ISSN 0265-2919
R 80p

THE HOME COMPUTER COURSE 8

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

An ORBIS Publication
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

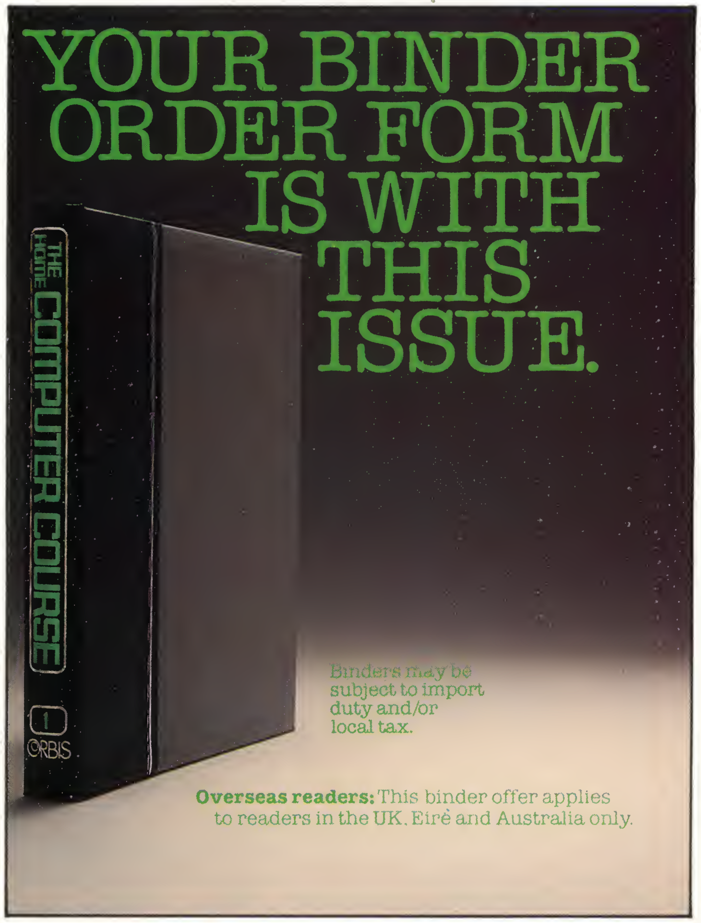
CONTENTS

Next Week

- Portable computers range from pocket- to small suitcase-size, but all offer a real measure of computing power. We take a look at this interesting new development

- Have you ever wanted to get inside a story, and take control? Adventure Games allow you to do just that — but watch out that the dragon doesn't capture you!

- We review Epson's HX-20 portable computer, the first machine to offer complete BASIC with a full range of commands and a built-in printer and cassette drive


YOUR BINDER ORDER FORM IS WITH THIS ISSUE.
THE HOME COMPUTER COURSE
1
ORBIS
Binders may be subject to import duty and/or local tax.
Overseas readers: This binder offer applies to readers in the UK, Eire and Australia only.

Editor Richard Pawson; **Art Editor** David Whelan; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistants** Steve Broadhurst, Liz Dixon; **Sub Editor** Teresa Bushell; **Researcher** Melanie Davis; **Consultant Editor** Gareth Jefferson; **Contributors** Tim Heath, Roger Ford, Richard King, Trevor Jones, Boris Sedacca, Mel Pullen; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Consultant** David Tebbutt; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; © 1983 by Orbis Publishing Ltd: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER COURSE – Copies are obtainable by placing a regular order at your newsagent.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595. Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

His Master's Voice

Computers are already an established part of the professional music scene. Now small music synthesisers are being built into many home computers too

Computers are both fun and serious. And they are not just for processing information or playing games. They can be used as musical instruments too — you could say for serious fun. The process of making music artificially is called music synthesis.

Computers can also be used to liven up the teaching of music and work out considerably cheaper than professional tutors. In a forthcoming issue, we'll be explaining how you can make such music on your home computer, but for the moment we'll concentrate on how the professionals do it. This is important, because most of the ideas that start out on professional synthesisers end up as standard on domestic models, or even home computers.

Automatic musical instruments have always been popular and they have a lot in common with computers. The pianola, a sort of automatic piano to be found in the drawing rooms of wealthy Victorians, was operated by means of a roll of punched paper, and musical boxes featured a metal drum or disk with 'teeth' that played a tune on a metal comb.

Even the old crank-handled street organs were in a sense programmable, since the tunes they played could be changed. Nevertheless, this did not commend them to Charles Babbage, one of computing's founding fathers, who wanted to have the organ grinders banned from playing in the street. Their response was to play directly underneath his window.

Nowadays it is the Musicians Union that is trying to ban programmable musical devices — in May 1982, the Central London Branch voted to forbid their use at recording sessions and live performances. Their obvious worry is that because such devices can imitate many different instruments, several at a time, musicians will become redundant.

Electronic synthesisers have been available for many years but the introduction of digital techniques has opened up a whole new area. Instead of having to fiddle with knobs and press buttons to produce every single sound, it is now possible to record any sound (from a conventional instrument to the bursting of a balloon), analyse it by computer into its constituent parts and replay it at any pitch.

Digitised sound is something like a newsprint photograph — if you look very closely at the page, you can see that the whole picture is made up of many small separate dots, whereas the original (analogue) photograph had tones that shade continuously into each other. In the same way, ordinary analogue sound can be broken up into a sequence of digits. This technique is known as sampling.

Such systems are expensive — the Fairlight and Synclavier are probably the best-known and most sophisticated models — but as they can reproduce the sounds of a number of musical instruments, they can work out cheaper than hiring individual musicians.

With computers becoming cheaper, and the cost of memory falling, digital machines are gaining in popularity — though it will be a long while before the analogue synthesisers disappear altogether. The latter use a technique known as 'subtractive synthesis' — which can be thought of as comparable to the way a sculptor carves his statue from a block of marble. You start with a basic sound created electronically and then pass

COURTESY OF EMI

One Man Band
Synthesiser players like Klaus Schultz, (shown here) are increasingly using the tremendous power of their microprocessor-based instruments to produce live, on stage, the sort of varied sound that 20 years ago would have required a whole orchestra

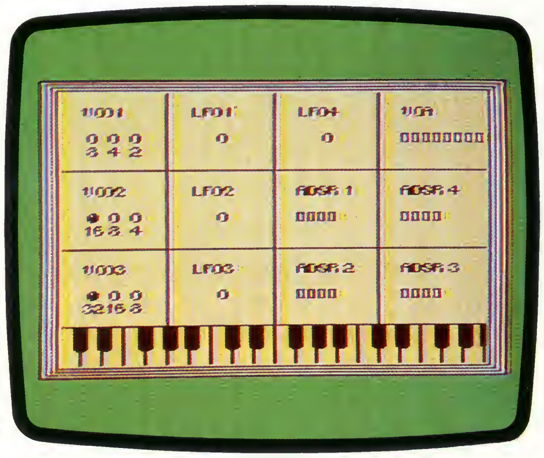
Special Effects
Home computer-controlled music synthesisers are becoming increasingly popular. Musical effects, which 10 years ago were only available on the most expensive professional equipment, can now be bought for around £100 to £200. The one pictured here has a facility to read into its memory, music that has been encoded onto paper as bar codes. It is now ready to be reproduced or altered by the player. Many of these home synthesisers will link directly into a home computer, to take advantage of the screen and additional memory. However, more and more home computers incorporate some form of music synthesis

IAN McKINNELL

this through a number of electronic processes. Each process modifies or chips away at parts of the sound to turn it into the shape required. Subtractive synthesis encourages the musician to experiment with different combinations of processes, and is relatively easy for the beginner to get to grips with.

