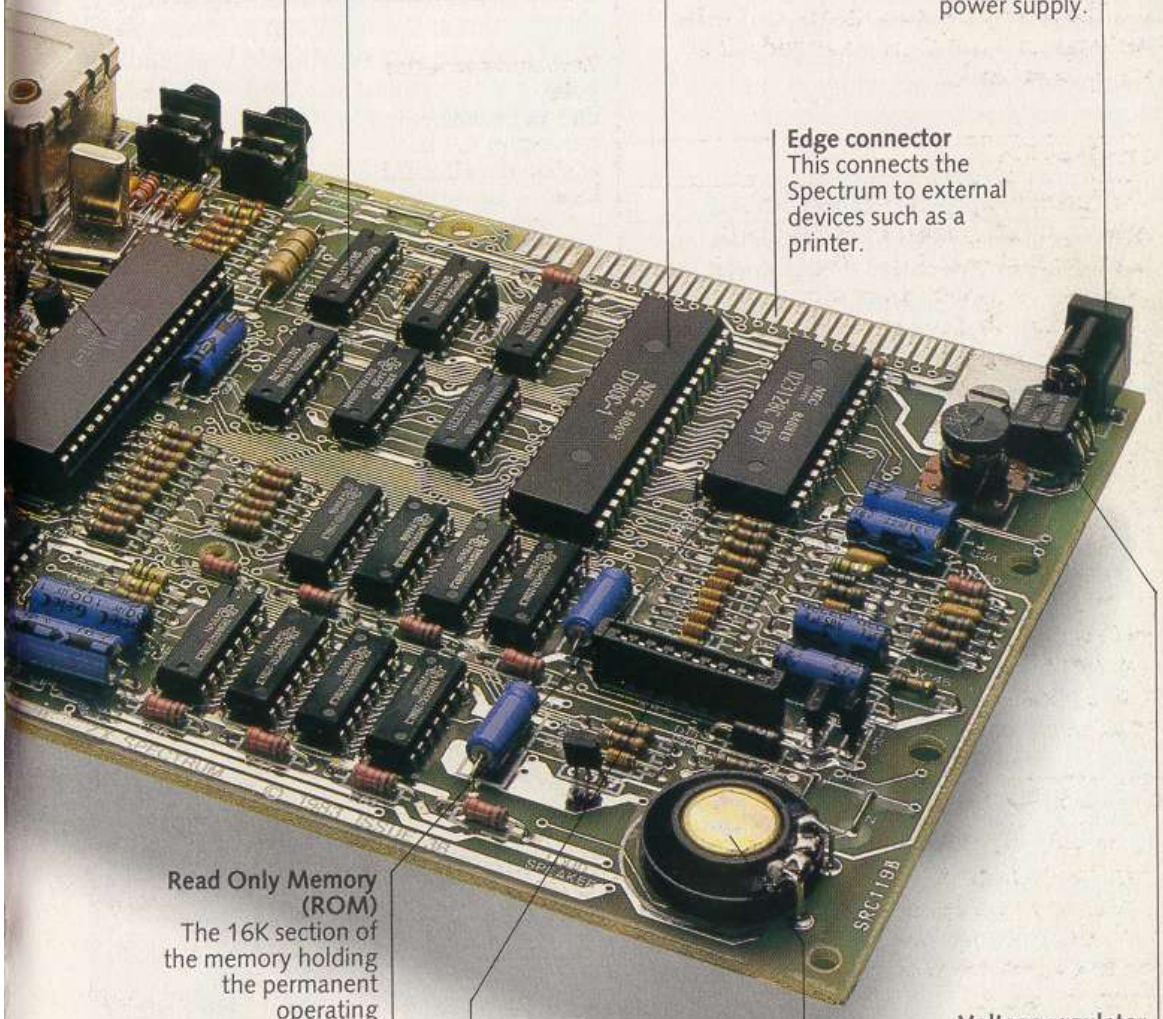
Cassette sockets
These are used to send information and programs from the memory to a tape and to feed them back into the memory from the tape.

Logic chips
These chips act as an interface in the exchange of information between the CPU and the RAM.

Central Processing Unit (CPU)
The 'brains' of the computer. The CPU is a Z80 microprocessor. It carries out all the computing calculations and controls the overall operation of the Spectrum.

9 VDC socket
This connects to the power supply.

Edge connector
This connects the Spectrum to external devices such as a printer.



Read Only Memory (ROM)
The 16K section of the memory holding the permanent operating instructions needed by the CPU. Among other things, these intructions convert BASIC programs into a form that the CPU can understand. The contents of these memory chips cannot be altered from the keyboard.

Keyboard connector point
One of the ribbon connectors to the keyboard is attached here.

Loudspeaker
This produces sound when required.

Voltage regulator
This component prevents any changes in voltage from affecting the computer.

HOW DOES YOUR ZX SPECTRUM+ WORK?

The operating ZX Spectrum +, like other microcomputer systems, consists of four main parts. These are the *input units*, such as the keyboard, which put information or a program into the computer; the temporary and permanent *memories*, which store information, programs and operating instructions; the *Central Processing Unit* CPU, which carries out the program instructions on the information, and the *output units*, which give the result.

Entering and running a program

What happens inside the Spectrum when you enter and run a very simple program? Here's a one-line example

```
10 PRINT 6+2
```

First, you operate the keyboard. Beneath the keys is a grid of criss-cross wires. Every time you press a key, a pair of wires makes contact and sends a code signal to the CPU. The CPU in turn sends the code to RAM, where it is stored.

When you run the program, the CPU takes the stored codes from RAM one by one in the order of the program. It first receives the code for PRINT, which tells it to get a particular operating code from ROM. This operating code goes to the CPU and the CPU gets ready to perform the actions that display a value on the screen. The CPU next gets the value of 6 from RAM. This too is in the form of a code, and the CPU stores it in a small internal memory called a register. Next comes the code for addition, and the CPU again gets the necessary operating code from ROM. Finally, the CPU takes the code for 2 from RAM. It adds this code to the value in the register to get the result (8). The CPU then converts the result into another set of codes and sends them to the display file. This is the section of RAM that holds codes for everything you see on the screen, and the number 8 appears on the screen.

Storing a program

If you ask the Spectrum to save the program on tape, the CPU again takes the codes from RAM. But instead of acting on them, it sends the codes to a converter unit which changes them into sound signals. These signals are then sent to the cassette player and recorded on tape.

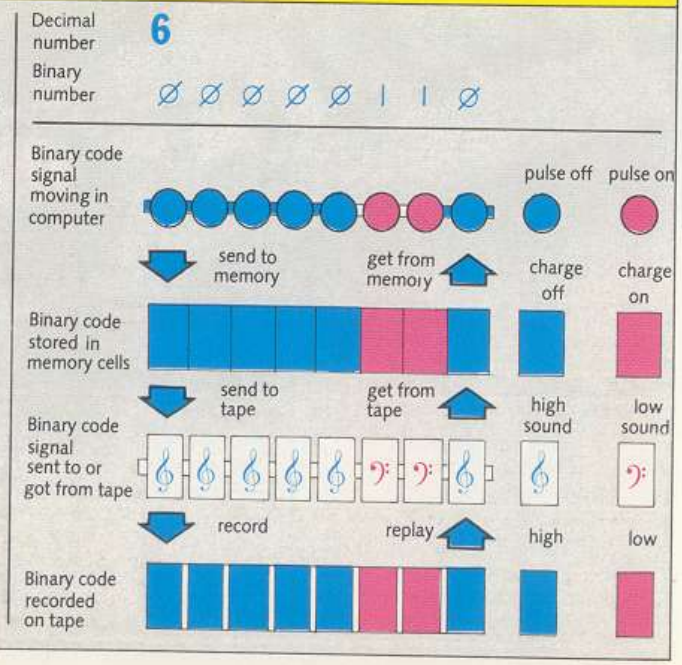
When you load the program later, the sound signals from the cassette player are changed back into computer codes by the converter. The CPU sends them back to RAM, where they are stored until required.

Binary codes

All the codes that make your Spectrum work are in *binary* form. They are called binary because they are all composed of just two types of signal. They can be represented as binary numbers, that is numbers which contain only two numerals – 0 and 1. The binary number for 6 for example is 00000110.

Inside your Spectrum, the codes consist of sequences of rapid pulses of electricity. If a pulse arrives at any point, this represents a 1 in binary. If a pulse does not arrive within a set period, this represents a 0. In computer codes therefore, 6 is *off-off-off-off-off-on-on-off*. The Spectrum handles eight codes simultaneously.

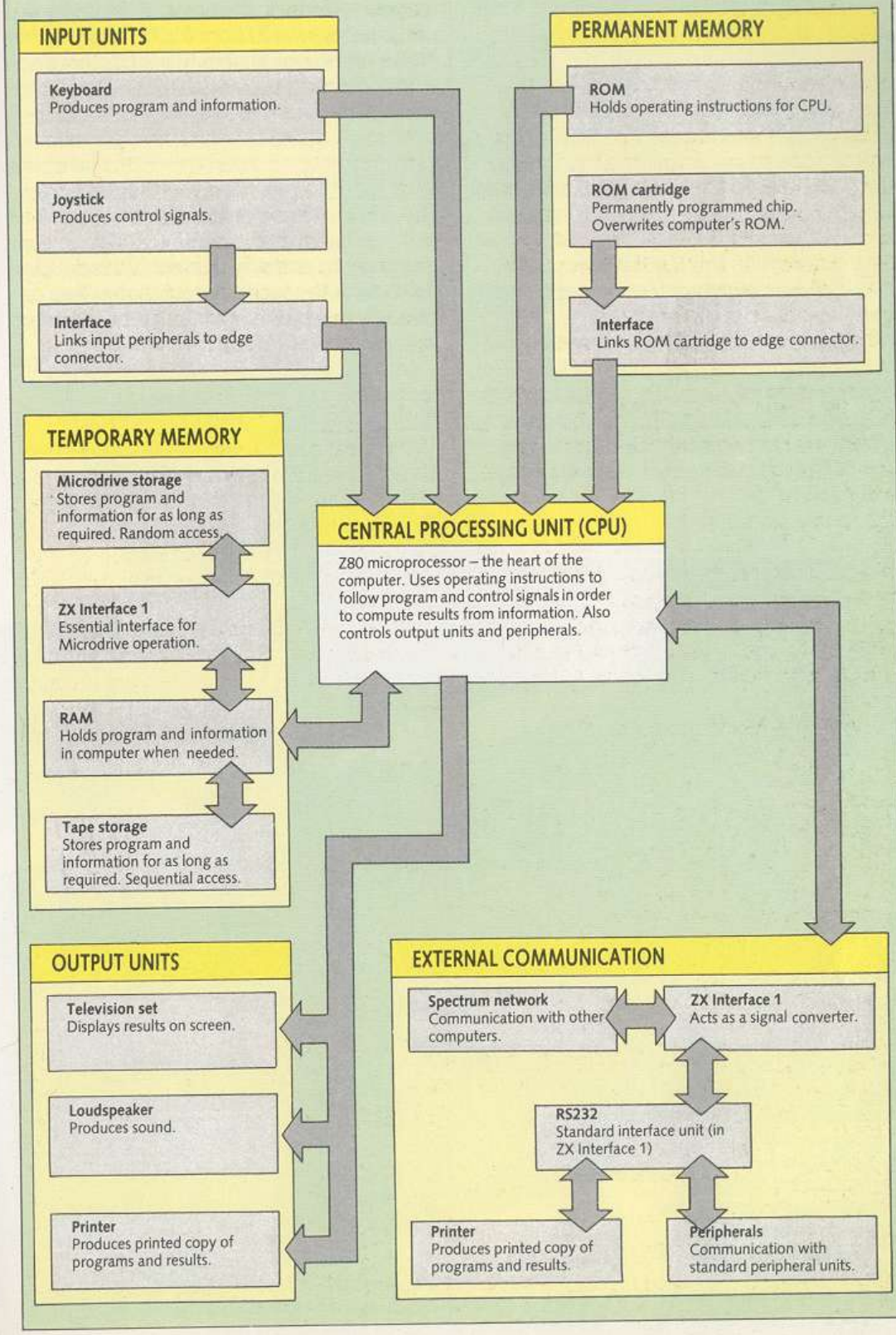
You can see in this diagram how different types of binary coding are used by the computer to move information from one place to another.



Spectrum input-to-output pathways

This diagram shows how coded information passes from input units like the keyboard, through the Spectrum's processing system, and then to output units like the television screen.

Single-headed arrows indicate pathways that operate in one direction only. Double-headed arrows indicate pathways that can operate in both directions.



HOW TO CONNECT PERIPHERALS

You can upgrade your ZX Spectrum + into a complete and powerful computer system by using Sinclair and other Spectrum-compatible peripherals. Central to this system is the ZX Interface 1, which enables you to connect Microdrives for fast and simple handling of programs and data, and which also connects with a wide range of different peripherals, including other Spectrums. With this interface, you can connect your Spectrum to standard printers, and hook the computer up to a modem, allowing you to transmit and receive programs and data via telephone lines. Other interfaces are available that link plug-in ROM cartridges to the computer for instant program loading. Through these, you can also attach joysticks to make games easier to control.

Spectrum-compatible printers

Some printers plug directly into the Spectrum's edge connector. If you already have a Sinclair ZX printer, for example, you can connect it up to the computer without using an interface. This kind of printer will also plug into the back of the ZX Interface 1. However, to use printers that require an RS232 output, you must use the D socket on the ZX Interface 1.

ZX Interface 1

The ZX Interface 1 unit is attached to the rear and base of the Spectrum. It links your Spectrum to as many as eight Microdrives, up to 63 other Spectrum computers and, through its RS232 standard interface unit, a vast range of standard peripherals.

Microdrives and Microdrive cartridges replace the cassette player and tapes for storing programs and data. By inserting Microdrive cartridges, you can save, verify and load programs in seconds. Each cartridge can store up to 85K of data and,

Loading a Microdrive
Microdrive cartridges are inserted into the slot in front of the drive.

Plugging in a ROM cartridge
The cartridge is inserted into the interface socket. When the computer is powered up, the program is automatically loaded, bypassing the ROM inside the computer.

Connecting peripherals
During operation, the ZX Interface 1 is plugged into the edge connector so that it is beneath and behind the computer. The illustration here shows the system before the computer is connected.

Microdrive units
Up to eight of these storage units can be attached to one Spectrum.

Ribbon cable
This connects the Microdrive to the computer via the ZX Interface 1.

Foot-note
The ZX Spectrum+ has two built-in feet which can be used to tilt the keyboard. These feet do not need to be used when a ZX Interface 1 is fitted.

using the maximum of eight Microdrives, your Spectrum will have up to 680K on-line storage capacity! Any program is located automatically with a typical access time of 3.5 seconds.

Using the network lead provided with the interface unit, you can link your computer to another one – either a ZX Spectrum or another ZX Spectrum +. This network can then be extended to a maximum of 63 other Spectrums. Information is exchanged between them at a rate of 10,000 characters a second.

The ZX Interface 1 unit also includes an RS232 interface with a 9-way D socket through which printers, other standard peripherals, modems and other computers can be linked to your Spectrum. A standard interface cable is also available.

Standard printers are connected through the ZX Interface 1.

ROM cartridges and joysticks

Interfaces like the ZX Interface 2 let you connect up ROM cartridges and joysticks. ROM cartridges load immediately on power-up to give you programs that would take a long time to load from tape.

ROM cartridge/joystick interface

Edge connector
Peripherals are linked to the computer via this connector.

Warning
Peripherals should always be connected before power is switched on.

THE ZX SPECTRUM+ MEMORY MAP

If you look at the photograph of the interior of the Spectrum on pages 42-43, you will see that there is one ROM chip, and 16 smaller RAM chips. These chips give the Spectrum its memory. The memory consists of 65536 storage units that each contain one byte (a number from Ø to 255). Each unit is identified by a number called its *address*.

ROM means *Read Only Memory*. This part of the memory contains operating instructions for the Central Processing Unit. It is a 16K ROM, meaning that it contains 16×1024 (16384) bytes or addresses. The bytes can only be read from this memory, so that they cannot be changed. (If they could, the computer would stop working.) You can obtain the byte at any address by using PEEK.