By contrast, anything created on a digital synthesiser needs to be as carefully planned as the construction of a large building. This is because the device makes use of 'additive synthesis' — the final sound is produced by adding components, one on top of the next. One has to be fairly near to the end of the process before the sound is even barely recognisable. However, it is possible to take

Musical Boxes
Many home computers now have packages available for them that exploit their music-making capacities. The screen display in these packages can be used to make a visual interpretation of the music being played, or can help the novice musician by interpreting the QWERTY keyboard into an approximation of the one found on a piano



IAN McKINNELL

a conventional sound, analyse it down into its basic components, store these in the computer's memory, either in internal RAM or on disk or tape, and use them as 'bricks' in constructing the sound required.

As well as the ability to create such a wide variety of individual sounds, computers can also be used to store musical sequences and compositions. Most top-selling synthesisers have a sequencer (the device that stores and recalls the sequences of sounds) as an option, if not built in as standard. The Fairlight, for example, can store up to 30 minutes of sound on its disk memory, and for up to eight 'voices', which can be thought of as individual instruments. The Synclavier has double that number.

It can easily be seen that with the most expensive synthesisers (and here we are talking about £15,000 and upwards), the composer, musicians and conductor are rolled into one person — who to a large extent is a computer programmer as well.

If you have ever tried to play back a tape recording of your own voice at a higher speed (or for that matter played a 33 rpm record with the turntable set to 45 rpm) you will be aware that the pitch rises dramatically. One of the intriguing features of computer-controlled synthesisers is their ability to overcome this effect and play back a piece of music at a faster or slower speed than its original performance without altering the pitch, or conversely to transpose the tune into a different key at the same speed.

It is even possible to take, say, a trumpet track, duplicate it, and at the same time change the sound to that of a French horn. The two instruments can then be played together in unison, or alternatively, in harmony. This is called 'track bouncing'.

Many of these machines are instructed by means of a composition language (the Synclavier's is called SCRIPT), which looks not dissimilar from BASIC — complete with line numbers — though perhaps a little more cumbersome to use. The most outstanding feature is called 'reverse compiling' — play a piece of music on the keyboard, and the computer produces a listing for your composition in SCRIPT. This is equivalent, if you can imagine such a thing, to being able to play a new game of your own devising on your computer's screen, then pressing a button to get a complete program listing for your game!

If your timing hasn't been quite perfect then it is possible to edit and replay the SCRIPT listing using a conventional (QWERTY) keyboard and screen, just as in BASIC. A less flexible but prettier system comes with one of the Yamaha synthesisers: it prints out what you have played in conventional music format, as notes on a stave.

Synthesisers are becoming established in the cinema, also. Walt Disney's film *TRON* received rave reviews for its stunning computer-generated graphics. Less well known is the fact that microcomputers were used for the music and sound effects. In addition to some 'real' music from the London Philharmonic Orchestra, a wide variety of sounds, including the noises made by the Goodyear airship and a refrigerator, were recorded and then altered using a Fairlight.

So many sounds were involved that a complete catalogue had to be kept up to date using the File Manager 800+ database package on an Atari 800 home computer. Some of the artificial voices in the film were also constructed using a home computer and a voice-synthesis device that allowed voice and music to be mixed together. Atari also lent them a package that had previously not been used outside the company. It amounted to a kind of additive synthesis program, which they had been using to create sophisticated sound effects on their own games software and coin-operated arcade machines. The claim was that this Electronic Sound Assembly System did for the creation of sound what word processing did for the creation of text.

There are in fact quite a number of add-ons that can turn an ordinary home computer into a sophisticated music synthesiser. The Apple is a particularly popular model for this application, because of the slots in the back of the computer into which additional devices can be plugged. Some of these use the computer to do all the creation, and simply provide a high-quality analogue synthesiser as an output, while others include a full piano-style keyboard.

At the domestic end of the market for computerised synthesisers, the Casio CT7000 features something called a 'polyphonic sequencer', which by means of a cassette deck and a large amount of RAM memory, manages to perform in a similar way to a professional multi track recording system. A domestic cassette recorder has just one track per side, a stereo tape deck has two. But a professional unit may have more than 24 so that each instrument can be recorded separately and then mixed together to make up the complete sound. With the CT7000 you can build up your own symphonies in this manner.

Its predecessor, the CT701, used a bar code reader (see page 40) to read music from printed bar codes into the machine's memory. Sadly, there seems to be no easy way for the user to create printed bar codes of his own creations, so this method is strictly available for use with Casio pre-printed music.

You don't even need this level of equipment to be successful in the pop world, as demonstrated by the German group Trio, who had more success with their hit song 'Da Da Da' using a £30 Casiotone VL1, than did Peter Gabriel with his Fairlight costing thousands of pounds.

Though the VL1 is a monophonic device (you can only play one note at a time), it does have the ability to emulate different musical instruments, and store a sequence of notes. There is also the ability to change certain qualities of the notes to create your own sound. This is known as altering the Attack, Decay, Sustain, Release (ADSR) envelope. Now the interesting thing is that home computers such as the Commodore 64 and Oric-1 have exactly the same features. And of course the very ability to be programmed in BASIC means that you have a built-in sequencer, too.

Music composition packages are available for many of these home computers — even the ones with comparatively simple music capabilities. Some of them display a musical stave on the screen, and music is composed by selecting the different kinds of musical notes with the aid of a joystick or a light pen and placing them on the stave. Pressing the 'fire' button on the joystick, or some equally simple command, results in your composition being played. Alternatively, the display may represent a piano keyboard — the notes being selected again with the light pen or joysticks, or with the computer's own keyboard.

Without the aid of such a package, musical effects can be programmed in BASIC. As with graphics facilities, the exact method varies greatly from machine to machine, as do the sophistication of the BASIC commands dedicated to this facility. The Dragon, for example, has just one voice, but features a PLAY command that initiates a sequence of notes typed in as the letters of the musical scale (A to G). The Commodore 64, by contrast, has very sophisticated musical facilities built into its hardware, but no such easy-to-use commands in its BASIC. In a future article we shall be instructing you how to get music from your machine.

Martin Rushent

The producer of bands such as Altered Images and Dexy's Midnight Runners, Martin Rushent is one of the leading proponents of computer-controlled music synthesisers. His studio in the Berkshire countryside contains no less than nine different systems, which represent the current state of the art. Each of them uses a different method of programming, and different codes for the notes of a musical scale. Because they have defied all attempts to be linked together into one gigantic system, Rushent uses a program he wrote himself on a small home computer to convert a piece of music from one system to another

LEN LEWIS

Properly Addressed

The CPU has to locate instructions and data stored in thousands of bytes of computer memory. We reveal what goes on inside the CPU when program instructions are executed