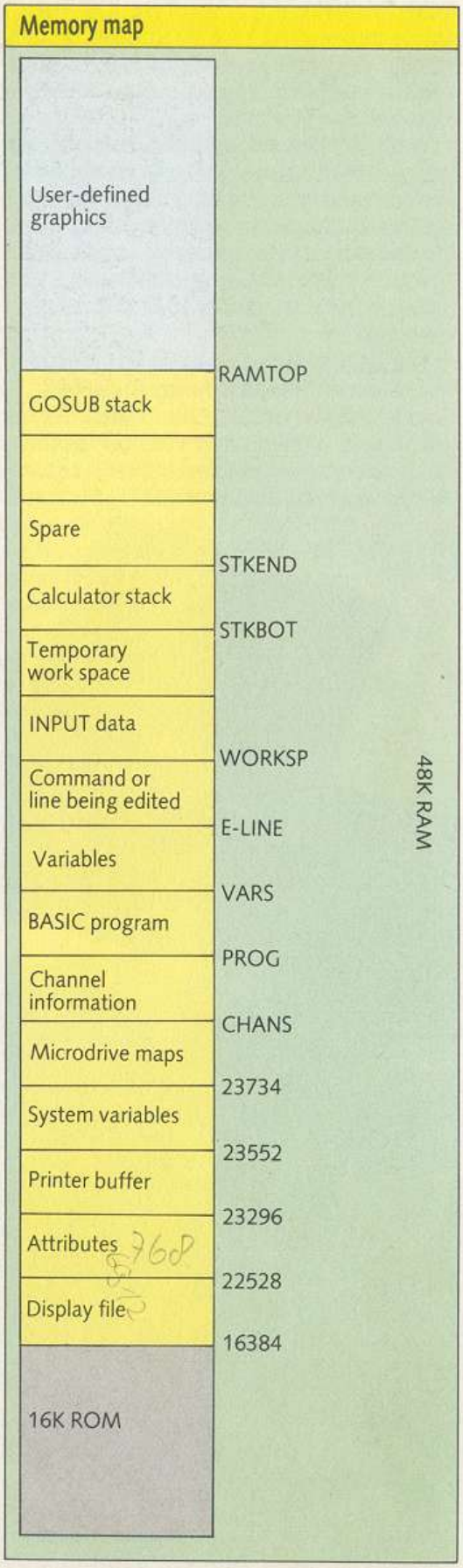
RAM means *Random Access Memory* and it contains the programs and information that are fed into the computer. The Spectrum has a 48K RAM, that is it contains 48×1024 (49152) bytes or addresses. Random access means that a byte at any address can be changed, and this can be done by using POKE.

The memory addresses extend from Ø to 65535, the first quarter being ROM and the remainder RAM.

System variables

The column opposite shows how the memory of the Spectrum is organized. On it you can see where the various sections that control the computer are located. Several of these may change position, and their boundaries are given by *system variables*.

The Spectrum's system variables are not variables like those used in BASIC. They are simply names for certain useful values that are located at particular addresses or locations in the memory. The purpose of the name is to help you remember the significance of the particular value stored at the location. For example, the system variable RAMTOP is the top address in RAM. This area of memory holds a BASIC program and the values of its variables. The address of RAMTOP is 23730.



LEARN ABOUT SINCLAIR BASIC

This chapter describes Sinclair BASIC in full. In it, you will find a summary of the way every keyword is used, and further details of how Sinclair BASIC works. The information given ranges from that needed for the simplest to the most advanced BASIC programming. This is *not* a chapter to be read through from beginning to end. Instead, it's a programmer's dictionary that will enable you to make the fullest use of the Spectrum's abilities.

PROGRAMMER'S REFERENCE GUIDE TO SINCLAIR BASIC KEYWORDS

This guide contains full descriptions of all the BASIC keywords available on the ZX Spectrum +. Each entry features

- Keyword location
- Keyword class
- Keyword purpose
- Keyword use
- Programming format

The details given for location, purpose and use are self-explanatory. Class and format are more complex, and to make the most effective use of the guide, you should first read carefully the information on this page.

Keyword classes

Keywords fall into one or more of four classes.

Command
A keyword which causes an action to occur and which can be used to form a direct command. It is carried out on being entered. Examples – RUN, LOAD.

Statement
A keyword which causes an action to occur and which can be used in a program line. It is carried out only when the program is run. Examples – DRAW, INPUT.

Function
A keyword which produces a value of some kind. It forms *part* of a command or statement. Examples – RND, INT.

Logical operator
A keyword which is used to express logic in a statement or command. It can determine or change the truth of certain conditions. The Spectrum has three logical operators – AND, OR and NOT.

Numbers and variables

Numbers
Stored to an accuracy of 9 or 10 digits. Number handling range is about 10^{38} to $4 * 10^{-39}$.

Variables accepted

Number Any length, starting with a letter. Spaces are ignored and all letters converted to lower-case letters. Capital and lower-case letters are not distinguished.

String Any single letter followed by **$**. Capital and lower-case letters are not distinguished.

Array For array variables and subscripts, see the entry on DIM.

Keyword format

The keyword format expresses the syntax of each keyword – that is, the correct combination of the keyword and other factors such as values and variables. The following abbreviations are used in giving the format.

Abbreviation	Explanation	Example
num-const	A numeric constant (a number)	24.5
num-var	A numeric variable (a variable that may contain a number)	sum
num-expr	A numeric expression (any valid combination of numeric constants, variables and keywords that gives a number)	sum*24.5 **RND***7
int-num-const int-num-var int-num-expr	A numeric constant, variable or expression whose value is rounded to the nearest integer.	
string-const	A string constant or string (any combination of characters *within quote marks*)	"ZX Spectrum +"
string-var	A string variable (a variable that may contain a string)	a$
string-expr	A string expression (any valid combination of string constants, variables and keywords that gives a string).	a$ + "ZX Spectrum +" a$ (6 **TO** 8)
letter	Any capital or lower-case letter	Y x
letter$	Any capital or lower-case letter followed by $	B$ a$
cond	A condition or sub-condition within a condition	x=10 **AND** t<10
statement	Any BASIC statement that is valid when used with another statement	**IF** t>10 **THEN STOP** **PRINT INK** 2;x
[ ]	An optional item that may be repeated	

NOTE The terms *numeric value* and *string value* are used in the text for numeric or string items respectively.

Signs in Sinclair BASIC

Sign	Location	Action/Use
$	4	String variable
'	7	Begins new line
(	8	Open bracket
)	9	Close bracket
<=	Q	Is less than or equal to
<>	W	Is not equal to
>=	E	Is greater than or equal to
<	R	Is less than
>	T	Is greater than
↑	H	Raise to power
-	J	Subtraction/negative
+	K	Addition/positive/ string concatenation

Sign	Location	Action/Use
=	L	Is equal to
:	Z	Separates statements in program line
/	V	Division
*	B	Multiplication
.	Own key	Decimal point
;	Own key	Displays at next column Separates statements within program statement
"	Own key	Open and close string
,	Own key	Displays at column Ø or 16 Separates values following keywords

ZX Spectrum+ character set

	0	1	2	3	4	5	6	7	8	9
0					**TRUE VIDEO**	**INV VIDEO**	**PRINT** comma	**EDIT**	cursor left	cursor right
10	cursor down	cursor up	**DELETE**	**ENTER**	number	GRAPHICS MODE	**INK** control	**PAPER** control	**FLASH** control	**BRIGHT** control
20	**INVERSE** control	**OVER** control	**AT** control	**TAB** control						
30			space	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	Ø	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	/	]	↑	_	£	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	©		
130										
140					GRAPHICS A	GRAPHICS B	GRAPHICS C	GRAPHICS D	GRAPHICS E	GRAPHICS F
150	GRAPHICS G	GRAPHICS H	GRAPHICS I	GRAPHICS J	GRAPHICS K	GRAPHICS L	GRAPHICS M	GRAPHICS N	GRAPHICS O	GRAPHICS P
160	GRAPHICS Q	GRAPHICS R	GRAPHICS S	GRAPHICS T	GRAPHICS U	**RND**	**INKEY$**	**PI**	**FN**	**POINT**
170	**SCREEN$**	**ATTR**	**AT**	**TAB**	**VAL$**	**CODE**	**VAL**	**LEN**	**SIN**	**COS**
180	**TAN**	**ASN**	**ACS**	**ATN**	**LN**	**EXP**	**INT**	**SQR**	**SGN**	**ABS**
190	**PEEK**	**IN**	**USR**	**STR$**	**CHR$**	**NOT**	**BIN**	**OR**	**AND**	<=
200	>=	<>	**LINE**	**THEN**	**TO**	**STEP**	**DEF FN**	**CAT**	**FORMAT**	**MOVE**
210	**ERASE**	**OPEN #**	**CLOSE #**	**MERGE**	**VERIFY**	**BEEP**	**CIRCLE**	**INK**	**PAPER**	**FLASH**
220	**BRIGHT**	**INVERSE**	**OVER**	**OUT**	**LPRINT**	**LLIST**	**STOP**	**READ**	**DATA**	**RESTORE**
230	**NEW**	**BORDER**	**CONTINUE**	**DIM**	**REM**	**FOR**	**GO TO**	**GO SUB**	**INPUT**	**LOAD**
240	**LIST**	**LET**	**PAUSE**	**NEXT**	**POKE**	**PRINT**	**PLOT**	**RUN**	**SAVE**	**RANDOMIZE**
250	**IF**	**CLS**	**DRAW**	**CLEAR**	**RETURN**	**COPY**				

ABS ABSolute value

Keyboard location
EXTEND MODE
G

Function

ABS gives the absolute magnitude of a numeric value, that is the value without a positive or negative sign.

How to use ABS
ABS is followed by a numeric value. An expression must be *enclosed in brackets*, for example

```
50 LET x=ABS (y-z)
```

ABS returns the absolute value of the numeric value.

Example
The command

```
PRINT ABS -34.2
```

displays 34.2.

Format
ABS num-const
ABS num-var
ABS (num-expr)

ACS Arc CoSine

Keyboard location
EXTEND MODE
SYMBOL SHIFT W

Function

ACS calculates the value of an angle from its cosine.

How to use ACS
ACS is followed by a numeric value. An expression must be *enclosed in brackets*, for example

```
60 LET x=ACS (y*z)
```

The value following ACS (y*z above) is the cosine of the required angle and may range from –1 to 1. ACS then returns the value of the angle in *radians*. To convert radians to degrees, multiply the value returned by ACS by 180/PI.

Example
The command

```
PRINT 180/PI * ACS 0.5
```

displays 60, the angle in degrees that has a cosine of 0.5.

Format
ACS num-const
ACS num-var
ACS (num-expr)

AND

Keyboard location
SYMBOL SHIFT Y

Logical Operator/Function

AND acts as a logical operator to test the truth of a combination of conditions. Only if *all* conditions are true is the overall combination true. AND also acts as a function to perform binary operations on two numeric or string values.

How to use AND
As a logical operator, AND links two conditions in a statement where the truth of the whole is to be tested, for example

```
90 IF x=y+z AND time<10
   THEN PRINT "Correct"
```

Only if both conditions (x=y+z and time<10) are true will the computer display Correct. If either or both conditions are false, then the whole combination is false and in this example, the program proceeds to the next line.

AND as a function
As a function, AND can operate on two numeric values, for example

```
50 LET x=y AND z
```

AND returns the first value (y) if the second (z) is not equal to 0, and returns 0 if the second value (z) is 0.

AND may also operate on a string value providing it precedes AND. A numeric value must always follow AND, for example

```
50 LET a$=b$ AND z
```

AND returns the first value (b$) if the second (z) is non-zero and a null string (" ") if the second value (z) is 0.

Note that the ZX Spectrum + assigns a value of 1 to a true condition and 0 to a false condition, and recognizes any non-zero value as true and 0 as false. It does not evaluate combinations of numeric values in accordance with standard truth tables.

Examples

```
60 LET correct=(x=y+z) AND
   time<10
70 LET score=score+10* (1
   AND correct)
80 LET a$=("Out Of Time Or
   Not " AND NOT
   correct)+"Correct"
```

If the two conditions in line 60 are true, then the numeric variable *correct* is assigned a value of 1. Then *score* is increased by 10 and a$ becomes "Correct". If either of the conditions is false, then *correct* has a value of 0; *score* is unchanged and a$ becomes "Out Of Time Or Not Correct".

Format
cond **AND** cond
num-expr **AND** num-expr
string-expr **AND** num-expr

ASN Arc SiNe

Keyboard location
EXTEND MODE
SYMBOL SHIFT Q

Function

ASN calculates the value of an angle from its sine.

How to use ASN
ASN is followed by a numeric value. An expression must be *enclosed in brackets*, for example

```
60 LET x=ASN (y*z)
```

The value following ASN (y*z above) is the sine of the required angle and it may range from –1 to 1. ASN then returns the value of the angle in *radians*. To convert radians to degrees, multiply the value returned by ASN by 180/PI.

Example
The command

```
PRINT 180/PI* ASN 0.5
```

displays 30, the angle in degrees that has a sine of 0.5.

Format
ASN num-const
ASN num-var
ASN (num-expr)

AT

Keyboard location
SYMBOL SHIFT I

See **INPUT; LPRINT; PRINT**

ATN Arc TaNgent

Keyboard location
EXTEND MODE
SYMBOL SHIFT E

Function

ATN calculates the value of an angle from its tangent.

How to use ATN
ATN is followed by a numeric value. An expression must be *enclosed in brackets*, for example

```
60 LET x=ATN (y*z)
```

The value following ATN (y*z above) is the tangent of the required angle. ATN returns the value of the angle in *radians*. To convert radians to degrees, multiply the value returned by ATN by 180/PI.

Example
The command

```
PRINT 180/PI* ATN 1
```

displays 45, the angle in degrees that has a tangent of 1.

Format
ATN num-const
ATN num-var
ATN (num-expr)

ATTR ATTRibutes

Keyboard location
EXTEND MODE
SYMBOL SHIFT

Function

ATTR gives the attributes of a specified character position on the screen. These are the ink and paper colours, brightness and flash status of the character at the position.

How to use ATTR
ATTR is followed by two numeric values *separated by a comma and enclosed in brackets*, for example

```
150 IF ATTR (v,h)=115 THEN
      GOSUB 2000
```

The first value following ATTR (v above) may range from 0 to 23 and is the line number of a position on the screen. The second value (h above) may range from 0 to 31 and is the column number of the position. ATTR then returns a number from 0 to 255. This number is the sum of the attributes at the specified position, and is made up as follows:

Ink colour	Colour code	(0 to 7)
Paper colour	8 times colour code	
Bright	64	
Flashing	128	

Example
If a character at position 11,16 is displayed in ink colour 3 (magenta), paper colour 6 (yellow) and is bright but not flashing, then the command

```
PRINT ATTR (11,16)
```

displays 115 (3 + 8x6 + 64 + 0).

ATTR in binary form
ATTR returns one byte in which bit 7 (most significant) is 1 for flashing or 0 for normal, bit 6 is 1 for bright or 0 for normal, bits 5 to 3 are the paper colour (in binary) and bits 2 to 0 are the ink colour.

Format
ATTR (num-expr, num-expr)

BEEP

Keyboard location
EXTEND MODE
SYMBOL SHIFT Z

Statement/Command

BEEP makes the loudspeaker produce a single note of a given duration and pitch.

How to use BEEP
BEEP may be used to form a statement in a program or a direct command. It is followed by two numeric values *separated by a comma*, for example

```
80 BEEP x,y
```

The first value (x) may range from 0 to 10 and defines the duration of the note in seconds. The second value (y) may range from −60 to 69 and defines the pitch of the note in semitones below middle C if negative or above middle C if positive.

Example
The command

```
BEEP 0.5,1
```

causes the note C# above middle C to sound for half a second.

Format
BEEP num-expr, num-expr

BIN BINary number

Keyboard location
EXTEND MODE
B

BIN turns a binary number into a decimal number.

How to use BIN
BIN is followed by a binary number consisting of up to sixteen 1s and 0s, for example

```
50 POKE USR "a", BIN 1010
     1010
```

BIN returns the decimal value of the binary number. It is commonly used in conjunction with POKE and USR as above for creating user-defined graphics characters, with 1 signifying a pixel of ink colour and 0 a pixel of paper colour.

Example
The command

```
PRINT BIN 11111110
```

displays 254, the decimal value of the binary number.

Format
BIN [1] [0]

BORDER

Keyboard location
B

Statement/Command

BORDER specifies the colour of the border around the screen display area.

How to use BORDER
BORDER may be used as a direct command or as a statement in a program. It is followed by a numeric value, for example

```
30 BORDER RND*7
```

The value following BORDER is rounded to the nearest integer and specifies the colour of the border as follows:

0 Black
1 Blue
2 Red
3 Magenta
4 Green
5 Cyan
6 Yellow
7 White

Note that BORDER also sets the paper colour of the lower part of the screen. Unlike INK and PAPER, a BORDER statement cannot be embedded (inserted) in a PRINT statement.

Format
BORDER int-num-expr

BRIGHT

Keyboard location
EXTEND MODE
SYMBOL SHIFT B

Statement/Command

BRIGHT causes characters to be displayed in brighter colours than normal.

How to use BRIGHT
BRIGHT may by used as a direct command but is normally used to form a statement in a program. It is followed by a numeric value, for example

```
80 BRIGHT 1
```

The value following BRIGHT is rounded to the nearest integer if necessary and may then be either 0, 1 or 8. A value of 1 causes all characters subsequently displayed by PRINT or INPUT statements to appear in a brighter ink and paper colour, and a value of 8 causes bright character positions to remain bright and normal character positions to remain normal when new characters are printed there. BRIGHT followed by 0 cancels both BRIGHT 1 and BRIGHT 8 so that all characters subseqently displayed are normal.

BRIGHT may also be embedded (inserted) within display statements formed by PRINT, INPUT, PLOT, DRAW and CIRCLE. BRIGHT follows the keyword but precedes the data or display parameters; it is followed by the same values *and a semicolon,* for example

```
50 PRINT BRIGHT 1;
   "WARNING"
```

The effect of BRIGHT is then local and applies only to the characters displayed, point plotted or line drawn by the display statement. Note that BRIGHT 1 brightens the paper colour of the *whole* character position of 8x8 pixels if any pixel in the position is plotted in an ink colour.

Format
BRIGHT int-num-expr [;]

CAT CATalogue

Microdrive file-handling command. See Microdrive and Interface 1 manual.

CHR$ CHaRacter string

Keyboard location
EXTEND MODE
U

Function

The characters and keywords available on the keyboard plus any user-defined graphics characters make up the Spectrum character set. By using CHR$ and a code number, each one can be obtained as a string. The character set also contains several control codes that affect the display of characters. These codes can be brought into operation and characters displayed by using PRINT before CHR$. The complete character set and code numbers can be found on page 51.

How to use CHR$
CHR$ is followed by a numeric value, for example

```
80 PRINT CHR$ x
```

An expression must be *enclosed in brackets.* The value following CHR$ (x above) is rounded to the nearest integer. If it is in the range 32 to 255, CHR$ returns a keyboard character, user-defined graphics character or a keyword as a string. The Spectrum uses the ASCII code for values from 32 to 95 and 97 to 126. If x is assigned a value of 65, the above statement displays A, for example.

CHR$ control codes
Values from 1 to 31 either return control codes or are not used. CHR$ 6 (PRINT comma), 8 (back-space) and 13 (new line or ENTER) affect displays on the screen if included in a PRINT statement. CHR$ may be followed by the code value *and a semicolon,* for example

```
60 PRINT "A"; CHR$ 6; "B"
```

This statement displays

```
A                    B
```

Another way of using CHR$ control codes is to form a composite string containing them. The statement

```
60 PRINT "A"+CHR$ 6+"B"
```

has exactly the same effect as the previous example.

Codes 16 to 23 affect colour and position and each may be used in a composite string together with CHR$ followed by a colour code value from 0 to 7 for CHR$ 16 (INK control) and CHR$ 17 (PAPER control), or by 0 or 1 for CHR$ 18 to CHR$ 21 (FLASH, BRIGHT, INVERSE and OVER controls). The command

```
PRINT CHR$ 16+CHR$
3+CHR$ 17+CHR$ 6+CHR$
18+CHR$ 1+ "ZX
SPECTRUM+"
```

displays ZX SPECTRUM+ flashing in red and yellow. Alternatively, as above, each plus (+) sign may be replaced by a semicolon.

CHR$ 22 (AT control) is followed by two CHR$ values to indicate the line and column numbers. The command

```
PRINT CHR$ 22+CHR$ 11+
CHR$ 16+CHR$ 42
```

displays a star in the centre of the screen.

CHR$ 23 (TAB control) is also followed by two values in the same way. The second value is normally 0 and the first value gives the TAB position. The command

```
PRINT CHR$ 23+ CHR$ 16+
CHR$ 0+CHR$ 42
```

displays a star halfway across the screen.

Note that only these controls are available. Using PRINT CHR$ with a keyword value greater than 164 simply displays the keyword and does not bring it into operation.

Format
CHR$ int-num-const [;] [+]
CHR$ int-num-var [;] [+]
CHR$ (int-num-expr) [;] [+]

CIRCLE

Keyboard location
EXTEND MODE
SYMBOL SHIFT H

Statement/Command

CIRCLE draws a circle on the screen.

How to use CIRCLE
CIRCLE is followed by three numeric values *each separated by a comma,* for example

```
80 CIRCLE x,y,z
```

Each of the three values is rounded to the nearest integer if necessary. CIRCLE then draws

a circle on the high-resolution graphics grid in the current ink colour. The first two values (x,y) define the horizontal and vertical coordinates of the centre, and the third value (z) defines the length of the radius. The dimensions must be such that the circle does not extend beyond the display area.

CIRCLE is affected by colour statements or commands and may include embedded colour statements with the same effects as PLOT and DRAW.

Example
The command

CIRCLE 128,88,87

draws a circle taking up most of this display area.

Format
CIRCLE [statement;] int-num-expr, int-num-expr, int-num-expr

CLEAR

Keyboard location
X

Statement/Command

CLEAR deletes the current values of all variables, freeing the memory space that the values occupied and space as far as RAMTOP, the top address of the BASIC system area. CLEAR may also be used to reset RAMTOP.

How to use CLEAR
CLEAR may be used as a direct command or it may form a statement in a program. It requires no parameters, for example

50 **CLEAR**

CLEAR then deletes the values that are currently assigned to all variables, including arrays. It also executes CLS and RESTORE to clear the screen and restore the data pointer to the first item of data. In addition, the PLOT position is reset to the bottom left-hand corner of the display area and the GOSUB stack is cleared.

Note that CLEAR is not required before re-dimensioning arrays as DIM deletes an existing array of the same name. Note also that RUN executes CLEAR.

CLEAR and RAMTOP
CLEAR may also be followed by a numeric value, for example

CLEAR 65267

CLEAR then executes CLEAR as above and also sets RAMTOP, the highest address of the BASIC system area, to the given value. RAMTOP is set at 65367 in the ZX Spectrum+, and lies below the area reserved for user-defined graphics. NEW clears out the memory as far as RAMTOP so using CLEAR to lower RAMTOP (by 100 bytes in the above example) provides more memory that is immune from NEW. Raising RAMTOP gives more space for BASIC at the expense of user-defined graphics. Note that the GOSUB stack is then located at RAMTOP.

The current RAMTOP address can be located by using the following command

PRINT PEEK 23730+256***PEEK** 23731

Format
CLEAR [num-expr]

CLOSE

Microdrive file-handling command. See Microdrive and Interface 1 manual.

CLS CLear Screen

Keyboard location
V

Statement/Command

CLS clears all text and graphics from the display area, leaving it blank in the current paper (background) colour.

How to use CLS
CLS may be used as a direct command or it may form a statement in a program. It requires no parameters, for example

250 **IF** a$="NO" **THEN CLS**

The display area (but not the border) is then cleared to the colour selected by the previous PAPER statement or command or to the default paper colour of white.

Note that CLS must be used after PAPER and before PRINT or any other display statement to produce a coloured background over the whole display area.

Format
CLS

CODE

Keyboard location
EXTEND MODE
I

Function
CODE gives the code number of a character in the Spectrum character set (see page 51).

How to use CODE
CODE is followed by a string value, for example

90 IF **CODE** a$<65 **OR CODE** a$>90 **THEN GOTO** 80

A string expression must be *enclosed in brackets.* CODE returns the code number of the *first* character in the string. If this is a null string (" "), then CODE returns 0.

SAVE/LOAD/VERIFY CODE
CODE is used in a different way with SAVE, LOAD and VERIFY. See respective entries.

Format
CODE string-const
CODE string-var
CODE (string-expr)

CONTINUE

Keyboard location
C

Command

If a program stops, CONTINUE can be used to restart the program from the point at which it stopped. If an error has occured to halt the program, then it must be rectified before CONTINUE will allow the program to resume.

How to use CONTINUE
CONTINUE is used as a direct command when a program has stopped. It requires no parameters. After CONTINUE a program then normally resumes at the same statement at which it stopped. If the cause was an error, then a command can be entered to rectify the error and CONTINUE will allow the program to continue from the statement. If the program stopped at a STOP statement giving report 9 or if it halted because the BREAK key was pressed giving report L, then CONTINUE causes the program

to resume from the next statement. A rectifying command can be entered first if necessary.

If CONTINUE is used to resume a direct command, then it will go into a loop if the command stopped at the first statement in the command. The display disappears, but control can be regained by pressing BREAK. CONTINUE gives report 0 if the command stopped at the second statement and report N at the third or subsequent statements.

Format
CONTINUE

COPY

Keyboard location
Z

Command

COPY makes Sinclair-type printers produce a copy of the screen display.

How to use COPY
COPY is used as a direct command when a program has been completed or stopped. It requires no parameters. After COPY, and providing the printer is connected, a copy of the first 22 lines of the screen display is then printed. Note that all ink (foreground) colours are printed in black; paper (background) colours are not printed. The printer can be stopped by pressing BREAK.

If a program listing appears on the screen, it can be printed by using COPY provided it was produced by a LIST command or statement. Note that a listing will appear on the screen on pressing ENTER after a program has been completed or stopped, but this 'automatic' listing cannot be printed with COPY.

Format
COPY

COS COSine

Keyboard location
EXTEND MODE
W

Function

COS gives the cosine of an angle.

How to use COS
COS is followed by a numeric value, for example

```
140 LET x=COS y
```

An expression must be *enclosed in brackets.* The value following COS is the angle *in radians.* COS then returns the cosine of the angle. Degrees may be converted into radians by multiplying by PI/180.

Note that COS returns a negative value for angles from 90 to 270 degrees and a positive value for angles from 0 to 90 and 270 to 360 degrees.

Example
The command

```
PRINT COS (60*PI/180)
```

displays 0.5, the cosine of 60 degrees.

Format
COS num-const
COS num-var
COS (num-expr)

DATA

Keyboard location
EXTEND MODE
D

Statement

DATA provides a list of items of data within a program. These items may be values of variables or strings to be displayed, for example. Each item is assigned to a variable by a READ statement.

Assignment is carried out in the order in which items of data appear in the program, but RESTORE can be used to begin assignment at the first item in a given DATA statement.

How to use DATA
DATA can only be used to form a statement in a program. It is normally followed by a list of numeric or string constants *each separated by a comma,* for example

```
50 DATA 31, "JAN", 28,
   "FEB"
```

Each constant is then assigned to a variable by a READ statement that reads the DATA. The DATA statement may be positioned anywhere in the program. The number, kind (numeric or string) and order of the constants must correspond to the number of times the READ statement is executed and the kind and order of variables in the READ statement. The list of data may be split up into several successive DATA statements if there are too many items to fit into one statement.

Example
The following program

```
10 FOR n=1 TO 2
20 READ x,a$
30 PRINT a$,x; "days"
40 NEXT n
50 DATA 31, "JAN", 28, "FEB"
```

displays

```
JAN          31 days
FEB          28 days
```

Using DATA with variables
The items of data in a DATA statement may consist of numeric or string variables or expressions provided the variables have previously been assigned values. In the above example, the DATA statement may be changed to

```
50 DATA d,m$,d-3, "FEB"
```

If d is previously assigned a value of 31 and m$ a value of "JAN", then the same display is given.

LOAD DATA, SAVE DATA and **VERIFY DATA**
DATA may also be used with LOAD, SAVE and VERIFY to store arrays on tape. See LOAD DATA, SAVE DATA and VERIFY.

Format
DATA num-expr [,num-expr] [,string-expr]
DATA string-expr [,num-expr] [,string-expr]

DEF FN DEFine FuNction

Keyboard location
EXTEND MODE
SYMBOL SHIFT 1

Statement

DEF FN enables the user to define a function that is not available as a keyword. A variety of parameters can be passed to the function in an FN statement, which calls the function and may return either a numeric or string value as a result.

How to use DEF FN
DEF FN may only be used as a statement in a program. If a numeric function is to be

defined, DEF FN is followed by any single letter and then by one or more numeric variables *each separated by a comma and enclosed in brackets,* for example DEF FN r(x,y). This is followed by an equals sign and then a numeric expression containing the variables, for example

```
1000 DEF FN r(x,y)=
     SQR (x↑2+y↑2)
```

The letter following DEF FN (r above) is a name that identifies the function. The variables may also only be single letters. Note that in both cases, the Spectrum does not distinguish between capital and lower-case letters.

The expression that follows the equals sign uses the variables (x and y above) to define the function.

A DEF FN statement may be placed anywhere in a program. To call the function that it defines, a FN statement is used. This is then followed by the function name letter and a list of numeric values *each separated by a comma and enclosed in brackets,* for example

```
50 PRINT FN r(3,4)
```

The values within the brackets are passed to the function in the same order as the variables in the DEF FN statement. Thus, in this example, x is assigned a value of 3 and y a value of 4. FN evaluates the expression and returns the value.

DEF FN may also be followed by a letter and a pair of brackets only, for example

```
1000 DEF FN r ()=INT (x+0.5)
```

The value currently assigned to the variable (x above) is passed to the function when it is called by FN. In this case, FN r() returns the value currently assigned to x rounded to the nearest integer.

DEF FN and strings

DEF FN and FN may also be used in the same way to define and call a string function. In this case, the function name is a single letter followed by $ and one or more of the variables in the statement is a letter followed by $. A corresponding string expression forms the definition, for example

```
1000 DEF FN a$(b$,x,y)=
     b$(x TO y)
```

The string expression following the equals sign in this example is a string slicer, and x and y are the first and last characters of a section of b$. FN must be followed by the function name and, in brackets, a string value together with any other parameters that are to be passed to the function. In this case, the command

```
PRINT FN a$
("FUNDAMENTAL", 1,3)
```

displays FUN, and the command

```
PRINT FN a$
("FUNDAMENTAL", 5,8)
```

displays AMEN.

Format

```
DEF FN letter ([letter] [,letter])
=num-expr
DEF FN letter$ ([letter$] [letter]
[,letter] [,letter$])=string-expr
FN letter ([num-expr] [,num-
expr])
FN letter$ ([string-expr] [num-
expr] [,num-expr] [,string-
expr])
```

DIM DIMension

Keyboard location

D

Statement

DIM is used to dimension (set up) an array of a given number of numeric or string variables. An array is a list of variables of the same name that are distinguished by *subscripts* (values that identify each variable or element in the array).

How to use DIM with numeric arrays

DIM is used to form a statement in a program. It is followed by a single letter that names the array, and one or more numeric values *each separated by a comma and enclosed in brackets,* for example

```
20 DIM x (10)
80 DIM z (20,5)
```

In the first case, a one-dimensional numeric array is created containing ten elements with subscripts from 1 to 10. The array has the name x and the subscripted variables are x (1) to x (10) inclusive. Any existing array of the same name is deleted, and the variables are each assigned a value of 0. Note that in dimensioning an array, the Spectrum does not distinguish between names with capital and lower-case letters – variable x (2) is the same as X (2). However, simple numeric variables having the same letter as an array name (x or X) can coexist and may be used separately if required.

The number of values in brackets equals the number of dimensions created in a numeric array. The second example sets up a two-dimensional array of 100 elements with 20 elements in the first dimension and 5 in the second. These elements are numbered z(1,1) up to z (20,5).

Arrays of any number of dimensions may be created.

The elements of a numeric array may subsequently be identified by the array name followed by a value *in brackets,* for example

```
70 PRINT x (a)
160 PRINT z (7,b)
```

DIM and string arrays

DIM is used in the same way as with numeric arrays except that a single letter followed by $ is used for the array name. Furthermore, an extra value must be added to the dimension values in brackets in order to define the length of each string, for example

```
30 DIM a$ (20,5)
90 DIM b$ (20,5,10)
```

The first statement creates an array of 20 elements, each of which contains a string of 5 characters. The subscripted variables are named a$ (1) to a$ (20) inclusive, and they are initially assigned a null (empty) string (" "). Any existing array of the same name is deleted and, unlike numeric arrays, a simple string variable of the same name cannot coexist.

The second example creates a two-dimensional string array of 100 elements with 20 elements in the first dimension and 5 in the second. All elements have a length of 10 characters.

When string values are subsequently assigned to a string array, they are padded out with spaces at the end of the string or truncated to the defined length if necessary.