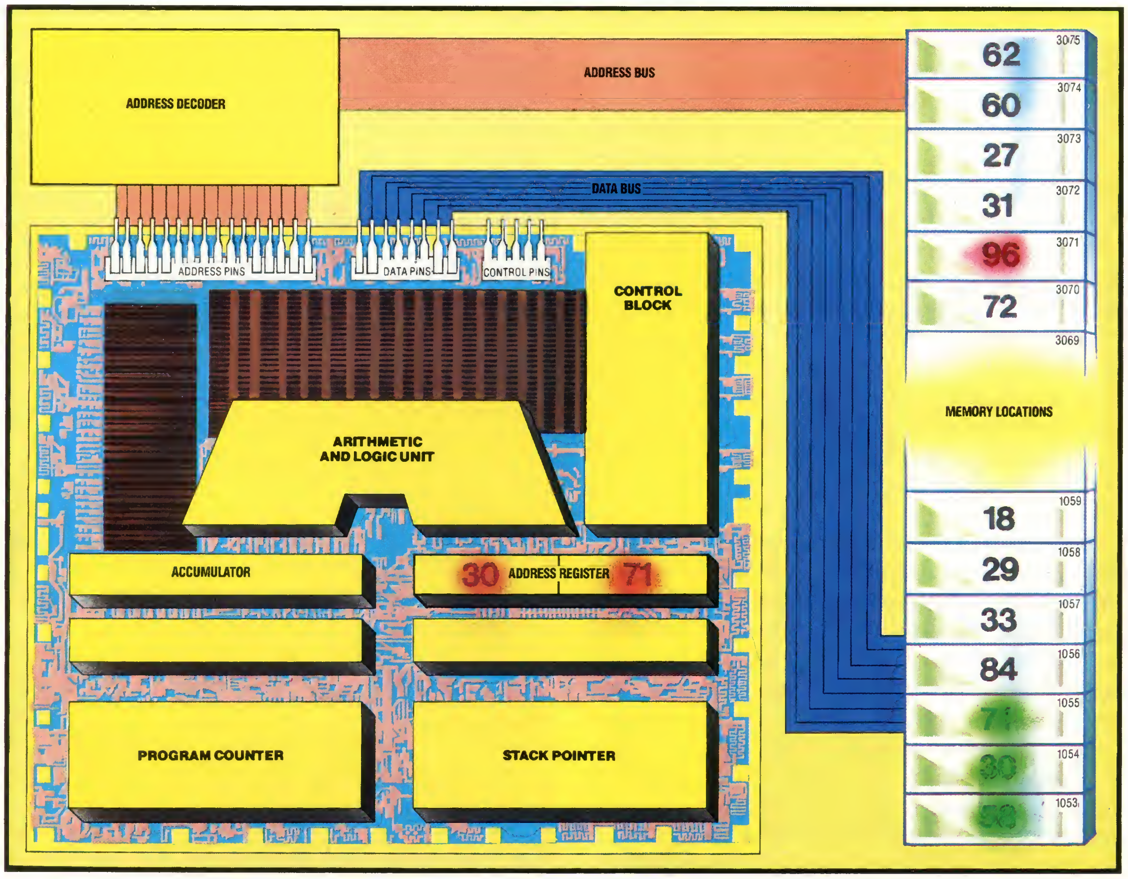
The CPU receives its instructions and data from locations in the computer's memory by setting its address pins to the required binary code for the memory location and then reading the contents of the location into the CPU via the data bus. In actual operation, however, the operation is rather more complicated.

The problem is that the bytes (eight-bit binary codes) in any of the thousands of memory cells in the computer's memory might be instructions, telling the CPU to do something, or data, which the CPU must manipulate in some way. How does the CPU know which bytes are instructions and which are data?

Recognising Codes

First, let's consider what an 'instruction' is. It is a code, in binary, which causes a specific sequence of operations to be performed within the CPU. Thus the code 00111010, if recognised by the CPU as an instruction rather than as just a piece of data, might make the CPU address the next two bytes in memory, read in the data from them, put that data in a special 'address register', set the address pins to the same number, go to the newly addressed memory location, get the contents of that location on the data bus and load those contents into the CPU's accumulator.

This can sound confusing when expressed in words, but what we have just described is one of the methods of memory addressing used in the popular Z80 CPU. The entire process of getting a byte of data from memory into the CPU is shown in the illustration. Suppose the CPU already knows that the next byte accessed from memory will be an instruction (not data) and that this byte resides in memory location 1053. (All the numbers used in this illustration are in decimal notation.) This address, 1053, will be put on the address bus. In binary, this is 0000010000011101. The 16 address pins are switched 'on' or 'off' to correspond to this number. When the 'address decoder' receives this address over the address bus, it 'decodes' it and switches on one, and only one, of its output lines. This is the line that selects



Chains Of Events
Many stages are involved in even the simplest CPU operation. Instructions, also called 'op-codes', are read into the CPU from memory. These instructions are decoded by the control block and cause specific operations to occur. In this example, instruction 58 is read in from memory location 1053. This particular instruction causes the following chain of events to occur: the byte in the next memory location (1054) will be read in and stored in one half of the CPU's 16-bit address register. The byte in the next location (1055) will be read in and stored in the other half. These two bytes now represent the address (elsewhere in memory) where some data is stored. The contents of the address register are now put on the address bus so that the next memory location accessed will be address 3071. The contents of this address are put onto the data bus and read into the CPU. This byte (96 in our example) is then placed in the CPU's accumulator, where it will stay until operated on by a further instruction. The address bus will then revert back to its previous address + 1, so that it will now be addressing location 1056. The CPU knows that whatever is contained in that location must be an instruction and a similar sequence of operations will be repeated. In this example, the next instruction is 84, which is interpreted by the control block to 'complement' or invert the bits in the accumulator. Since 84 is a 'one byte' instruction, the CPU knows that the byte in the next memory location, 1057, will also be an instruction

the memory location 1053.

The next stage is that the contents of this address, which is 58, or 00111010 in binary, is placed onto the data bus and 'loaded' into the CPU. Here, because the CPU is expecting an instruction, the byte is interpreted by the control block and causes a very precise sequence of operations to be performed. This particular instruction specifies that the next two bytes in memory will contain 16 bits to be used as a memory location, and that the contents of this location are to be loaded into the CPU's accumulator. As soon as the CPU recognises this instruction, it knows that the next two bytes in memory will specify an address and that the contents of that address will be loaded into the accumulator. It consequently knows that it will not receive another instruction from memory until after these operations have been performed, and that the next instruction will reside in location 1056.

The instruction we are using as an example causes the address bus to be incremented by one, so that the next memory location addressed is 1054. The contents of this location are then put on the data bus and loaded into the CPU. This time, however, they are put into one half of an address register. Having done that, the CPU increments the address bus again so that it now addresses location 1055. The contents of this location go on to the data bus and are similarly loaded into the CPU, this time to be stored in the other half of the address register.

Transferring Numbers

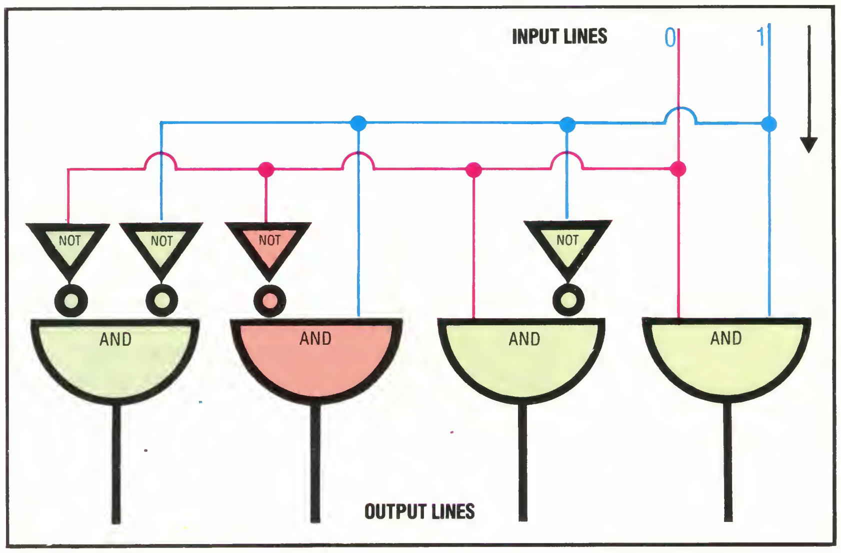
The next stage — and remember that all of these actions happen automatically as a result of the original instruction — is that the numbers in the address register are transferred to the address bus. These numbers, as we can see, are 3071. The memory location now being addressed is therefore 3071. This address (0000101111111111 in binary) is decoded by the address decoder and selects memory cell 3071. The contents of this location, 96, (01100000 in binary) are put onto the data bus and loaded into the CPU. This time, however, the data will be put in the CPU's accumulator. After this the address bus will be set to 1056 and the CPU will expect to find another instruction there.