The elements of a string array are identified by the array name

followed, in brackets, by one or more numeric values giving the subscript number(s). For example, element a$(2) may be "SMITH" and element b$(12,4) may be "DERBYSHIRE". However, an extra value may be added to define a particular character in the string. In these examples, a$(2,2) would be "M" (the second character in "SMITH"), and b$ (12,4,5) would be "Y".

Zero-dimension string arrays
It is possible to create a zero-dimension string array by using only one value in brackets, for example

```
10 DIM c$ (15)
```

This array has only one element, which is c$, and its length is fixed at the defined value (15 characters).

Format
DIM letter (num-expr [,num-expr])
DIM letter$ (num-expr [,num-expr])

DRAW

Keyboard location
W

Statement/Command

DRAW is used to draw straight lines and curves on the screen.

How to use DRAW
DRAW is normally used to form a statement in a program. If a straight line is required, it is followed by two numeric values *separated by a comma,* for example

```
40 DRAW x,y
```

A straight line is then drawn on the high-resolution graphics grid from the position defined by the previous PLOT statement or the position reached by the previous DRAW statement, whichever is last. Both values following DRAW are rounded to the nearest integer if necessary. The first value (x above) defines the horizontal distance from this position, and the second value (y) the vertical distance. These values are negative if the line is to go to the left or down respectively, and the position reached must be within the display area.

If there is no previous PLOT or DRAW statement, DRAW commences at position 0,0 (the bottom left-hand corner of the screen).

DRAW is affected by colour statements or commands and may include embedded statements with the same effects as with PLOT and CIRCLE.

DRAWing curved lines
DRAW may be followed by a third value to produce a curve that is a part of a circle, for example

```
40 DRAW x,y,z
```

The third value (z above) defines the angle (in radians) through which the line turns as it is drawn. The line turns to the left if this value is positive, and to the right if it is negative. Values of PI or –PI produce a circle.

Example
The following program draws a triangle:

```
10 PLOT 127,150
20 DRAW 70,–100
30 DRAW –140,0
40 DRAW 70,100
```

Adding, 1 or, –1 to the DRAW statement causes the sides to curve in or out respectively.

Format
DRAW [statement;] int-num-expr, int-num-expr [,int-num-expr]

ERASE

Microdrive file-handling command. See Microdrive and Interface 1 manual.

EXP EXPonent

Keyboard location
EXTEND MODE
X

Function

EXP is a mathematical function that raises the exponent *e* to a given power.

How to use EXP
EXP is followed by a numeric value for example

```
60 LET y=EXP x
```

An expression must be *enclosed in brackets.* EXP then returns the exponent *e* raised to the power of the argument (x above).

Example
The command

```
PRINT EXP 1
```

displays 2.7182818, the value of *e.*

Format
EXP num-const
EXP num-var
EXP (num-expr)

FLASH

Keyboard location
EXTEND MODE
SYMBOL SHIFT V

Statement/Command

FLASH causes character positions to flash, making the ink and paper colours alternate at a constant rate.

How to use FLASH
FLASH may be used as a direct command but it is normally used to form a statement in a program. It is followed by a numeric value, for example

```
50 FLASH 1
```

The value following FLASH is rounded to the nearest integer if necessary and may then be either 0, 1 or 8. A value of 1 causes all characters subsequently displayed by PRINT or INPUT to flash. A value of 8 causes flashing character positions to remain flashing and normal character positions to remain normal when new characters are printed there. FLASH followed by 0 cancels both FLASH 1 and FLASH 8 so that all characters subsequently displayed are normal.

FLASH may also be embedded (inserted) within display statements formed by PRINT, INPUT, PLOT, DRAW and CIRCLE. FLASH follows the keyword but precedes the data or display parameters; it is followed by the same values *and a semicolon,* for example

```
120 PRINT FLASH 1; INK 2;
    PAPER 6; "WARNING"
```

The effect of FLASH is then local and applies only to the characters displayed, point plotted or line drawn by the display statement. Note that FLASH 1 causes the whole 8x8 pixel position to flash if any pixel is plotted in an ink colour.

Format
FLASH int-num-expr[;]

FN FuNction

Keyboard location
EXTEND MODE
SYMBOL SHIFT 2

Function

FN calls a user-defined function. It is always used in conjunction with DEF FN, which defines the function to be called.

How to use FN
If a numeric function is to be called, FN is followed by a letter and then a pair of brackets. If any parameters are to be passed to the function, then these are *each separated by a comma and enclosed in the brackets*, for example

```
170 LET x=FN r (3,4)
```

The parameters (3 and 4 above) are then passed to the function called r. FN then returns the result. If no parameters are to be passed, the pair of brackets must still be included, for example

```
70 PRINT FN r()
```

In this case, the function uses the values currently assigned to its variables.

FN calls a string function in the same way, except that $ must be added after the letter (see DEF FN).

FN does not work recursively.

Format
FN letter ([num-expr] [,num-expr])
FN letter$ ([string-expr] [num-expr][,num-expr][,string-expr])

FOR

Keyboard location
F

Statement/Command

FOR is always used with the keywords TO and NEXT to create a FOR NEXT loop. This structure enables a section of the program to repeat a given number of times.

How to use FOR
FOR always forms a statement with TO. FOR is followed by a letter, an equals sign, and then two numeric values separated by TO, for example

```
60 FOR a=1 TO 9
```

The letter (a above) forms a control variable. The statements that are to be repeated follow, and one or more of these normally makes use of the control variable. The loop then ends with a NEXT statement, in which NEXT is followed by the control variable, for example

```
90 NEXT a
```

On execution, FOR deletes any variable of the same name as the control variable and assigns it an initial value equal to the value before TO (1 above). The statements are then executed with the control variable having this value. On reaching NEXT, the value of the control variable is increased by 1. If this value is less than or equal to the value after TO (the limit value of 9 above), the program returns to the FOR statement and the FOR NEXT loop is repeated. If the control variable has a greater value than the limit value, then the loop ends and the program continues with the statement after NEXT.

In the above example, the loop is repeated nine times with the control variable a increasing from 1 to 9. On leaving the loop, a has a value of 10.

Note that the Spectrum does not distinguish between capital and lower-case letters when naming the control variable.

Using STEP in a FOR NEXT loop
STEP is a keyword that can be incorporated in a FOR statement if the control variable is to increase by a value other than 1 or to decrease. STEP follows the limit value and is followed by a numeric value, for example

```
60 FOR a=1 TO 9 STEP 2
```

The control variable is increased by the step value (2 above) until it is greater than the limit value. The control variable a has successive values of 1, 3, 5, 7 and 9 and leaves the loop with a value of 11.

A negative step value causes the value of the control variable to decrease. In this case, the initial value must be greater than the limit value and the loop ends when the value of the control variable is less than the limit value, for example

```
60 FOR a=9 TO 1 STEP -1
```

The value of a decreases from 9 to 1 and leaves the loop with a value of 0.

Nesting loops
One or more FOR NEXT loops may be placed inside each other, a procedure called 'nesting' loops. The order of the control variables in the NEXT statements must be the reverse of the order of the control variables in the FOR statements. FOR NEXT loops may be nested to any depth, that is as many loops as required may be placed inside each other.

Format
FOR letter=num-expr **TO** num-expr [**STEP** num-expr]
NEXT letter

FORMAT

Microdrive file-handling command. See Microdrive and Interface 1 manual.

GOSUB

Keyboard location
H

Statement/Command

GOSUB causes the program to branch to a subroutine, which is a separate section of the program. This is useful if a subroutine is required several times in a program.

How to use GOSUB
GOSUB may be used as a statement or direct command and it is followed by a numeric value, for example

```
GOSUB 1000
```

On execution, the value following GOSUB (1000 above) is rounded to the nearest integer, and the program branches to the line number having this value. The use of a variable or expression enables the program to branch to a subroutine at a calculated line number. Note that if the line number does not exist, the program still branches and continues with the first statement that is then encountered.

A subroutine ends with RETURN, and the program then branches back to the statement

following the GOSUB statement. Subroutines may be nested so that one is reached from another, in which case RETURN sends the program back to the statement following the last GOSUB statement executed.

The GOSUB stack
Whenever GOSUB is executed, its line number is placed on the GOSUB stack in the memory. If two or more GOSUBs are executed before RETURN, their line numbers stack up so that the last number is on top of the stack. RETURN always takes the top line number from the stack and goes to this line to continue the program.

Note that error 4 (Out of memory) can occur if there are not enough RETURN statements.

Format
GOSUB int-num-expr

GOTO

Keyboard location
G

Statement/Command

GOTO makes a program branch to a particular line.

How to use GOTO
GOTO may be used as a direct command to run a program from a given line number *without* first clearing the screen. It may also be used to form a statement in a program. GOTO is followed by a numeric value, for example

```
60 GOTO 350
```

On execution, the value following GOTO is rounded to the nearest integer and the program branches to the line number having this value. The use of a variable or expression allows the program to branch to a calculated line number. Note that if the line does not exist, then the program still branches and continues with the first statement that is then encountered.

Format
GOTO int-num-expr

IF

Keyboard location
U

Statement/Command

IF is always used with the keyword THEN to prompt a decision that affects subsequent action. To do this, the computer tests something to find out whether or not it is true. If it is true, then one course of action follows. If it is untrue, another occurs.

How to use IF and THEN
IF normally forms a statement with THEN. IF is first followed by a numeric value or by a condition, and second by THEN and one or more valid BASIC statements, for example

```
80 IF x THEN GOTO 250
240 IF a$="NO" THEN PRINT
    "THE END": STOP
```

A constant, variable or expression (such as x above) is considered to be true if it has a non-zero value. In this case, the statement following THEN and any more statements *in the same line* are executed. The program then proceeds to the next line. If the value is 0, then the constant, variable or expression is considered to be false. The following statements are then not executed and the program skips to the next line. In the example, the program will not go to line 250 if x is 0.

If a condition (a$="NO") following IF is true, then the statements following THEN are executed. If the condition is false, then the program moves to the next line. In this example, if a$ has the value "NO" then "THE END" is displayed and the program stops. If a$ has any other value, the program continues from the next line.

The Spectrum gives a true condition a value of 1 and a false condition a value of 0. It recognises any non-zero value as true and 0 as false. A variable can be assigned the value of a condition by a statement such as

```
70 LET x=a$="NO"
```

Note that, unlike in some other BASICs, THEN cannot be omitted before GOTO.

Format
IF num-expr **THEN** statement [: statement]
IF cond **THEN** statement [: statement]

IN

Keyboard location
EXTEND MODE
SYMBOL SHIFT I

Function

IN checks the status of the keyboard and other input and output devices. It reads a byte from a given port address that indicates the status of the device connected to the port.

How to use IN
IN is followed by a numeric value, for example

```
150 LET x=IN y
```

The value following IN may range from 0 to 65535 and specifies the port address that is to be read. IN then returns the byte read from this port.

Keyboard addresses
The keyboard has eight addresses, each of which may contain one of five different bytes depending on which key is pressed. The addresses are 65278, 65022, 64510, 63486, 61438, 57342, 49150 and 32766. Byte values at these addresses may be 175, 183, 187, 189 or 190.

Format
IN num-const
IN num-var
IN (num-expr)

INK

Keyboard location
EXTEND MODE
SYMBOL SHIFT X

Statement/Command

INK specifies the foreground colour in which characters are displayed, points plotted and lines and curves drawn.

How to use INK
INK may be used as direct command but is normally used to form a statement in a program. It is followed by a numeric value, for example

```
70 INK x
```

The value following INK is rounded to the nearest integer and may then range from 0 to 9. The following foreground colours are then given.

0	Black
1	Blue

2	Red
3	Magenta (Purple)
4	Green
5	Cyan (Blue-green)
6	Yellow
7	White
8	Transparent
9	Contrasting black or white

INK 8 specifies that the existing colour remains unchanged at any position on the screen where INK 8 is used. INK 9 causes the ink colour to be either black or white so that it shows up against the paper (background) colour.

Global and local ink colours
When INK forms a statement alone, as above, the colour is global and all subsequent displays occur in this foreground colour. INK may also be embedded (inserted) in display statements formed by PRINT, INPUT, PLOT, DRAW and CIRCLE. INK follows the keyword but precedes the data or display parameters; it is followed by the same values *and a semicolon*, for example

```
60  CIRCLE INK 4; 128, 88, 87
```

The effect of INK is then local and applies only to the characters displayed, point plotted or line drawn by the display statement, this example drawing a green circle. Thereafter the ink colour reverts to the global colour or default colour of black.

Format
INK int-num-expr [;]

INKEY$ INput KEY string

Keyboard location
EXTEND MODE
N

Function

INKEY$ is used to detect the pressing of the keys on the keyboard.

How to use INKEY$
INKEY$ requires no argument and is generally used to assign a character to a string variable or to test for a particular character, for example

```
70  LET a$=INKEY$
130  IF INKEY$="N" THEN
STOP
```

On execution, INKEY$ returns the character given by the key that is being pressed at that instant. If no key is being pressed, then INKEY$ returns a null (empty) string (""). Note that INKEY$ distinguishes between capital and lower-case letters and other shifted and unshifted characters. (Use IN to detect any key without distinguishing characters.)

Unlike INPUT, INKEY$ does not wait but goes immediately to the next statement. It is therefore normally placed inside a loop that repeats until the required key is pressed.

Example
This line suspends operation until the Y key is pressed (without CAPS SHIFT or CAPS LOCK).

```
60  IF INKEY$<>"y" THEN
    GOTO 60
```

Format
INKEY$

INPUT

Keyboard location
I

Statement/Command

INPUT enables data to be entered during the running of a program.

How to use INPUT
INPUT normally forms a statement in a program and is used in a very similar way to PRINT. In its simplest form, it is followed by a numeric or string variable, for example

```
60  INPUT x
90  INPUT a$
```

The computer then waits until either a number or a string is entered. The value is displayed at the beginning of the bottom line as it is keyed in. On pressing ENTER, the value is assigned to the named variable and the program continues.

An INPUT statement may include more than one variable and will display characters to form a prompt. This is done in exactly the same way as with PRINT, using quote marks to enclose the prompt characters and semicolons or commas as necessary to separate items. Display statements such as INK, FLASH and PAPER may be embedded, for example

```
80  INPUT INK 2; "What is your
    name? ";n$, ("How old are
    you, "+n$+"? "); age
```

Note the following differences to PRINT. INPUT waits when it comes to a variable, so all variables and expressions (such as that including n$ above) which are to be included in prompts must be *enclosed in brackets*. Display begins at the start of the bottom line and then scrolls up if more than one line is used. AT may be used in an INPUT statement in the same way as with PRINT. AT 0,0 displays at the start of the line above the bottom line and the display scrolls up if more than two lines are displayed.

How to halt INPUT
If INPUT is followed by a numeric variable and STOP is entered, then the program stops. With a string variable, the first quote mark that appears may be deleted and then STOP entered to halt the program.

Using INPUT with LINE
INPUT LINE may be used with string variables only. Normally, INPUT with a string variable causes a pair of quotes to be displayed. As the string is keyed in, it appears between the quotes. To remove these quotes, use INPUT LINE followed by the string variable. If a prompt is required, it is placed between INPUT and LINE, for example

```
70  INPUT "What is your
name? "; LINE n$
```

Format
INPUT [prompt][;][,][']
num-var
INPUT [prompt][;][,][']
string-var
INPUT [prompt][;][,]['] **LINE**
string-var
[prompt]=[string-const]
[(string-expr)][**AT**
int-num-expr,int-num-expr]
[statement] [;][,][']

INT INTeger

Keyboard location
EXTEND MODE
R

Function

INT changes non-integers (numbers that are not whole) into integers or whole numbers.

How to use INT
INT is followed by a numeric value for example

```
70  LET x=INT y
```

An expression must be *enclosed in brackets*. INT then returns the value *rounded down* to an integer.

Example
The command

```
PRINT INT 45.67, INT -7.66
```

displays

```
45                    -8
```

Format
INT num-const
INT num-var
INT (num-expr)

INVERSE

Keyboard location
EXTEND MODE
SYMBOL SHIFT M

Statement/Command

INVERSE causes colours to be inverted at character positions so that the ink colour becomes the paper colour and vice-versa.

How to use INVERSE
INVERSE is normally used to form a statement in a program. It is followed by a numeric value, for example

```
70  INVERSE 1
```

The value following INVERSE is rounded to the nearest integer and may then be either 0 or 1. INVERSE 1 causes all subsequent displays made by PRINT and INPUT to be produced in inverse colours. INVERSE 0 restores the ink and paper colours to normal.

Note that INVERSE can be embedded (inserted) within display statements in the same way as INK. However, if used with CIRCLE, PLOT or DRAW, INVERSE 1 causes a line or point to be plotted in the paper colour so that it disappears.

Format
INVERSE int-num-expr

LEN LENgth of string

Keyboard location
EXTEND MODE
K

Function

LEN gives the length of a string.

How to use LEN
LEN is followed by a string value, for example

```
50  LET x=LEN a$
```

An expression must be *enclosed in brackets*. LEN returns the number of characters in the string.

Example
The following line

```
120  INPUT a$: IF LEN a$>9
     THEN GOTO 120
```

passes only strings containing up to nine characters.

Format
LEN string-const
LEN string-var
LEN (string-expr)

LET

Keyboard location
L

Statement/Command

LET is used to assign a value to a variable. In Sinclair BASIC, LET cannot be omitted in an assignment statement.

How to use LET
LET normally forms a statement in a program but may be used as a direct command. It is followed by a numeric or string variable, an equals sign, and then a value. The value may be numeric or string, depending on the variable preceding LET, for example

```
60  LET x=x+1
80  LET a$="Correct"
```

The value is then assigned to the variable.

Note that simple variables are undefined until assigned values by LET, READ or INPUT. Array variables however are initialized to 0 or a null string (see DIM).

Format
LET num-var=num-expr
LET string-var=string-expr

LINE

Keyboard location
EXTEND MODE
SYMBOL SHIFT 3

see **SAVE**

LIST

Keyboard location
K

Command/Statement

LIST produces a listing of the program currently in the memory.

How to use LIST
LIST is normally used as a direct command but may form a statement in a program. To list a complete program, it is used alone. After the direct command

```
LIST
```

the first page of the listing appears and subsequent pages will scroll up the screen at the touch of any key except N, the space bar, STOP or BREAK.

LIST may also be followed by a line number, in the form of a numeric value, for example

```
LIST 100
```

The value following LIST is then rounded to the nearest integer if necessary, and the listing commences at this line. If there is no line with this number, the listing commences at the next line.

Format
LIST [int-num-expr]

LLIST Line printer LIST

Keyboard location
EXTEND MODE
V

Command/Statement

LLIST makes Sinclair-type printers produce a print-out listing of the program currently in memory.

How to use LLIST
LLIST is used in exactly the same way as LIST (see LIST for further details). Note that the screen display does not change as the listing is printed.

Format
LLIST [int-num-expr]

LN Logarithm (Natural)

Keyboard location
EXTEND MODE
Z

Function

LN gives the natural logarithm (the logarithm to base *e*) of a value. It acts as the inverse of EXP.

How to use LN
LN is followed by a numeric value, for example

```
60 LET x=LN y
```

An expression must be *enclosed in brackets*. The value following LN must be greater than 0. LN then returns the natural logarithm of this value.

Format
LN num-const
LN num-var
LN (num-expr)

LOAD

Keyboard location
J

Command/Statement

LOAD loads a complete program into the memory from a tape.

How to use LOAD
LOAD is normally used as a direct command, but it may form a statement in a program in order to load a new program. LOAD is followed by a filename, which is a string value up to ten characters long, for example

```
LOAD "filename"
```

On execution, the program currently in memory, and all the values of its variables, are deleted. The Spectrum then searches for the named program and loads it when it is located. Note that the computer distinguishes between capital and lower-case letters in program names.

If a null string follows LOAD, as in this command

```
LOAD " "
```

then the Spectrum loads the first complete program that it locates.

Note that LOAD is used differently when a Microdrive is connected. See the Microdrive and Interface 1 manual for details.

Format
LOAD string-expr

LOAD CODE

Keyboard locations
J
EXTEND MODE
I

Command/Statement

LOAD CODE is used to load a section of the memory with information that has been stored on tape. The information consists of a set of bytes, and these are sent to a set of addresses in the memory. LOAD CODE can be used to load a display, or to load information for user-defined graphics, for example.

How to use LOAD CODE
LOAD CODE may be used as a direct command or it may form a statement in a program. LOAD is followed by a filename, which is a string value, and then CODE, for example

```
LOAD "data" CODE
```

The filename following LOAD is the name of the information to be loaded and is subject to the same restrictions as program names (see LOAD). LOAD CODE then searches for the named information and when found, displays Bytes: followed by the name. The Spectrum then loads the bytes into the memory at the addresses from which they were saved. Any existing information is overwritten.

CODE may also be followed by one or two numeric values, *separated by a comma,* for example

```
LOAD "picture" CODE
16384,6912
```

The values following CODE are rounded to the nearest integer and then define the starting address (16384 above) at which the named information is to be loaded, and the number of bytes (6912) that are to be sent to locations beginning at this address. If the number is wrong, the tape loading error report is given. If only one value follows CODE, it defines the starting address from which *all* the bytes are to be located.

The above example can also be carried out by the keywords LOAD SCREEN$.

For details on storing bytes, see SAVE CODE.

Format
LOAD string-expr **CODE**
[int-num-expr][,int-num-expr]

LOAD DATA

Keyboard locations
J
EXTEND MODE
D

Statement/Command

LOAD DATA is used to load arrays from tape. The arrays are recorded using SAVE DATA.

How to use LOAD DATA
LOAD DATA may be used to form a statement in a program or as a direct command. LOAD is first followed by a filename, which is a string value, followed by DATA and a letter or a letter and $, and finally by a pair of *empty* brackets, for example

```
270 LOAD "numbers"
DATA n()
300 LOAD "names" DATA
n$()
```

The filename following LOAD is the name that is given to the array on tape and it is subject to the same restrictions as program names used with LOAD. The letter or letter$ following DATA is the name to be given to the array in the program when it is loaded and used.

On execution, the Spectrum searches for the named array on tape. When found, the message Number array: or Character array: followed by the name appears and the array is loaded. Any array currently in memory having the same letter name (n or n$ above) is deleted, and a new array having this letter name and the values stored on tape is created. Note that with character arrays, any string variable currently in memory having the same letter name is also deleted.

Format
LOAD string-expr **DATA**
letter[$]()

LOAD SCREEN$

Keyboard locations
J
EXTEND MODE
SYMBOL SHIFT K

Statement/Command

LOAD SCREEN$ enables a screen display to be loaded directly from tape. It sends information from the tape to the section of the memory controlling the screen display in order to produce the picture.

How to use LOAD SCREEN$
LOAD SCREEN$ may be used to form a statement in a program or as a direct command. LOAD is followed by a filename, which is a string value, and then SCREEN$, for example

LOAD "picture" **SCREEN$**

The filename following LOAD is the name that is given to the screen information on tape, and it is subject to the same restrictions as program names used with LOAD. The Spectrum then searches for the named information and when found, loads it first into the display file and then the attributes section of the memory. The picture slowly builds up in the current ink and paper colours and then the attributes (true colours and so on) are added.

For details on storing screen information, see SAVE SCREEN$.

Format
LOAD string-expr **SCREEN$**

LPRINT Line printer PRINT

Keyboard location
EXTEND MODE
C

Statement/Command

LPRINT makes Sinclair-type printers print an item of data in the same way that PRINT causes the item to appear on the screen.

How to use LPRINT
LPRINT may form a statement in a program or a direct command. It is followed by items of data that may be separated by semicolons, commas or apostrophes, for example

```
60 LPRINT "Number ";x'
   "Name ";n$,"Age ";a
```

When output to the printer, the items are printed in the same format as PRINT would cause them to be displayed on the screen. An LPRINT statement or command may also include TAB statements, certain CHR$ controls, INVERSE and OVER statements and control codes with the same effect as PRINT. An AT statement may also be included, but the line number is ignored and the item of data printed at the given column position in the same line.

Format
LPRINT [**TAB** int-num-expr;)] [**AT** int-num-expr,int-num-expr;] [**CHR$** (int-num-expr);][statement;][num-expr][string-expr][;][,][']

MERGE

Keyboard location
EXTEND MODE
SYMBOL SHIFT T

Statement/Command

MERGE allows two programs to be merged together.

How to use MERGE
MERGE may be used to form a statement in a program or as a direct command. It is followed by a filename in the form of string value, for example

500 **MERGE** "prog2"

The filename following MERGE is the name of the program to be merged with the program currently in memory. This name is subject to the same restrictions as program names used with LOAD. MERGE then loads the new program without first deleting the existing program. However, the new program overwrites any lines in the existing program that have the same line numbers as lines in the new program, and variables with the same name are also overwritten.

Format
MERGE string-expr

MOVE

Microdrive file-handling command. See the Microdrive and Interface 1 manual.

NEW

Keyboard location
A

Command/Statement

NEW clears the BASIC memory area (the area as far as RAMTOP) removing any program currently in this part of the memory.

How to use NEW
NEW is normally used as a direct command but may form a statement in a program. It is used alone. On execution, the program and variables are deleted. The memory is cleared as far as RAMTOP so that user-defined graphics characters, which are stored above RAMTOP, are not affected.

Format
NEW

NEXT

Keyboard location
N

Statement/Command

NEXT is always used in conjunction with FOR to create a FOR NEXT loop.

How to use NEXT
NEXT is normally used to form a statement in a program to complete a FOR NEXT loop. It is followed by a letter that is the control variable in the loop, for example

90 **NEXT** a

In Sinclair BASIC, the control variable must be included.

See FOR for further details of FOR NEXT loops.

Format
NEXT letter

NOT

Keyboard location
SYMBOL SHIFT S

Logical Operator/Function

NOT is used to reverse the truth of a condition so that a false condition becomes true and vice versa.

How to use NOT
NOT is followed by a condition or by a numeric value, for example

```
90 IF NOT x=y+z THEN
   PRINT "Wrong"
```

```
90 LET correct=x=y+z: IF
   NOT correct THEN PRINT
   "Wrong"
```

When NOT is followed by a condition (x=y+z above), the Spectrum first assigns a value of 1 to the condition if it is true and 0 if it is false. NOT then acts as a function, reversing the value produced, so that the reverse of the condition can be tested. Note that a condition should be enclosed in brackets if it contains AND or OR.

If NOT is followed by a numeric value, it returns 0 if the value following is non-zero and 1 if the value following is 0. Thus in the above examples, the Spectrum prints "Wrong" if x<>y+z or if correct has a value of 0.

Format
NOT cond
NOT num-expr

OPEN#

Microdrive file-handling command. See the Microdrive and Interface 1 manual.

OR

Keyboard location
SYMBOL SHIFT U

Logical Operator/Function

OR acts as a logical operator to test the truth of a combination of conditions. If one or more conditions are true, then the overall combination is true. OR also acts as a function to perform binary operations on two numeric values.

How to use OR
As a logical operator, OR links two conditions in a statement where the truth of the whole is to be tested, for example

```
70 IF INKEY$="N" OR
   INKEY$="n" THEN STOP
```

If any or both of the conditions is true, then the overall combination is true. In the line above, one of the conditions (INKEY$="N" and INKEY$="n") becomes true as soon as the N key is pressed, regardless of whether CAPS SHIFT or CAPS LOCK is operating or not. The whole combination is then true and the program stops.

OR as a function
The ZX Spectrum+ assigns a numeric value of 1 to a true condition and 0 to a false condition. It recognizes any non-zero value as true and 0 as false. OR may therefore be preceded or followed by a numeric value, for example

```
40 LET x=y OR z
```

The variable x is then assigned a value of 1 if z is non-zero or a true condition, or a value of y if z is 0 or a false condition.

This is useful in arithmetic. In the following example, the fare is halved if the age is less than 14.

```
60 PRINT fare*(0.5 OR
   age>13)
```

If the age is less than 14, the condition age>13 is false, so the fare is multiplied by 0.5. If age>13 is true, then the fare is multiplied by 1.

Note that the Spectrum does not evaluate combinations of numeric values in accordance with standard truth tables.

Format
cond **OR** cond
num-expr **OR** num-expr

OUT

Keyboard location
EXTEND MODE
SYMBOL SHIFT O

Statement/Command

OUT sends a byte to a given input/output port address in order to drive an output device.

How to use OUT
OUT may be used to form a statement in a program or as a direct command. It is followed by two numeric values, *separated by a comma*, for example

```
40 OUT 254,3
```

Both values are rounded to the nearest integer. The first value (254 above) may then range from 0 to 65535 and is the port address. The second value (3) may range from 0 to 255 and is the byte to be sent to this address.

Bits 0 to 2 of the byte output to port address 254 set the border colour; the above example therefore turns the border magenta. Bit 3 at this address drives the MIC socket and bit 4 the loudspeaker. Port address 251 drives the printer and ports 254, 247 and 239 are used with other peripherals.

Format
OUT int-num-expr,int-num-expr

OVER

Keyboard location
EXTEND MODE
SYMBOL SHIFT N

Statement/Command

OVER is used to overprint one character on another. It can also be used to plot points or draw lines or curves in a paper colour instead of an ink colour.

How to use OVER
OVER is normally used to form a statement in a program. It is followed by a numeric value, for example

```
80 OVER 1
```

The value following OVER is rounded to the nearest integer and may then be either 0 or 1. OVER 0, which is the default (preset) state, causes any character to obliterate a previous character at the same character position and replace it. OVER 1 causes any two characters displayed at the same character position to be combined.

OVER may be embedded (inserted) in a PRINT or INPUT statement in the same way as INK so that it affects only the characters displayed by the statement. This statement for example, underlines a word

```
60 PRINT AT 11,15; "YES";
   OVER 1; AT 11,15 ;"___"
```

However, note that characters are combined so that the paper colour is given where the ink colours overlap.

OVER in high resolution
OVER may be used with PLOT, DRAW and CIRCLE. Without OVER, lines and curves can overlap each other, but they must have the same ink colour otherwise the ink colour in the whole character position changes where they cross. If OVER 1 is used, lines or curves produce the paper colour where

they overlap or meet characters. Plotting points or drawing lines or curves again in exactly the same position with OVER 1 causes them to disappear.

Format
OVER int-num-expr

PAPER

Keyboard location
EXTEND MODE
SYMBOL SHIFT C

Statement/Command

PAPER is used to select the paper or background colour used for the screen display. This may be either the colour of the background over the whole display area, or the colour behind individual characters, points or lines that appear in single character positions.

How to use PAPER
PAPER may be used to form a statement in a program or as a direct command. It is followed by a numeric value, for example

```
80 PAPER x
```

The value following PAPER is rounded to the nearest integer and it may then range from 0 to 9. The paper colours that are then given are the same as those given by INK. Paper colours may also be global or may be made local by embedding (inserting) them in display statements in exactly the same way as ink colours. See INK for further details.

Whenever characters are printed following a PAPER statement, whether global or local, the background over the whole of the character position affected changes to the selected colour. This is also true when points are plotted or lines or circles drawn with an embedded PAPER statement but not following a global command or statement.

To produce a coloured background over the whole display area, it is necessary to use CLS after a PAPER statement. The entire display is cleared to this colour, which then remains the overall background colour.

Format
PAPER int-num-expr[;]

PAUSE

Keyboard location
M

Statement/Command

PAUSE can be used to suspend a program for a definite or indefinite time.

How to use PAUSE
PAUSE is normally used to form a statement in a program. It is followed by a numeric value, for example

```
130 PAUSE 100
```

The value following PAUSE is rounded to the nearest integer and may then range from 0 to 65535. It defines the delay that occurs as this number of frames of the television picture, so that a value of 50 produces a pause of 1 second in the UK and Europe where the frame frequency is 50 Hz.

However, note that any pause may be cut short by pressing any key and that PAUSE 0 gives an unlimited pause that lasts until a key is pressed.

Format
PAUSE int-num-expr

PEEK

Keyboard location
EXTEND MODE
O

Function

PEEK gives the value of the byte stored at a particular address in the memory.

How to use PEEK
PEEK is followed by a numeric value, for example

```
80 LET x=PEEK (256*y)
```

Note that an expression must be *enclosed in brackets*. The value following PEEK is rounded to the nearest integer if necessary, and may then range from 0 to 65535 to give an address in the memory. PEEK then returns the value of the byte (a number from 0 to 255) at the address specified.

Example
The number of frames of the television display that have occurred since the Spectrum was last switched on is stored at addresses 23672 to 23674. As the frames are produced at a regular rate, PEEKing these locations gives a method of measuring time. The following line displays the time in seconds since the Spectrum was last powered up (less any time spent in producing sound and operating peripherals such as the cassette player and printer).

```
10 PRINT (PEEK 23672+256*
   PEEK 23673+65536*PEEK
   23674)/50
```

Note *If the mains frequency is 60 Hz and not 50 Hz (the UK standard), change 50 to 60.*

Format
PEEK int-num-const
PEEK int-num-var
PEEK (int-num-expr)

PI

Keyboard location
EXTEND MODE
M

Function

PI gives the value of *pi* (π) for use in calculations. PI is the ratio of the circumference of a circle to its diameter.