Now that the CPU has one piece of data in its accumulator, what sort of instruction might be expected to be encountered next? It could be almost anything — CPUs have from a few dozen to a few hundred instructions they recognise, depending on the CPU — but suppose we wanted to invert the data in the accumulator. Inverting means changing each one into a zero and each zero into a one. The instruction to do this would be located at address 1056. On our imaginary CPU, the code for this instruction would be 84. When this number was received by the CPU, the data in the accumulator would be inverted. The number that was in the accumulator was 96 (01100000 in binary). The instruction to invert would cause it to be changed to 10011111 in binary. The instruction to invert a number in the accumulator is a 'one-byte' instruction, so again the CPU would know that the contents of the next memory location, 1057, would again be an instruction rather than data.

This method of addressing a memory location to retrieve a piece of data is only one of several methods available to the programmer. The specific instruction bytes we used in the example (58 to load the accumulator and 84 to invert the contents of the accumulator) are the instructions for our hypothetical CPU, but the same principle applies to all other microprocessor chips. The only difference is that different codes are used for the various instructions and each make of CPU has slightly different 'instruction sets'.

I/O (Input/Output) locations must also have unique addresses, but the principles for addressing them by the CPU are the same. Usually, in eight-bit microprocessors, only eight of the address lines are available for addressing I/O locations, so the maximum number of I/O addresses is 256. This, however, is more than enough for most small computer applications.

Address Decoding
The 16 lines constituting the address bus are capable of uniquely identifying any one of 65,636 separate memory locations. The combination of ones and zeros on the address bus are decoded in address decoders. Part of the decoding is performed by address decoders built up from simple logic gates in chips mounted on the circuit; much of the decoding is performed by equivalent circuits inside the memory chips themselves. The illustration shows how two address lines can be decoded to select one, and only one, of four chips



Address decoding is always needed so that the device selected by the CPU (whether it be a memory location or an I/O location) is made uniquely active while all the other memory or I/O locations remain inactive. This process is called 'enabling'. When there are only a small number of address lines to decode, it is possible to use simple logic gate chips to perform the decoding. The principle of a two-to-four line decoder is shown in the illustration. It is usual to use this type of simple decoding for selecting I/O devices. As the number of address lines increases, however, the complexity of the decoding circuit grows massively. When there are 65,536 separate memory locations that must be individually and uniquely selected, it is usual for most of the address decoding to be performed inside the memory chips.

Fully Functional

There are built-in functions in Basic, which means that a lot of the programming has already been done for you. Knowing how to use them adds power to your computing

Suppose you wanted to calculate the square root of a number in one of your programs. There are a number of ways this could be done. The crudest and least satisfactory way would be to create a table of square root values and to use this to give you the value wanted for a particular number. You probably learnt how to do this in school. An alternative method is to use the square root 'function', built in to most versions of BASIC. Here, the arithmetic of the operation is taken care of by BASIC without the programmer having to worry about it. Let's see how it works:

```
10 REM THIS PROGRAM FINDS THE SQUARE ROOT
20 REM OF A NUMBER
30 PRINT "INPUT THE NUMBER YOU WANT TO"
40 PRINT "FIND THE SQUARE ROOT OF"
50 INPUT N
60 LET A = SQR(N)
70 PRINT "THE SQUARE ROOT OF ";N;" IS ";A
80 END
```

Type in this short program and see that it does indeed give you the square root of any number you type in. Let's look at the rules of how to use this 'square root' function.

A 'function' in BASIC is generally a command word (SQR in this case) followed by brackets that enclose the expression to be operated on. In this program, N is the number input from the keyboard. It is the number we want the square root of. Line 60 says 'let the square root of N be assigned to the variable A'. Line 70 prints out the value of A.

The expression inside the brackets is called the 'argument' of the function and does not always have to be a variable: it is equally possible to use actual numbers. Type this in and see what happens when you run it:

```
10 PRINT SQR(25)
20 END
```

You will see that this works just as well. Similarly, we can use more complex arguments inside the brackets. Try this one:

```
10 LET A = 10
20 LET B = 90
30 LET C = SQR(A+B)
40 PRINT C
50 END
```

This little program can be shortened by combining lines 30 and 40 like this:

```
10 LET A = 10
20 LET B = 90
30 PRINT SQR(A+B)
40 END
```

The way to think of functions is as short programs built in to BASIC that are available for the programmer to use at any time. Most versions of BASIC have quite a large number of functions as well as the facility of allowing the programmer to define new ones for use within a program. Later, we will see how this is done. Here we will look at a few more of the commonly available functions. They come in two varieties: numeric functions, in which the argument (the part inside the brackets) is a number, numeric variable or numeric expression, and string functions, in which the argument is a character string or expression made up from character strings. First we'll look at a few of the numeric functions.

Previously, on page 77, we used a program that calculated the number of tiles needed to tile a room. A small 'bug' in this program was that the answer could well involve fractions of a tile. 988.24 could represent a possible result of running this program. At times like this we want a way of rounding the answer to the nearest whole number. Whole numbers are referred to mathematically as 'integers' and one of the functions in BASIC will 'return' the integer part of any number. Here's how it works:

```
10 PRINT "INPUT A NUMBER CONTAINING A
   DECIMAL FRACTION"
20 INPUT N
30 PRINT "THE INTEGER PART OF THE NUMBER
   IS ";
40 PRINT INT(N)
50 END
```

If this program is run and the number you input is 3.14, the program will print on the screen:

```
THE INTEGER PART OF THE NUMBER IS 3
```

Of course, if we are dealing with tiles, we would then need to add 1 to the answer, to make sure that we bought more than the required amount, not less.

On other occasions, we may want to find the 'sign' of a number to see if it is negative, zero or positive. To do this, most BASICs incorporate a SGN function. Try this:

```
10 PRINT "INPUT A NUMBER"
20 INPUT N
30 LET S = SGN(N)
40 IF S = -1 THEN GOTO 100
```

```
50 IF S = 0 THEN GOTO 120
60 IF S = 1 THEN GOTO 140
100 PRINT "THE NUMBER WAS NEGATIVE"
110 GOTO 999
120 PRINT "THE NUMBER WAS ZERO"
130 GOTO 999
140 PRINT "THE NUMBER WAS POSITIVE"
150 GOTO 999
999 END
```

If you look at the values 'returned' by the SGN function to S in line 30 (these are tested in lines 40, 50 and 60) you will see that there are three possible values. −1 is returned if the argument in the brackets was a negative number, 0 if the argument was zero and 1 if the argument was a positive number. Using the SGN function in line 30 saves several lines of programming. We could have written:

```
IF N < 0 THEN LET S = -1
IF N = 0 THEN LET S = 0
IF N > 0 THEN LET S = 1
```

The action performed by a BASIC function can always be achieved through normal programming; using a function just saves time, space and programming effort.

Here are a few more numeric functions. ABS returns the 'absolute' value of a number. The absolute value of a number is the same as its real value with the sign removed. Thus, the absolute value of −6 is 6. Try this:

```
10 LET X = -9
20 LET Y = ABS(X)
30 PRINT Y
40 END
```

MAX finds the maximum value of two numbers. Thus:

```
10 LET X = 9
20 LET Y = 7
30 LET Z = X MAX Y
40 PRINT Z
50 END
```

MIN is similar to MAX but finds the smaller of two numbers. Try this:

```
10 PRINT "INPUT A NUMBER"
20 INPUT X
30 PRINT "INPUT ANOTHER NUMBER"
40 INPUT Y
50 LET Z = X MIN Y
60 PRINT Z
70 END
```

Notice that these latter two functions have two arguments instead of one, and they don't need to be enclosed in brackets. Most BASICS also have a number of other numeric functions, including LOG to find the logarithm of a number, TAN to find the tangent, COS to find the cosine and SIN to find the sine. We will look at some of the ways these 'trigonometrical' functions can be used later.