How to use PI
PI requires no values or variables when used in a statement or command, for example

```
DRAW 255,0,-PI
```

PI returns a value of 3.1415927, so that the above command draws a large semicircle on the screen.

Format
PI

PLOT

Keyboard location
Q

Statement/Command

PLOT is used in high-resolution graphics to plot a pixel or dot of colour at a particular position on the screen.

How to use PLOT
PLOT is used to form a statement in a program or as a command. It is normally followed by two numeric values

separated by a comma, for example

```
50 PLOT 128,87
```

Both values following PLOT are rounded to integers if necessary. The first value may then range from 0 to 255 and defines the horizontal coordinate of a position on the screen. The second value may range from 0 to 175 and defines a vertical coordinate. A pixel is then normally plotted in the current ink colour at the defined position – in the above example at the centre of the screen.

Note the following effects of colour statements or commands on PLOT. After OVER 1, an existing dot at the same position is changed to the paper colour. Following INVERSE 1, the dot is plotted in the current paper colour. After BRIGHT 1 or FLASH 1, the whole character position on the low-resolution screen in which the pixel is plotted will be bright or flash.

These four keywords and INK may also be embedded (inserted) within a PLOT statement in the same way as with PRINT, for example

```
160 PLOT INK 2;x,y
```

Their effect is the same but is then local and limited to the pixel plotted by the statement. If PAPER is embedded in a PLOT statement, the paper colour of the whole character position around the pixel changes to the given colour.

Note that PLOT also defines the starting position of the next DRAW statement.

Format
PLOT [statement;] int-num-expr,int-num-expr

POINT

Keyboard location
EXTEND MODE
SYMBOL SHIFT 8

Function

POINT is used to find out whether the colour at a particular position on the high-resolution screen is either an ink colour or a paper colour. POINT does not check the actual colour itself.

How to use POINT

POINT is followed by two numeric values *separated by a comma and enclosed in brackets*, for example

```
240 IF POINT (x,y)=1 THEN
    GOSUB 600
```

The two values following POINT are rounded to integers if necessary. The first value may then range from 0 to 255 and defines the horizontal coordinate of a pixel on the screen. The second value may range from 0 to 175 and defines a vertical coordinate. POINT then returns 1 if the pixel at the defined position is ink colour or 0 if it is paper colour.

Format
POINT (int-num-expr, int-num-expr)

POKE

Keyboard location
O

Statement/Command

POKE is used to change the value of the byte at a particular address in the memory. Values are normally POKEd to memory locations in order to produce actions not given by the BASIC keywords.

How to use POKE
POKE is used to form a statement in a program or as a command. It is followed by two numeric values *separated by a comma*, for example

```
POKE 23609,255
```

The two values following POKE are rounded to the nearest integers if necessary. The first value may then range from 16384 to 65535 and is an address in RAM. The second value may range from 0 to 255 and is the byte to be written to the defined address.

In the above example, 255 is POKEd to address 23609, which controls the sound produced when a key is pressed. A value of 255 gives a long beep instead of the normal click, other values producing a shorter beep.

Format
POKE int-num-expr, int-num-expr

PRINT

Keyboard location
P

Statement/Command

PRINT displays data on the screen. The data may be any single character or sequence of characters. A PRINT statement may incorporate other keywords to define the position and colour of the data.

How to use PRINT
PRINT may be used alone or it may be followed by data. This data may be in the form of any numeric or string expressions, or a mixture of these.

When using PRINT with data, two or more separate items must each be separated by a semicolon, comma or apostrophe.

Certain other keywords may be inserted in any order between PRINT and the data, provided each statement formed by the keyword *ends in a semicolon*. These keywords are CHR$, TAB, AT, INK, PAPER, FLASH, BRIGHT, INVERSE and OVER.

PRINT with strings
PRINT alone or followed by a null string (" ") displays a blank line and moves the cursor to the beginning of the next line.

PRINT followed by a string constant (any characters within double quote marks) displays the characters as they appear between the quote marks. The command

```
PRINT "3/542/76/21"
```

for example displays

```
3/542/76/21
```

PRINT followed by a string variable or expression displays the string or strings they represent.

PRINT with numbers
PRINT followed by any numeric expression displays the expression's value. Numbers are displayed in decimal notation with up to eight significant digits and no trailing zeros after the decimal point.

Very large and very small numbers are displayed in a shorter scientific notation, as two figures separated by the letter E. This indicates a number

in which the first part (the mantissa) is raised to the power of the second part (the exponent). The command

```
PRINT 3/542/76/21
```

for example displays

```
3.4680798E-6
```

PRINT formatting with punctuation signs
PRINT followed by items of data separated by a semicolon displays the items placed next to each other without a space. The command

```
PRINT 1;2;3
```

displays

```
123
```

PRINT followed by items of data separated by a comma displays each item at the beginning or in the middle of a line depending on the position of the first item. The command

```
PRINT 1,2,3
```

displays

```
1              2
3
```

PRINT followed by items of data separated by an apostrophe displays the item after the apostrophe at the beginning of the next line. The command

```
PRINT 1'2'3
```

displays

```
1
2
3
```

If a PRINT statement or command ends with a semicolon, comma or apostrophe, then the item displayed by the next PRINT statement is affected in the same way.

PRINT and other keywords
PRINT may be followed by TAB, a numeric value, a semicolon and then an item of data, for example

```
60 PRINT TAB x; a$
```

The value following TAB (x above) is rounded to the nearest integer if necessary and is then divided by 32 and the remainder returned to give a value between 0 and 31. The item of data is then displayed at this column position in the same or the next line.

PRINT may be followed by AT and then two numeric values *separated by a comma*, a semicolon and an item of data, for example

```
50 PRINT AT l,c; "Data"
```

The first value (l above) may range from 0 to 21 and defines the number of the line or row in which the data will be displayed. The second value (c) may range from 0 to 31 and defines the number of the column in which the first character or digit of the data will be displayed. Non-integer values are accepted and rounded to the nearest integer. The command **PRINT AT** 11,16;"*" displays a star in the centre of the screen.

PRINT may also be followed by one or more CHR$ functions. See CHR$ for further details.

PRINT and colour keywords
The display produced by PRINT is affected by colour statements or commands given by INK, PAPER, FLASH, BRIGHT, INVERSE and OVER that are currently in operation. PRINT may also be followed by one or more of these six statements *each followed by a semicolon* before the item of data, for example

```
50 PRINT AT 11,16; INK 2;
   FLASH 1;"*"
```

The item of data is then displayed with the attributes specified by the colour keyword(s). These attributes are local and apply only to the item displayed. Following execution of the PRINT statement, they revert to their default or previously declared global values. PRINT will also obey local colour control codes inserted with the data (see page 33).

Format
PRINT [**TAB** int-num-expr;] [**AT** int-num-expr, int-num-expr;] [**CHR$** (int-num-expr);] [statement;] [num-expr] [string-expr] [;] [,] [']

RANDOMIZE

Keyboard location
T

Statement/Command

RANDOMIZE, which appears on the keyboard as RAND, is used in conjunction with RND to produce sequences of numbers that are either random or predictable.

How to use RANDOMIZE
RANDOMIZE is used either to form a statement in a program or as a command. It is optionally followed by a numeric value, for example

```
RANDOMIZE 1
10 RANDOMIZE
```

The value following RANDOMIZE is rounded to the nearest integer if necessary and may then range from 0 to 65535. A value greater than 0 sets the system variable SEED to this value, following which RND always generates the same sequence of numbers (see page 48 for information on system variables). The actual sequence depends on the value of RANDOMIZE.

If RANDOMIZE is followed by 0 or no value, then SEED is given the value of another system variable called FRAMES, which counts the frames displayed on the television since the Spectrum was switched on. As SEED changes 50 or 60 times a second, the sequence of numbers generated by RND following RANDOMIZE or RANDOMIZE 0 is highly random.

If RANDOMIZE is not used, RND generates the same sequence of numbers from power up and after using the reset button or NEW.

Format
RANDOMIZE [int-num-expr]

READ

Keyboard location
EXTEND MODE
A

Statement/Command

READ is used in conjunction with DATA to assign values to variables using the values in a DATA statement.

How to use READ
READ is normally used to form a statement in a program. It is followed by one or more

numeric variables or string variables *each separated by a comma*, for example

```
20 READ a$,x
```

When READ is first executed, it takes the same number of values as there are variables from the start of the first DATA list and assigns the values to the variables in order. When READ is next executed, the next set of DATA values is assigned to the variables named in the READ statement and so on.

For further details, see DATA.

Format
READ num-var [,num-var] [,string-var]
READ string-var [,num-var] [,string-var]

REM REMark

Keyboard location
E

Statement

REM is used to put remarks or reminders into a program. These may be the title and author of the program, and explanations of lines in the program such as the purpose of a variable. The remarks play no part in the running of the program and can be seen only in the listing.

How to use REM
REM forms either a line of its own in a program or the last statement in a line. It is followed by any remark that can be keyed in as required, for example

```
80 INPUT n$: REM n$ is name
```

When the computer encounters REM, it ignores everything that follows REM in that line.

Format
REM any characters

RESTORE

Keyboard location
EXTEND MODE
S

Statement/Command

RESTORE is used in conjunction with READ and DATA to make READ take values from a particular DATA statement instead of the first or next DATA statement in the program.

How to use RESTORE
RESTORE normally forms a statement in a program. It is optionally followed by a numeric value, for example

```
160 RESTORE 800
```

The value following RESTORE is rounded to the nearest integer if necessary, and should then be the number of a line in the program containing a DATA statement. Following RESTORE, the next READ statement will assign the values contained in this DATA statement. If the numbered line does not exist or does not contain a DATA statement, then READ goes to the next DATA statement after this line.

If RESTORE is followed by 0 or no value, then the next READ statement goes to the first DATA statement in the program.

Format
RESTORE [int-num-expr]

RETURN

Keyboard location
Y

Statement/Command

RETURN is used to terminate a subroutine and return the computer to the main program or a previous subroutine.

How to use RETURN
RETURN is normally used to form a statement in a program. It is used alone at the end of a subroutine, for example

```
1080 RETURN
```

On execution, the program branches to the statement following the last GOSUB statement executed.

See GOSUB for further details.

Format
RETURN

RND RaNDom number

Keyboard location
EXTEND MODE
T

Function

RND is used to generate a random number.

How to use RND
RND is used alone in a statement or command, for example

```
60 LET x=RND
```

RND then returns a random number less than 1 and greater than or equal to 0.

When the Spectrum is switched on or reset, or NEW is used, numbers are subsequently returned by RND in the same sequence. The sequence is generated by taking the powers of 75 (75, 75*75, 75*75*75 and so on) dividing each power by 65537 and using the remainder only, then subtracting 1 from the remainder and dividing this result by 65536.

If a more random sequence or another fixed sequence is required, then use RANDOMIZE before RND.

Random whole numbers
Many of the Spectrum's statements and functions, such as INK and CHR$, round numbers to the nearest integer and RND may be used with them directly. INK RND*7, for example, produces an ink colour at random. Others require integers and any whole number from 1 to x is given by INT (RND*x)+1. To generate a random integer from 0 to x, use INT (RND*x+0.5).

Format
RND

RUN

Keyboard location
R

Command/Statement

RUN makes a program run, normally from the first line.

How to use RUN
RUN may be used as a direct command or it may form a statement in a program. It is optionally followed by a numeric value, for example

```
RUN 50
```

If no value follows RUN, then the program runs from the first line. If a value is included, it is rounded to the nearest integer if necessary and the program

then runs from this line. If the line does not exist, the program runs from the next line in the program. Note that RUN performs CLEAR before running the program, so that variable values are erased. To avoid this, use GOTO followed by a line number.

If a program has been saved using LINE, then it runs automatically on loading and RUN is not required.

Format
RUN [int-num-expr]

SAVE

Keyboard location
S

Command/Statement

SAVE sends a program to the cassette player in order to store it on tape.

How to use SAVE
SAVE is normally used as a direct command but may form a statement in a program. It is followed by a filename which is a string value, for example

SAVE "filename"

The filename may contain up to ten characters. On execution, the message

Start tape, then press any key

is displayed. On pressing any key, the program is sent to the cassette player and on conclusion, the report **0 OK, 0,1** appears.

Note that SAVE is used differently when a Microdrive is connected. See the Microdrive and Interface 1 manual for details.

Automatic running
If the stored program is to run automatically on loading, then SAVE should be used in conjunction with LINE. The program name is followed by LINE and a numeric value, for example

SAVE "filename" **LINE 1**

The value following LINE is rounded to the nearest integer if necessary, and should then be either 1 or the number of a line in the program. The program is then sent to tape in the same way as SAVE. On loading, the program runs automatically from the line having the defined number or, if no such line exists, from the next line in the program. In practice, using LINE 1 causes the whole program to run automatically.

Format
SAVE string-expr [**LINE** int-num-expr]

SAVE CODE

Keyboard locations
S
EXTEND MODE
I

Command/Statement

SAVE CODE sends a section of information in the memory to the cassette player to be stored on tape. The information can then be placed back in the memory using LOAD CODE.

How to use SAVE CODE
SAVE CODE may be used as a direct command or to form a statement in program. SAVE is followed by a filename which is a string value and then CODE, which is in turn followed by two numeric values *separated by a comma*, for example

SAVE "picture" **CODE**
16384,6912

The filename following SAVE may contain up to ten characters. The two values following CODE are each rounded to the nearest integer if necessary. The first then gives the starting address (16384 above) of the information in the memory, and the second value (6912) gives the number of bytes that are to be stored. The information is then sent to tape in the same way as a program with SAVE.

The information saved by the above command is the screen display.

Format
SAVE string-expr **CODE**
int-num-expr,int-num-expr

SAVE DATA

Keyboard locations
S
EXTEND MODE
D

Statement/Command

SAVE DATA stores an array on tape. The array can then be loaded using LOAD DATA.

How to use SAVE DATA
SAVE DATA may be used to form a statement in a program or as a direct command. SAVE is followed by a filename, then DATA, a letter or letter with $, and finally by a pair of *empty* brackets, for example

```
450 SAVE "numbers" DATA
    n()
750 SAVE "names" DATA n$()
```

The array's filename may contain up to ten characters. The letter or letter$ following DATA is the name of the array in the program that is to be stored on tape. They array is then sent to tape in the same way as a program with SAVE.

Format
SAVE string-expr **DATA** letter [$]()

SAVE SCREEN$

Keyboard locations
S
EXTEND MODE
SYMBOL SHIFT K

Command/Statement

SAVE SCREEN$ stores the screen dispay on tape. It can be loaded back into the computer at a later date using LOAD SCREEN$.

How to use SAVE SCREEN$
SAVE SCREEN$ may be used as a direct command or to form a statement in a program. SAVE is followed by a filename which is a string value and then SCREEN$, for example

SAVE "picture" **SCREEN$**

The filename may have up to ten characters. The display is then sent to tape in the same way as a program with SAVE.

Format
SAVE string-expr **SCREEN$**

SCREEN$

Keyboard location
EXTEND MODE
SYMBOL SHIFT K

Function

SCREEN$ detects which

character appears at a particular position on the screen.

How to use SCREEN$
SCREEN$ is followed by two numeric values *separated by a comma and enclosed in brackets*, for example

```
160 IF SCREEN$ (l,c)="X"
    THEN PRINT "CRASH"
```

The values following SCREEN$ are rounded to the nearest integer if necessary. The first value (l above) may then range from 0 to 21 and gives the line number of a position on the screen. The second value (c above) may range from 0 to 31 and gives the column number of the position. SCREEN$ then returns the character displayed at this position as a string constant (the character in quote marks, "X" above for example). If no character is present at the position, SCREEN$ returns a null (empty) string ("").

Note that SCREEN$ may also be used with SAVE and LOAD to store the screen display on tape and load it from tape. See SAVE SCREEN$ and LOAD SCREEN$ for details.

Format
SCREEN$ (int-num-expr, int-num-expr)

SGN SiGN

Keyboard location
EXTEND MODE
F

Function

SGN indicates whether a number is positive, negative or zero.

How to use SGN
SGN is followed by a numeric value, for example

```
50 LET x=SGN y
```

An expression must be *enclosed in brackets*. SGN then returns 1 if the value of the argument (y above) is positive, −1 if it is negative and 0 if it is zero.

Format
SGN num-const
SGN num-var
SGN (num-expr)

SIN SINe

Keyboard location
EXTEND MODE
Q

Function

What SIN does
SIN gives the sine of an angle.

How to use SIN
SIN is followed by a numeric value, for example

```
80 LET x=SIN y
```

An expression must be *enclosed in brackets*. The value following SIN is the angle *in radians*, and SIN returns the sine of the angle. Degrees can be converted into radians by multiplying by PI/180.