BASIC also has several built-in functions that operate on character strings. We used some of these in our name-sorting program (page 135) but at the time did not look closely at how they worked. Now we will look at these and a few other string functions in more detail.

One of the most useful string functions is LEN. This counts the number of characters in a string enclosed in double quotation marks or the number of characters assigned to a string variable. Try this:

```
10 LET A$ = "COMPUTER"
20 LET N = LEN(A$)
30 PRINT "THE NUMBER OF CHARACTERS IN THE
   STRING IS ";N
40 END
```

Why would we ever need to know how many characters there are in a string variable? To see why, enter and run this short program designed to build a 'name triangle'. It prints first the first letter of a word, then the first and second letter and so on until the whole word is printed.

```
5 REM PRINTS A 'NAME TRIANGLE'
10 LET A$ = "JONES"
20 FOR L = 1 TO 5
30 LET B$ = LEFT$(A$,L)
40 PRINT B$
50 NEXT L
60 END
```

Now run this program. Can you figure out what the printout will be? It should look like this:

```
J
JO
JON
JONE
JONES
```

This short program uses the LEFT$ function to extract characters from a string. LEFT$ takes two arguments. The first specifies the string and the second (which comes after a comma) specifies the number of characters to be extracted from the string, starting from the left of the string. A$ has been assigned the string "JONES" so LEFT$(A$,1) would 'return' the letter J. LEFT$(A$,2) would return the letters JO. The short program above uses an index, L, that ranges from 1 to 5, so that the second argument in the LEFT$ function goes up from 1 to 5 each time through the loop. We knew exactly how many characters there were in the word we wanted to print (JONES), so it was easy to decide that 5 should be the upper limit in the FOR-NEXT loop. But what would we do if we did not know beforehand how many characters there would be in the loop?

This is where the LEN function comes in. LEN takes a string (in double quotes) or a string variable as its argument. Here are a few examples to show how it works:

```
10 REM PROGRAM TO TEST THE 'LEN' FUNCTION
20 PRINT LEN("COMPUTER")
30 END
```

The program should print 8 when it is run. It has counted the number of characters in the word COMPUTER and returned this value. Let's do the same thing in a slightly different way:

```
10 REM FINDING THE LENGTH OF A STRING
20 LET A$ = "COMPUTER COURSE"
30 LET L = LEN(A$)
40 PRINT L
50 END
```

If this program is run, the computer should print 15 on the screen. There are 15 characters in this string, not 14. Don't forget that the space between the two words is a character as far as the computer is concerned. Now let's apply the LEN function in a modification of our earlier program to print out a 'triangular name':

```
10 REM THIS PROGRAM PRINTS A 'NAME
   TRIANGLE'
20 PRINT "TYPE IN A NAME"
30 INPUT A$
40 LET N = LEN(A$)
50 FOR L = 1 TO N
60 LET B$ = LEFT$(A$,L)
70 PRINT B$
80 NEXT L
90 END
```

Each time this loop is executed, the value of L increments from 1 up to N (which is the length of the name in the string). If you input the name SIMPSON, line 40 will be equivalent to LET N = LEN ("SIMPSON"), so N will be set to 7. The first time through the loop, line 50 will set L to 1 and line 60 will be equivalent to LET B$=LEFT$ ("SIMPSON",1) so B$ will be assigned one character from the string, starting from the left. This character is S.

The second time through the loop, L will be set to 2 so line 60 will be equivalent to LET B$ = LEFT$ ("SIMPSON",2). This will take the first two characters from the string and assign them to string variable B$. B$ will therefore contain SI.

The LEN function found that there were 7 characters in the string SIMPSON and assigned this value to variable N, so the last time through the loop B$ will be assigned all seven characters from the string and the whole string will be printed.

Note that LEFT$ has a companion function, RIGHT$, which takes characters from the right of the string variable in exactly the same way.

Finally, we will look at one more string function, also used in the name-sorting program. This is INSTR; it is used to find the location of the first occurrence of a specified string (called a 'sub-string') within a string. In the name-sorting program, INSTR was used to locate the position of the space between the first name and surname. Here's how it works:

```
10 LET A$ = "WATERFALL"
20 LET P = INSTR(A$,"FALL")
30 PRINT P
40 END
```

Before entering and running the program, see if you can anticipate what value will be printed for P. Remember, INSTR locates the starting position of the first occurrence of the 'sub-string' within the string. If the string is WATERFALL, the starting position of the sub-string FALL will be 6 — the F in FALL is the sixth letter in WATERFALL. Some BASICS do not use INSTR, but have a similar function called INDEX instead. Here's how to use INSTR (or INDEX) to locate a space within a string:

```
10 REM FINDING THE POSITION OF A SPACE IN A
   STRING
20 LET A$ = "HOME  COMPUTER"
30 LET P = INSTR(A$," ")
40 PRINT P
50 END
```

Notice that the second argument in the INSTR function (line 30) is " ". The quotes enclose a space — the character to be searched for. The program will print 5 as the value of P since the space is in the fifth position in the string. Work out what would be printed if line 30 were changed to:

```
LET P = INSTR(A$,"C")
```

Lastly, a handy function used with the PRINT statement. See what happens when you run this program:

```
10 PRINT "THIS LINE IS NOT INDENTED"
20 PRINT TAB(5); "THIS LINE IS INDENTED"
30 END
```

Can you see what happened? The second line was printed starting five places in from the left margin. TAB is analogous to the tabulator on a typewriter. Here is another short program using the TAB function:

```
10 REM  USING THE TAB FUNCTION
20 PRINT "ENTER THE TAB VALUE"
30 INPUT  T
40 LET W$ = "TABULATION"
50 PRINT TAB(T);W$
60 END
```

Now you can go back to the name-sorting program on page 136 and see how some of those functions were used there.

Exercises

■ **Loops 1** What will be printed when this program is run?

```
10 LET A = 500
20 FOR L = 1 TO 50
30 LET A = A -1
40 NEXT L
50 PRINT "THE VALUE OF A IS ";A
```

■ **Loops 2** What will you see on the screen if this program is run?

```
10 REM
20 REM THIS IS A TIMING LOOP
30 REM SEE HOW LONG IT TAKES
40 REM
```

Basic Flavours

TAB

On the Spectrum and ZX81 functions such as TAB, LEN, CHR$ can be used without brackets, so that TAB(30), for example, can be written TAB 30

LEFT$

RIGHT$

MID$

None of these commands is available on the Spectrum or ZX81; their equivalents in Sinclair BASIC are:

LEFT$(Z$,N)	replace by Z$(TO N)
RIGHT$(Z$,N)	replace by Z$(LEN(Z$)–N+1 TO)
MID$(Z$,P,N)	replace by Z$(P TO P+N–1)

When N=1 in a MID$() command then the Sinclair equivalent is very much simpler than the above:

MID$(Z $,P,1)	replace by Z$(P)

INSTR

This is not available on the Spectrum, ZX81, Commodore 64, Vic 20, and Oric-1, but you can write a sub-routine to replace it. Suppose that a program line is:

```
20 LET P=INSTR(A$,"FALL")
```

Replace it by:

```
20 LET X$=A$: LET Z$="FALL":GOSUB
      9930:LET P=U
9929 STOP
9930 LET U=0:LET X=LEN(X$): LET
      Z=LEN(Z$)
9940 FOR W=1 TO X–Z+1
9950 IF MID$(X$,W,Z)=Z$ THEN LET U=W
9960 IF U=W THEN LET W=X–Z+1
9970 NEXT W
9980 RETURN
```

On the Spectrum and ZX81 replace line 9950 by:

```
9950 IF X$(W TO W+Z–1)=Z$ THEN LET
      U=W
```

```
50 PRINT "START"
60 FOR X = 1 TO 5000
70 NEXT X
80 PRINT "STOP"
90 END
```

■ **Loops 3** What result will be printed if the following program is run and you type in the number 60 when asked?

```
10 PRINT "THINK OF A NUMBER AND TYPE IT IN"
20 INPUT N
30 LET A = 100
40 FOR L = 1 TO N
50 LET A = A + 1
60 NEXT L
70 PRINT "THE VALUE OF A IS NOW ";A
80 END
```

■ **Loops 4** What will happen if this program is run?

```
10 PRINT "I LIKE BASIC"
20 GOTO 10
30 END
```

■ **Loops 5** What will you see on the screen if this program is run?

```
10 FOR Q = 1 TO 15
20 PRINT "I'M FEELING LOOPY"
30 NEXT Q
40 END
```

■ **Read-Data 1** What result will be printed?

```
10 READ X
20 READ Y
30 READ Z
40 PRINT "WE'RE TESTING THE 'READ'
 STATEMENT"
50 DATA 50,100,20
60 PRINT X + Y +Z
```

■ **Read-Data 2** What will be printed on the screen if this program is run?

```
100 FOR L = 1 TO 10
110 READ X
120 PRINT "X = ";X
130 NEXT L
140 DATA 1,2,3,5,7,11,13,17,19,23
```

Answers in next issue

Answers To 'Exercises' On Pages 136-137

Variables

[✓ A] [✓ B6] ~~2Z~~ D$ ~~15~~ X$ [✓ A12] [✓ D9] [✓ Q81] [✓ Q5] ~~8F~~ H$

Arithmetic 1

```
10 LET B = 6
20 PRINT B
```

Arithmetic 2

```
10 LET A = 5
20 LET B = 7
30 LET C = 9
40 LET D = A + B + C
50 PRINT D
```

Arithmetic 3

17

Arithmetic 4

25

25

Comparisons 1

5

Comparisons 2

601 (integers are assumed)

Comparisons 3

10000

Print 1

```
PRINT "THE VALUE OF T IS ";T
```

Print 2

```
640 PRINT "SORRY, BUT YOUR SCORE OF ";S;" IS TOO LOW"
```

Print 3

This was a deliberate mistake.
The semi-colon at the end of the line will cause a syntax error at run time. The program should read:

```
200 LET A$="THE HOME COMPUTER COURSE?"
210 LET B$="HOW DO YOU LIKE "
220 PRINT B$;A$
```

And the result would then have been:

```
HOW DO YOU LIKE THE HOME COMPUTER COURSE?
```

Input 1

6

Input 2

```
PLEASE TYPE YOUR NAME
HI (YOUR NAME) I'M YOUR COMPUTER
```

Note that the answers to 'Variables' will differ for some machines, which do not allow more than one alphabetic character (i.e., no numeric suffix)

Jupiter Ace

The only low-cost home computer to feature Forth instead of Basic as its standard programming language — a challenge for ambitious programmers

The Jupiter Ace is an enthusiast's machine, and one of the few computers that does not have BASIC as its standard language. Though production was discontinued in 1983, devotees keep up a brisk trade in the Ace and its software.

The built-in language is FORTH and is the distinguishing feature of this machine. But the hardware is cheap enough for anyone who is interested in learning FORTH to buy the Ace rather than upgrade their existing computer. The computer comes with a manual that is an excellent tutorial on FORTH.

The Ace is laid out very much like the Sinclair models. In fact its case is the same white, flimsy plastic that the very first Sinclair computer (the

Introducing Forth

BASIC Version

```
100 REM A BASIC PROGRAM TO PRINT 'SHAZAM!'
110 FOR X = 1 TO 6
120 PRINT "SHAZAM!"
130 NEXT X
140 END
    RUN
```

FORTH Version

```
( A FORTH PROGRAM TO PRINT 'SHAZAM!')
: SHOUT ." SHAZAM! " ; .
: CHORUS 6 0 DO SHOUT LOOP ;
 CHORUS
```

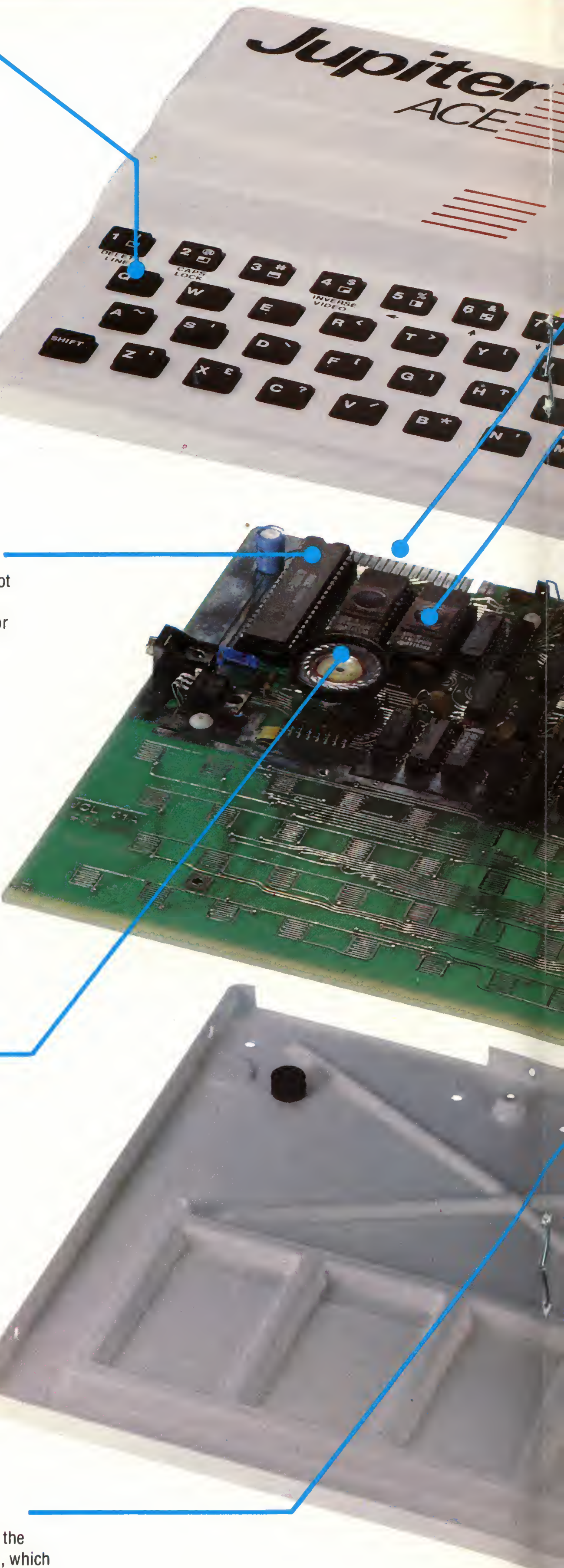
Both the above programs will do exactly the same thing, but the BASIC version resembles a recipe, while the FORTH program looks like a wizard's spell!

FORTH starts with a collection of command words (called the dictionary), and has the ability to learn new words. In our FORTH program two new words are added to the dictionary: SHOUT is defined as a character string to be printed and CHORUS is defined as a mixture of 'primitives' (pre-defined words in the dictionary) and the new word SHOUT.

FORTH also has a memory (called the stack), and the ability to process the numbers in it. The FORTH program does the same arithmetic and logic as the BASIC program, but by manipulating the stack instead of through algebraic expressions.

FORTH is an infuriating 'Rubik's Cube' of a language; it is also a powerful programming method, and a whole new way of thinking. Some programmers love it, some hate it.