Note that SIN returns a positive value for angles between 0 and 180 degrees, and a negative value for angles between 180 and 360 degrees.

Example
The command

```
PRINT SIN (30*PI/180)
```

displays 0.5, the sine of 30 degrees.

Format
SIN num-const
SIN num-var
SIN (num-expr)

SQR SQuare Root

Keyboard location
EXTEND MODE
H

Function

SQR gives the square root of a number.

How to use SQR
SQR is followed by a numeric value, for example

```
70 LET x=SQR y
```

An expression must be *enclosed in brackets*. The value following SQR (y above) must be greater than zero, and SQR returns its square root.

Format
SQR num-const
SQR num-var
SQR (num-expr)

STEP

Keyboard location
SYMBOL SHIFT D

See **FOR**

STOP

Keyboard location
SYMBOL SHIFT A

Statement/Command

STOP halts a program at a particular point. It may be necessary to use STOP to end the main section of a program in order to confine subroutines to a separate section. STOP is also valuable in debugging a program.

How to use STOP
STOP is normally used to form a statement in a program. It is used on its own, for example

```
650 STOP
```

On execution, the program stops and the report

9 STOP statement

appears with the line and statement number at which the program halted.

Debugging procedures, such as displaying and changing the values of variables, may then be undertaken. Entering CONTINUE subsequently causes the program to resume at the next statement with the new values.

Format
STOP

STR$

Keyboard location
EXTEND MODE
Y

Function

STR$ converts a number into a string.

How to use STR$
STR$ is followed by a numeric value, for example

```
90 LET a$=STR$ x
```

An expression must be *enclosed in brackets*. STR$ then returns the value of its argument (x above) as a string constant. If x were assigned the value of 65, then the above statement assigns a$ the value of "65".

Format
STR$ num-const
STR$ num-var
STR$ (num-expr)

TAB

Keyboard location
EXTEND MODE
P

See **LPRINT, PRINT**

TAN TANgent

Keyboard location
EXTEND MODE
E

Function

TAN gives the tangent of an angle.

How to use TAN
TAN is followed by a numeric value, for example

```
130 LET x=TAN y
```

An expression must be *enclosed in brackets*. The value following TAN is the angle *in radians*, and TAN returns the tangent of the angle. Degrees may be converted into radians by multiplying by PI/180.

Note that TAN returns a positive value for angles between 0 and 90 degrees and between 180 and 270 degrees. For angles between 90 and 180 degrees and angles between 270 and 360 degrees, TAN returns a negative value.

Format
TAN num-const
TAN num-var
TAN (num-expr)

THEN

Keyboard location
SYMBOL SHIFT G

See **IF**

TO

Keyboard location
SYMBOL SHIFT F

Function

TO has two different uses in Sinclair BASIC. It is used in conjunction with FOR to set up a FOR NEXT loop (see FOR for details) and it is also used in string slicing (the splitting up of strings into smaller substrings).

How to use TO for string slicing
TO is used to define the first and last characters of a substring within a main string. TO is preceded by a string value, an opening bracket, then an optional numeric value. It is followed by another optional numeric value and then a closing bracket, for example

```
80 PRINT a$ (4 TO 7)
```

A string expression must also be *enclosed in brackets*. The string value (a$ above) is the string that is to be sliced. The two numeric values (4 and 7) define the positions of the first and last characters of the substring within the string. TO then returns the substring (characters 4 to 7 of a$).

The first numeric value has a default value of 1 and the second has a default value equal to the position of the last character in the string. The first value can therefore be omitted if the substring begins with the first character in the string, and the second value can be omitted if the substring ends with the last character in the string.

Format
string-const ([num-expr] **TO** [num-expr])
string-var ([num-expr] **TO** [num-expr])
(string-expr) ([num-expr] **TO** [num-expr])

USR User SubRoutine

Keyboard location
EXTEND MODE
L

Function

USR is used to call a machine code subroutine that has been placed in the memory at a specific address. It is also used to place the data for user-defined graphics in the reserved locations at the top of the memory.

USR and machine code
To use machine code, USR is followed by a numeric value, for example

```
 80 PRINT USR 65000
100 RANDOMIZE USR 65000
```

An expression should be *enclosed in brackets*. The value following USR is rounded to the nearest integer and is then the starting address in the memory at which a machine code subroutine has been placed. Any statement containing USR then calls the subroutine at this address and USR returns the value of the contents of the bc register pair. RANDOMIZE USR or RESTORE USR, for example, runs the subroutine only, whereas PRINT USR additionally displays the bc register value.

USR and user-defined graphics
To create user-defined graphics, USR is used with POKE. It is followed by a string constant or variable to return an address for the POKE statement, for example

```
50 POKE USR "a", 255
```

The string value following USR may be a single letter ranging from A to U or a to u, capital letters not being distinguished from lower-case letters.

USR then returns the starting address of one of the 21 sections of the memory reserved for user-defined graphics. Each section contains eight addresses to which eight bytes are POKEd to create one graphics character. The bytes may be given in decimal form or in binary form (see BIN).

Format
USR int-num-const
USR int-num-var
USR (int-num-expr)
USR string-const
USR string-var

VAL VALue

Keyboard location
EXTEND MODE
J

Function

VAL changes a string with a numeric value into a number.

How to use VAL
VAL is followed by a string constant or variable, for example

```
70 LET x=VAL a$
```

The value of the string constant or variable is stripped of its quote marks, and must then be a numeric value. VAL evaluates this, returning it as a numeric constant.

Examples
If a$ has the value "435", then the above statement assigns a value of 435 to x. However, VAL can also evaluate expressions, for example

```
10 INPUT a$,x
20 PRINT VAL a$
```

The string value that is assigned to a$ should be an expression using x, for example "x*x". A numeric value is then assigned to x, for example 5. VAL strips the quotes of the string value to get x*x and evaluates it using the value assigned to x, displaying the result 25.

Format
VAL string-const
VAL string-var

VAL$ VALue (string)

Keyboard location
EXTEND MODE
SYMBOL SHIFT J

Function

VAL$ evaluates a string as a string expression.

How to use VAL$
VAL$ is followed by a string variable, for example

```
130 PRINT VAL$ a$
```

The value of the string variable is stripped of its quote marks, and must then be a string expression. VAL$ evaluates the expression and returns the value as a string constant.

Examples
Try this program

```
10 INPUT a$,x$
20 PRINT VAL$ a$
```

The string value that is assigned to a$ should be an expression using x$, for example "x$+x$". A string value is then assigned to x$, for example "DO". VAL$ strips the quotes of the value of a$ to get x$+x$ and evaluates it using the value assigned to x$, displaying the result DODO.

Format
VAL$ string-var

VERIFY

Keyboard location
EXTEND MODE
SYMBOL SHIFT R

Command/Statement

VERIFY checks that a program has been correctly stored on tape following SAVE.

How to use VERIFY
VERIFY is normally used as a direct command in exactly the same way as LOAD and is followed by the program name, for example

```
VERIFY "filename"
```

When the tape is started, the name of every program found is displayed and any program on tape having the same name is compared with the program in the memory. If the two are found to be the same, the report

```
0 OK, 0:1
```

is given.
VERIFY is used differently when a Microdrive is connected. See the Microdrive and Interface 1 manual for details.

VERIFY CODE and VERIFY DATA
VERIFY CODE can be used in exacty the same way as LOAD CODE to verify that a section of memory information has been stored on tape. VERIFY DATA works in the same way as LOAD DATA to check that an array has been stored on tape. See LOAD CODE and LOAD DATA for further details.

Format
VERIFY string-expr
VERIFY string-expr **CODE** [int-num-expr] [,int-num-expr]
VERIFY string-expr **DATA** letter[$]()

VERIFY CODE

Keyboard location
EXTEND MODE
SYMBOL SHIFT R
EXTEND MODE
I

See **VERIFY**

VERIFY DATA

Keyboard location
EXTEND MODE
SYMBOL SHIFT R
EXTEND MODE
D

See **VERIFY**

ZX SPECTRUM+ SCREEN REPORTS

When the Spectrum stops execution of BASIC, a report appears at the bottom of the screen. This indicates that a command or program has been completed or that an error has occurred. Each report consists of a code number or letter followed by a brief message and then the numbers of the line and statement at which the computer stopped. A command is shown as line 0 and within a line, statement 1 is at the beginning of the line, statement 2 is after the first colon or THEN, and so on. CONTINUE normally causes the program to resume at the statement specified in the report.

0 OK
Successful completion, or an attempt to jump to a line number greater than any in the program. CONTINUE disregards this report and resumes at the statement specified in the previous report.

1 NEXT without FOR
NEXT has been encountered without a corresponding FOR *and* a variable exists with the same name as the control variable.

2 Variable not found
A simple variable has been used without assigning it a value or loading the value from tape, or a control variable has been used with NEXT without first setting it up in a FOR statement, or a subscripted variable has been used before dimensioning the array with DIM or loading an array from tape.

3 Subscript wrong
A subscript is beyond the dimensions of the array.

4 Out of memory
There is not enough memory space left to complete the statement or command.

5 Out of screen
INPUT has generated more than 23 lines in the lower part of the screen, or a line number of 22 or more has been used with PRINT AT.

6 Number too big
The computer has tried to produce a number greater than about 10^{38}.

7 RETURN without GOSUB
The number of RETURN statements is one greater than the number of GOSUB statements.

8 End of file
Microdrive file-handling report.

9 STOP statement
STOP has been used to halt the program. CONTINUE will resume at the next statement.

A Invalid argument
A function has been given a wrong argument or value.

B Integer out of range
A value has been rounded to the nearest integer and is out of the range that can be accepted.

C Nonsense in BASIC
The text of the (string) argument does not form a valid expression.

D BREAK – CONT repeats
BREAK has been pressed. CONTINUE will repeat the statement at which the computer stopped.

E Out of DATA
READ has tried to read beyond the end of the final DATA statement in the program.

F Invalid file name
SAVE has been used with a name containing more than ten characters.

G No room for line
There is not enough memory space left to enter the new program line.

H STOP in INPUT
STOP has been entered in response to INPUT or began the data entered. CONTINUE repeats the INPUT statement.

I FOR without NEXT
A FOR NEXT loop has not been carried out because the limits or STEP value were wrong (for example, FOR x=5 TO 0 without STEP) *and* the corresponding NEXT has not been found.

J Invalid I/O device
Microdrive file-handling report.

K Invalid colour
The value specified for INK, PAPER, FLASH, BRIGHT, INVERSE or OVER or the corresponding control character is out of range.

L BREAK into program
BREAK was pressed. The report specifies the last statement to be executed and CONTINUE resumes at the next statement.

M RAMTOP no good
The value specified for RAMTOP is either too big or too small.

N Statement lost
A jump has been attempted to a statement that no longer exists.

O Invalid stream
Microdrive file-handling report

P FN without DEF
An FN statement has been used without the corresponding DEF FN statement.

Q Parameter error
An FN statement contains the wrong number of values to be passed to the function, or one of the values is the wrong type (a string instead of a number or vice versa).

R Tape loading error
The loading, merging or verification procedure has failed.

BEYOND BASIC

BASIC is an all-purpose computer language which works very well for most applications. However, it is not the only computer language that you can use on the Spectrum. Software that provides other languages, such as FORTH, micro-PROLOG and LOGO, is available. These languages work in a very different way to BASIC and open up new possibilities for your computer.

Because BASIC is an all-purpose language, it can be rather cumbersome in some applications. It is also comparatively slow. Other languages can give greater flexibility combined with simplicity of programming and faster running speed. FORTH, for example, allows you to define your own words and use them in the instructions that the computer understands and which it executes about ten times as fast as the equivalent commands in BASIC. With micro-PROLOG, the computer will understand simple English phrases. It stores these in its memory as a basis for dialogue with the user. LOGO is a computer language developed for educational use. It features very simple commands which can be used in a highly flexible way. However, if you want to write really fast programs for your ZX Spectrum +, you will need to understand how to program in machine code.

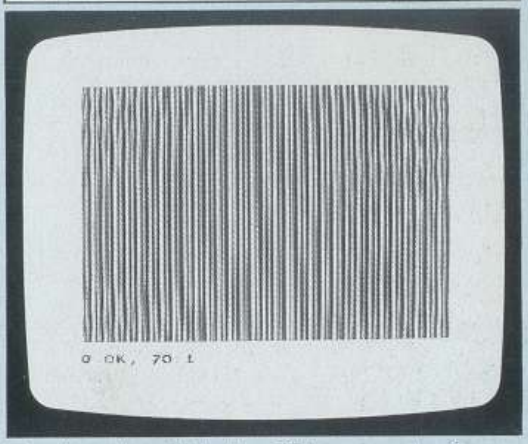
Machine code

BASIC is used to enable you to give instructions to the computer in a form that is easy for you to understand. The Spectrum's CPU – the powerful Z80A chip – does not in fact understand BASIC. A section of the memory contains a permanent program called the BASIC interpreter that converts your BASIC instructions into a sequence of code signals. These codes actually drive the Z80A to perform your instructions.

The interpreter takes some time to translate your BASIC instructions into the Z80A code or machine code, as it is called. However, you can bypass the BASIC interpreter if you wish and send machine code directly to the Z80A. Your program will then be carried out very quickly. The price that has to be paid for this is the extra time needed to write the machine code program. Unlike BASIC, it is a very 'unfriendly' language, meaning that it takes quite a time to learn. Machine code programming is outside the scope of this guide. However, there are many books available which teach Spectrum machine code to an advanced level. To get just a taste of the speed of machine code, key in and run this short demonstration program.

INSTANT STRIPES

```
 10 FOR x=0 TO 15
 20 READ n: POKE 65000+x,n
 30 NEXT x
 40 DATA 33,255,63,1,1,24,22
 50 DATA 55
 60 DATA 35,11,120,177,200,114,
24,248
 70 RANDOMIZE USR 65000
```



Try changing 55 in line 50 to any value from 1 to 255 and see how the stripes change. You can also produce coloured stripes by using an INK command first. However, this is not the point of the program. See how *fast* the display is produced by using machine code – it is virtually instantaneous. BASIC takes over two seconds to fill the screen.

This program works because the DATA statements contain 16 codes that are stored at address 65000 onwards in the memory by lines 10 to 30. Line 70 sends the codes to the Z80A and the display is produced right away. Note that the eighth code controls the width of the stripes.

Many of the games that are available for your Spectrum are written in machine code to produce this ultra-fast action. To help you write machine code, programs called assemblers are available. These provide instructions that you key in instead of just numbers, which is what machine code itself requires. The instructions are not English words like the keywords BASIC, but abbreviations or mnemonics which represent the operations that the computer must undertake. You must therefore have an understanding of how the computer works on a step-by-step basis before you can use assembly language.

COMPUTER JARGON – WHAT IT MEANS

Many words that are used in computing are also used in everyday life but with different meanings. Here are explanations of some of these words that appear in this book, together with special computing terms. Words in *italics* have explanations themselves. If there is a word or term in the book that you can't understand and it is not here, try looking it up in the index.

Address A single unit of the *memory*. There are 65536 addresses in the ZX Spectrum +.

Argument A *value* that is used by a *function* to get a result.

Array A group of related *data* that is held together in one section of the *memory*.

Attributes Codes that give the colours of *characters*.

BASIC The computer language used by the ZX Spectrum + and most other home microcomputers.

Binary code The kind of code that is used by computers. It consists of sequences of on or off states – for example, on-off electric pulses.

Bit An on or off state in *binary code*. Short for binary digit.

Byte A set of eight *bits* that represents a number having a value from Ø to 255. Each *address* in the *memory* holds one byte.

Character Any single letter, numeral (Ø to 9), sign or *graphics* unit that can be displayed or printed.

Character set The complete set of preset *characters* and certain control codes used by the computer.

Command A single instruction that is carried out by the computer, or a *direct command*.

Concatenation The combining of *strings* by adding them together.

Constant A number or a group of one or more letters or any other *characters*.

CPU (Central Processing Unit) The central part of the computer that does the computing and controls the other units. The ZX Spectrum + uses a Z80 microprocessor.

Cursor The position on the screen where something is to be displayed next. It may be marked by a flashing sign indicating the *mode* that the computer is in.

Data *Information* that the computer either gets from a *program* or that is fed into the computer in order to produce results.

Direct command A set of one or more instructions that is carried out immediately it is given to the computer.

Edit To change details within a *program*.

Enter To give a completed instruction or information to the computer.

Expression A combination of *constants*, *variables* and *keywords*.