It is the ability to define and use new command words that really gives FORTH its power. Effectively, the user can tailor the programming language to suit the application he is implementing. FORTH is particularly suitable for programming domestic robot devices, for example, because the programmer can build up his own dictionary of commands: MOVE, FETCH, FIND, FOLLOW and RETURN for example

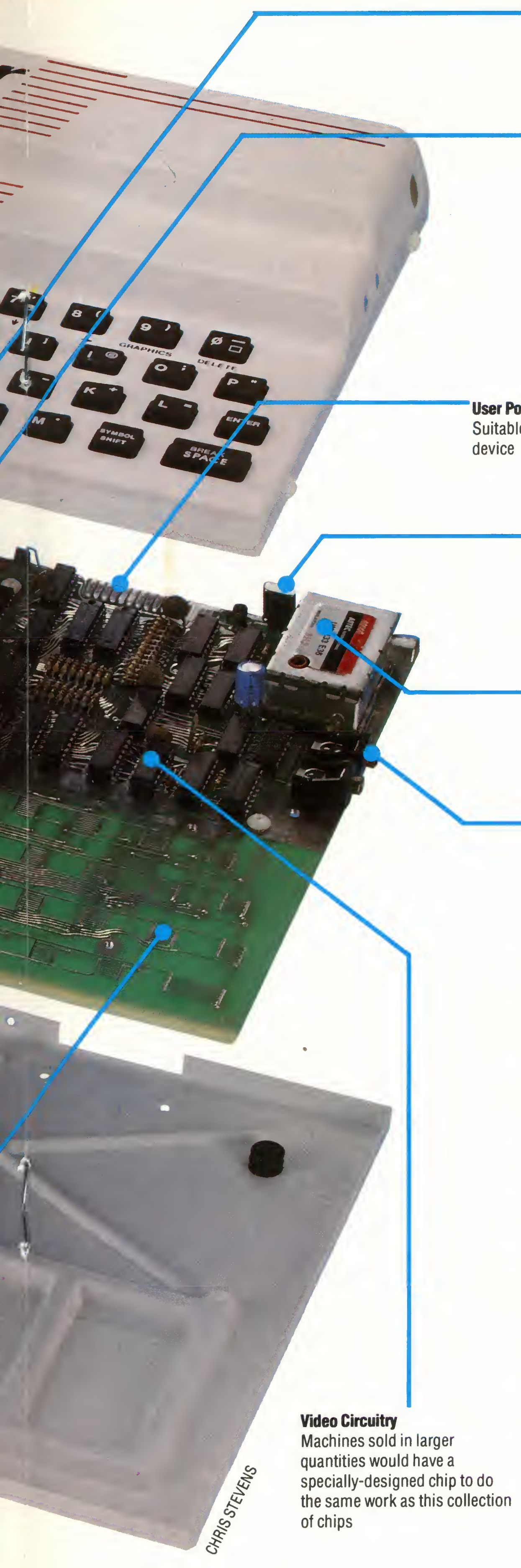


Jupiter Ace Keyboard
The 40 moving keys are moulded from one sheet of rubber — much like the Sinclair Spectrum. The top row of keys all have three functions, which are accessed with the SHIFT and SYMBOL SHIFT keys. In addition, seven graphics characters (plus 'space') can be used to construct simple diagrams and graphs

Microprocessor
The Z80A has been used — not surprising as the machine's designers were responsible for the Sinclair Spectrum, too

Speaker
This is a solid-state piezo-electric device (much like the bleepers found on digital watches), which can be used to generate simple sounds

Keyboard Pads
When a key is pressed, the special rubber material, which conducts electricity, makes a contact between two sets of metal tracks on the printed circuit board



Memory Expansion Port
With a suitable adaptor, the Ace can also make use of Sinclair ZX81 expansion packs

FORTH Language
Two 4 Kbyte EPROMs have been used to hold the language. ROMs have to be ordered by the thousand, so EPROMs are used for smaller production quantities. The top of each EPROM has a protective covering, otherwise any ultraviolet light would erase the contents

User Port
Suitable for a printer or other device

Clock
This drives the microprocessor at 1MHz

RF Modulator
Provides black and white output only for a television

Cassette Port
One of the most reliable systems devised

Video Circuitry
Machines sold in larger quantities would have a specially-designed chip to do the same work as this collection of chips

ZX80) had. There is a small solid-state bleeper that can produce a range of single tones, but with a greater knowledge of the Ace complex sounds can be produced. Like all the Sinclair computers, the Ace has a power connector, a socket for a lead to a television and two large connectors for any other special pieces of equipment.

The cassette has a simple two-jack socket connection, which is one of the most reliable cassette interfaces on any computer.

The display is black and white with 32 characters to a line, although it has a graphics resolution of 64 by 48. Each of the characters can be redesigned by the user to create specific mathematical symbols or game shapes.

There is not such a pressing need as with other cheap machines to increase the amount of memory available. FORTH programs are usually quite short, so a surprising amount can be put into the standard 3 Kbytes of memory. To write larger programs, more memory is needed. Both 16 Kbyte and 32 Kbyte RAM packs are available, and an adaptor can be obtained that will enable the machine to accept most Sinclair devices. A Centronics printer interface, a sound box and a device to improve the 'feel' of the keypads can all be obtained.

JUPITER ACE

PRICE
Obtainable second-hand from £40

SIZE
215 × 190 × 30 mm

WEIGHT
246g

CPU
Z80A

CLOCK SPEED
1MHz

MEMORY
3Kbytes RAM expandable externally to 51Kbytes: 8Kbytes ROM

VIDEO DISPLAY
Black and white, 32 × 22 rows of text, 64 × 48 graphics

INTERFACES
TV connector, cassette, power (9v), two edge connectors; first has complete address and data lines from the processor, second has data and some selection lines

LANGUAGES SUPPLIED
FORTH

OTHER LANGUAGES AVAILABLE
none

COMES WITH
Power supply in mains plug, cassette and TV leads

KEYBOARD
Rubber keypad similar to Sinclair Spectrum but softer and less accurate. Keys have to be pressed dead centre. All keys have auto repeat and two shifts allow all ASCII codes to be produced

DOCUMENTATION
Easily the best manual for a small computer and would put quite a lot of larger computer suppliers to shame. The author wrote both the ZX81 and the Spectrum manuals. In 180 pages there is an introduction to FORTH and a full description of the Ace, both supported by copious examples. The book is available separately for £5.95. The contents list annotates each chapter, there are four appendices giving quick references to all the facilities available and an index

Information about currently available add-ons and software for the Ace can be found in 'Forth User', obtainable from the Jupiter Ace Users' Club, c/o John L. Noyce, P.O. Box 450, Brighton BN1 8GR

Spirited Graphics

Large-capacity memories make it possible for home computers to produce colourful and fast-moving images

One of the most striking features of home computers is their ability to produce graphics and moving displays, commonly known as animation. On most microcomputers, the user can plot individual points, draw lines and circles, and change the background and foreground colours.

For fast-action games and simulations, we need to be able to simulate movement. The easiest way of doing this is to produce a series of still pictures, one after the other. This must be done rapidly enough to give the illusion of movement. Television pictures are produced using a similar method.

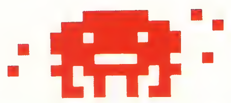
Speed Of Action

Another way of creating the illusion of movement is to print a character, erase it, and print it again in a position that is slightly displaced from the original. To achieve a smooth flow of movement, the distance moved at each step should be minimal. Similarly, the time taken to produce the shape and to blank it out should be as short as possible.

Using BASIC to produce animation results in slow movement. One way to overcome this is to resort to Assembly language, an approach that requires a lot of practice, care and attention if you want flicker-free displays. Added to this, we have the extra problem of controlling the individual characters, as more and more are added to the screen scene.

Several computers, notably the Commodore 64, Sord M5, Texas Instruments TI99/4A, and the Atari range, overcome this problem by offering animation that utilises the same techniques that the coin-operated arcade machines employ. The technique is known as 'sprite graphics'.