False Any state or result that the computer decides is untrue or incorrect. False has a numeric value of Ø.

Function An operation in which the computer takes one or more *values* (or *arguments*) and uses them to give a result that is another value.

Graphics The production of images such as pictures, charts or diagrams by the computer.

Hardware The computer itself and any associated devices or machines, such as *peripherals*.

Information Words, numbers and signs in any combination that the computer is required to handle.

Input *Programs* and *data* fed into the computer.

Interface A unit that connects the computer and/or *peripherals* together and which ensures that they can communicate with each other.

K A measure of the *memory* capacity of a computer. 1K is equal to 1 kilobyte or 1024 bytes. The memory capacity in K is equal to the total number of *addresses* in the memory, each of which can store one byte. The ZX Spectrum + has a 48K RAM and a 16K ROM, giving a total of 64K.

Keyword A computer instruction in *BASIC*. It may require some *values* to work.

Line An instruction or set of instructions in a *program*. It is given a number so that it is carried out at the correct point in a sequence of other lines.

Listing The *lines* of a *program* listed in order.

Load To feed a *program* or *data* into the computer from a storage device such as a cartridge or cassette.

Logic The process by which the computer decides whether results are right or wrong, or states are *true* or *false*.

Loop A section of a *program* that is repeated one or more times.

Machine code The language that the ZX Spectrum + understands. Programs in *BASIC* are translated into machine code by the computer as it runs them.

Memory The part of the computer that holds the *program* and *data* when required, and also the permanent operating instructions.

Mode In the Spectrum, one of five states which dictate which *keywords* and *characters* can be produced by each key on the keyboard. During programming, mode is indicated by a flashing letter within the *cursor*.

Nesting The arrangement of *loops* within a program so that one or more loops are carried out within another.

Numeric variable A variable that holds a number. Numeric variables consist of one or more letters.

Operator An instruction that performs arithmetic or *logic*.

Output Results produced by the computer.

Peripheral Any device that is connected to the computer.

Pixel The smallest dot of colour that can appear on the screen. Short for 'picture cell'.

Print Either to display results or *graphics* on the screen or to print them on a printer.

Program A sequence of instructions to be carried out by the computer.

RAM (Random Access Memory) The part of the *memory* that can be given a *program* and *data*, and other changing *values*. Also known as volatile memory. RAM contents are erased when power is disconnected. The ZX Spectrum + has a 48K RAM.

Register A small memory unit separate from the main *memory*. Registers within the *CPU* are used to carry out the process of computing.

Report A message displayed by the computer reporting its actions.

Resolution The degree of detail possible in computer *graphics*.

ROM (Read Only Memory) The part of the *memory* containing permanent *programs* and instructions for the computer. The ZX Spectrum + has a 16K ROM.

Save To store a *program* or *data* in a storage device such as a cartridge or cassette.

Scroll The movement which enables a display that exceeds the size of a single screen to be viewed.

Software Any *program*, including permanent programs in *ROM* or cartridges.

Statement Either a *keyword* that is used to form an instruction in a program *line*, or the instruction itself.

String A group of one or more *characters* enclosed in quotes to distinguish them from numbers and *numeric variables*.

String variable A *variable* that holds a *string*. String variables always consist of a single letter and the $ sign.

Syntax The correct sequence of *keywords, constants, variables* and *expressions* required to form a valid *BASIC* instruction.

True Any state or result that the computer decides is true or correct. True has a numeric value of 1.

Value Any number or a *string* that may be given or represented by a *constant, variable* or *expression*.

Variable One or more units of the *memory* that hold a particular *constant* for use by the computer. Each is given a name or letter to identify it easily. The ZX Spectrum + distinguishes between *numeric variables* and *string variables*.

INDEX

Page numbers in *italic* refer to illustrations and captions.

Aerial socket and leads *4-5*
Altering programs 9
Amplifying sound 37
Animation 34-5
Arithmetic operators 22; *22*
ATTR 35

Bar charts 25; *25*
BASIC 18, 49-73
BEEP 36; *18*
BIN 33
Binary code *44*
Border colour 24-5; *6*
Bouncing ball program 35
Brackets 23
BREAK *19*
BRIGHT 31

Calculations 22-3; *22, 23*
Capitals mode 21; *20*
CAPS LOCK 21; *18*
CAPS SHIFT 8, 21; *18*
Cartridges, Microdrive 12, 46; *46*
  ROM 12, 47; *47*
Cassette players, as amplifiers 37; *37*
  choosing 12
  connections 5, 13; *13*
  counters 14
  loading programs 14-16
  saving programs 38-40
  tone controls 14, 15, *16*
  volume controls 14, 15, *16*
Cassette tapes 12, *44, 45*
  care of 12
  labelling 14
  sound of 12
  storage 12
Central Processing Unit (CPU) 43, 44, 48, 75; *43, 45*
Characters, creating 32-3
  selecting 20
Character set *51*
Chessboard program 33
Chips *42-3*
CIRCLE 28
Collisions 34-5
Colon 23, 51
Colour 24-5; *24-5*
  codes *24*
  control codes *33*
  combinations 25
  display keys *19*
  mixed 32
  testing 6; *24*
Comma 23, 51
Commands 22, 50
Connections 5
  cassette player *13*
  power 5
  television *4*
Cursor controls *19*

DATA 33
DELETE 10
DRAW 28-9

EAR Socket 37; *5, 13*
Edge connector *5, 43, 47*
EDIT *18, 21*
  program lines 21
ENTER *9, 10, 11, 19*
Entering programs 8-9
Errors, correcting 10, 21
  screen reports 74
EXTEND MODE 8, 21; *18*
  extended mode 21; *20*

FLASH 31
Flashing circles program 9
FOR NEXT 26-7, 29, 30, 31, 34
FORTH 75
Full stop 23, 51
Functions 50

GOTO 23
GRAPH 21; *18, 26*
Graphics, animation and, 34-5
  colour 24-5
  creating characters *32-3*
  filling in shapes 29; *29*
  high-resolution 26, *28-9*
  low-resolution *26-7*
  patterns *30-1*
  random effects 30
Graphics mode 21; *20*
Grid, high-resolution 28, *80*
  low-resolution 26, *80*
Hardware, definition 12
High-resolution graphics 26, *28-9*

IF THEN 29
Ink colour 24-5
INPUT 23, 29
Input-output pathways 45
Interfaces *45, 46-7*
INV VIDEO *18*
INVERSE 31

Joysticks 45, 47

Keyboard *18-19*
  graphics characters *26*
  modes 20-1
Keying in 8, 9
Keys 18-19; *18-19*
  operating 20-1; *20-1*
Keyword mode 20; *20*
Keywords 9, 18-19, 50, 52-73 20-1
  selecting 19, 20

LET 23
Letter mode 21; 20
Lines 8
  deleting 21
  editing 21
LIST 21
Listings 8, 21
LOAD 14-16
Loading 13, 14-15; *14-16*
Logic chips *43*
LOGO 75
Loops 26-7, 30
Loudspeaker 43
Low-resolution graphics *26-7*

Machine code 75
Manic mosaic program *10*
Memory 12; *42, 43, 44-8*
Memory map *48*
MIC socket 37; *5, 13*
Microdrives 46; *5, 46*
  cartridges 12, 45
  loading *46*
Micro-PROLOG 75
Mistakes, correcting 10, 21
Modems 46
Modes 20-1
Multiplication table program *23*
Music 36-7

Names program *8*
NEW 11, 12; *18*
New programs 11
9VDC socket *5, 43*
Number keys *19*
Numbers 50

Paper colour 24-5
Patterns program *9*
Peripherals *45, 46-7*
Pictures, designing *30-1*
  low-resolution *26-7*
Pitch, musical *36*
Pixels 28
PLOT 28
Point 23, 51
POKE 48
Polyhedra program *10*
Power supply 4, 5; *5, 43*
PRINT 22
Printers 45, 47; *45, 47*
Program lines, deleting 21
  editing 21
Programming 17-40
Programs, altering 9
  beginning new 11
  correcting mistakes 10
  entering 8-9, 44
  loading 12, 13, 14-15; *14-15*
  restarting 10
  running 8-9, 44
  saving 13, 38-40
  verifying 39
Punctuation signs 23, 51
Pyramids program *31*

Quote mark 23, 51

Radio interference 4
Rainbow program *26-7*
RAM (Random Access Memory) 42, 48; *42, 45*
RAM packs 4

RAMTOP *48*
Random effects 30
READ 33
Ready-to-run software 12-13; *13*
REM 39
Reset button 11, 12; *5*
Restarting programs 10
Ribbon cable 46
RND 26,30
ROM (Read Only Memory) 48; *43,45*
ROM cartridges 12, 47; *46-7*
RS232 interface 47; *45*
Running programmes 8-9

SAVE 38-9
Saving 13, 38-40
Screen reports 74
Scrolling 8
Semicolon 23, 51
Shapes, filling in 29; *29*
Shimmering sunrise program *11*
Signs, calculations 22, 50
  selecting 19
Sinclair BASIC 49-73
Sketchpad program *29*
Sockets 5
Software 12
  loading 14-16; *14-16*
  ready-to-run 12-13; *13*
  suitability 12
  types 12
Sound effects 36-7
Space bar *19*
Squares program *30*
Star program *28*
Stars and stripes program *11*
Statements 22, 50
STEP 29
Storage 44, 45
Strings 22
Subroutines 30-1
SYMBOL SHIFT 8, 21; *19*
Symbols, selecting 20
Symmetrical patterns program *30*
System variables 48

Tapes 12, 45
  care of 12
  labelling 14, 39
  sound of 12
  storage 12
Television, connecting *5*
  suitability 4
  tuning 6; *6*
Tone controls, cassette player 12,14,15
TRUE VIDEO *18*
Tuning television sets 6; *6*
TV encoder *42*

Uncommitted Logic Array (ULA) 42

User-defined characters 80; *32-3*

Variables 22-3, 50
Voltage regulator *43*
Volume controls, cassette player 12,14,15

Z80 microprocessor 43, 75; *45*
ZX Interface 1 45, 46-7
ZX Robot program *27*
ZX 16K RAM 4

First published 1984 by Dorling Kindersley Ltd, 9 Henrietta Street, London WC2E 8PS in association with Sinclair Research Ltd, 25 Willis Road, Cambridge

Second printing 1984

British Library Cataloguing in Publication Data

Ardley, Neil
ZX Spectrum + User Guide
1. Sinclair ZX Spectrum + (Computer)
I. Title
001.64'04 QA77.8.S625

ISBN 0-86318-080-9

Editor David Burnie
Art Editor Peter Luff
Designer Debra Lee
Photographer Trevor Melton
Screen-shot photographer Vincent Oliver
Managing Editor Alan Buckingham

Typesetting by The Letter Box Company (Woking) Ltd, Woking, England
Reproduction by A. Mondadori, Verona
Printed and bound in Italy by A. Mondadori, Verona

Screen display grids

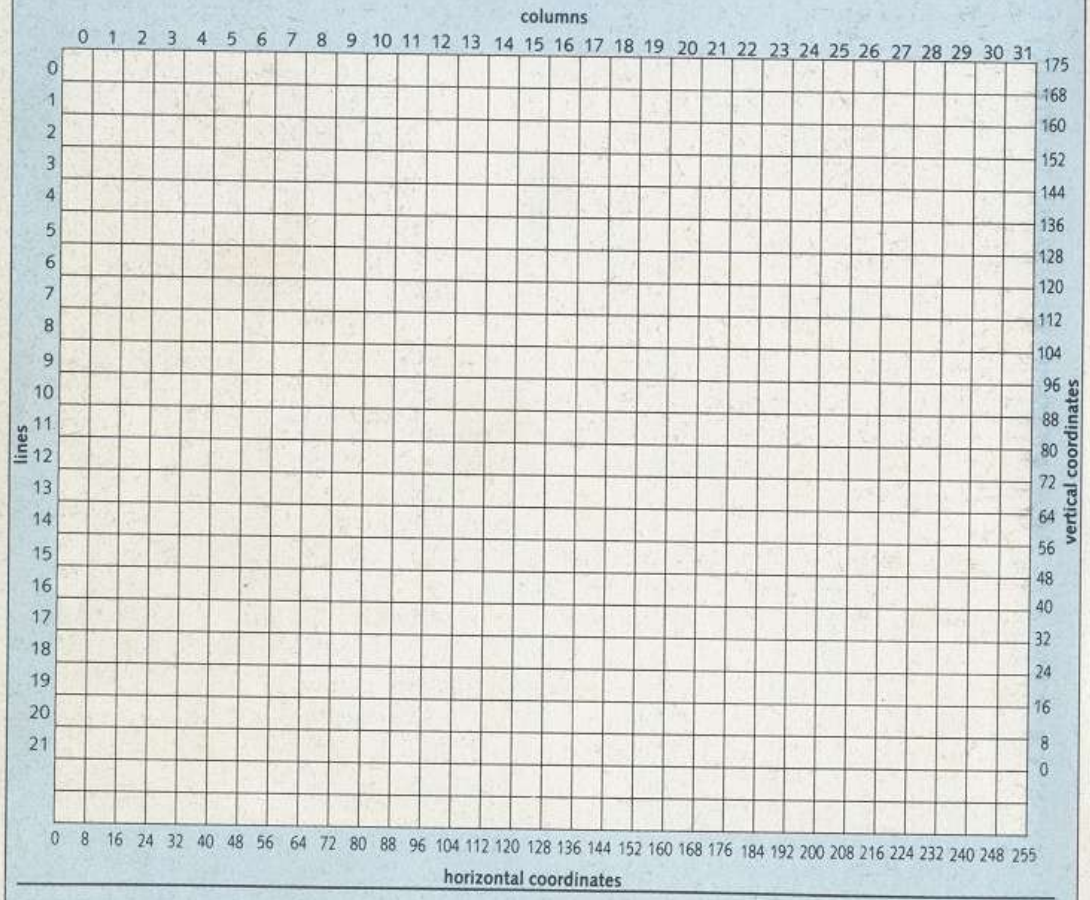
The low- and high-resolution screen

The grid below shows coordinates for both low-resolution and high-resolution graphics. Although individual graphics keywords work with either low- or high-resolution coordinates only, you can use both low- and high-resolution keywords when producing a display.

The low-resolution grid is divided into two parts, the main display area and then two lines at the bottom of the screen. PRINT AT will produce characters in the upper part and INPUT AT in the lower part. The coordinates for the low-resolution screen are shown on the top and left sides of the grid.

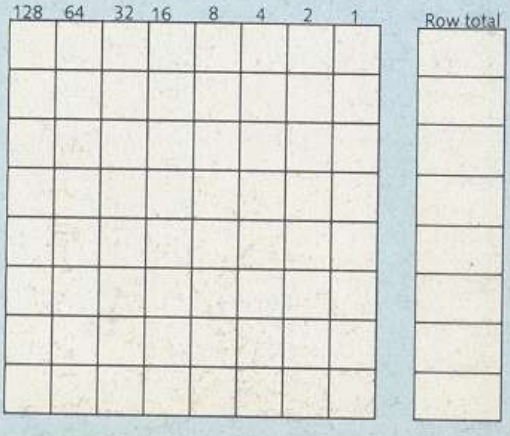
The high-resolution grid occupies only the main display area. PLOT, DRAW and CIRCLE are used to produce high-resolution graphics. The coordinates for high resolution are shown on the bottom and right sides of the grid.

In low resolution, each square on the grid is treated as a single unit. In high resolution, each of the 64 pixels that make up the square can be separately controlled. The pixels can also be programmed individually to make up a user-defined graphics character (see bottom grid).



User-defined graphics grid

To define a character, pencil in a design on this grid using whole squares only. Then, working row by row, add together the numbers above each square that you have filled in. Each row total should then be noted in the column on the right. Repeat this until you have a total for each row on the grid. Then, using the techniques on pages 32–33, you can program the computer to use these numbers to produce a character.



	100 Programs for the ZX Spectrum C	2501	£12.50	

*All prices include VAT (where applicable) an

Please send me the titles listed above. I enclose [redacted] der for £_____
made payable to Dorling Kindersley Ltd.

TOTAL £

(BLOCK LETTERS PLEASE)

Mr/Mrs/Miss/Ms _____

Address _____

_____

_____ Postcode _____ Signature _____

Please allow 28 days for delivery. Offer applies to UK and Eire only.

ORDER FORM

SPECTRUM SOFTWARE

The entire range of software available for Spectrum computers (including all existing titles) is completely compatible with your new ZX Spectrum+.

ISBN 0-86318-080-9

DORLING KINDERSLEY LTD
in association with
SINCLAIR RESEARCH LTD

£4.95

9 780863 180804