Sprites are 'objects', or shapes, that can be moved independently of each other about the screen. This is done simply by changing the contents of a couple of memory locations, which specify the X and Y coordinates (the left-right and up-down positions). Typically, X can range from 0 to 255, and Y from 0 to 191. Some systems even permit you to specify the speed and direction of movement of each sprite, and the computer does the rest.



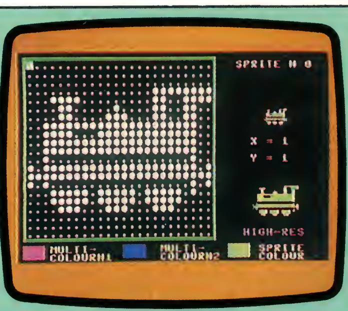
Sprites are normally implemented using special chips or hardware circuitry within the computer. It is possible to buy software for other computers to achieve a similar result, but this is generally less satisfactory.

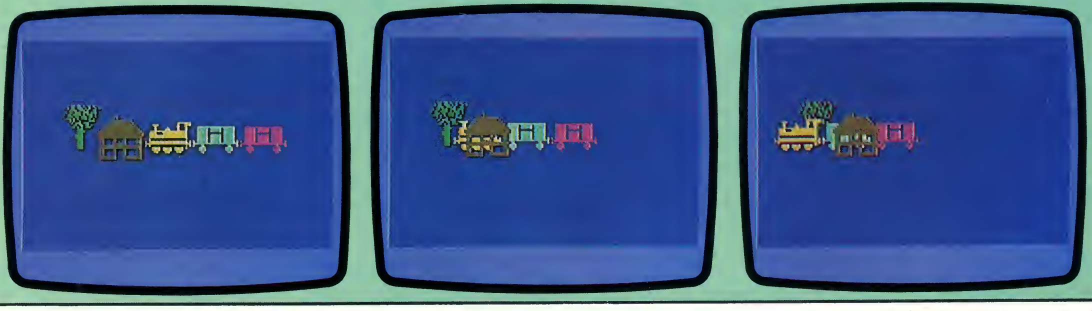
The speed at which sprites can move varies,

Train Of Thought

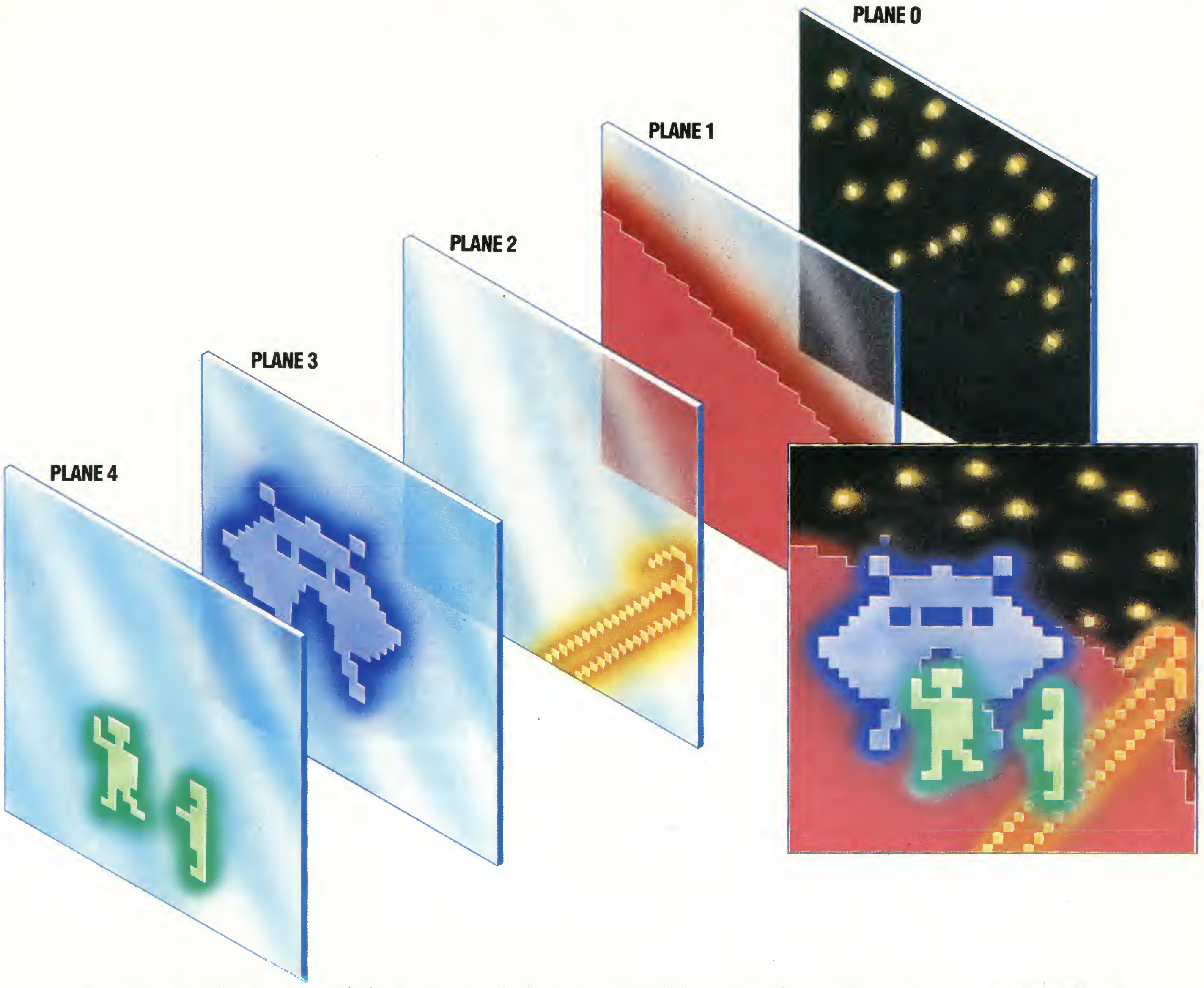
The train pictured was constructed as three sprites (engine plus two trucks) using a package called Spritemaker on the Commodore 64. The image was created in large-scale using the editing facilities of that package and then saved onto cassette, together with images for the house and tree.

Loaded back into the 64's memory, the sprites were then manipulated using POKE commands to fix their positions on screen, their colour, and the train's speed. The 'priority' of the sprites was specified so that the train would pass behind the house but in front of the tree





IAN McKINNELL

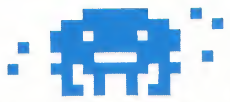


because each one resides on what is known as a 'sprite plane'. A screenful of graphics is thus built up from a number of planes stacked behind each other, though the naked eye perceives them as one screen.

A three-dimensional effect can be achieved as sprites can pass behind or in front of one another. Sprites are numbered from zero to the maximum number available, which is, for example, 32 on the Sord M5. If two sprites overlap, the one with the lower number will be displayed. Therefore by careful ordering of your sprites, a three-dimensional effect can be easily obtained, such as a train passing a tree, obscuring it as it passes by. Alternatively, the train itself may be partially blocked out as it passes a house with a lower number.

Using the colour range that your computer offers, each sprite can be coloured individually. Sometimes a sprite can be expanded or shrunk, again by changing the contents of a memory location.

Obviously, sprites really show their worth in games programs, and a particularly useful feature is known as 'collision detection'. When two or more sprites overlap (for example, when your missile connects with the enemy spaceship) the system can be programmed to jump to another part of the program to create the graphics for an explosion, and increase the player's score.



Before using sprites you have to create them in much the same way as you would design a new character. The letters of the alphabet, numbers, and special graphics symbols are stored inside the computer in a chip known as a 'character generator'.

Characters are, as we mentioned earlier in the course, generally built up on an eight by eight matrix of dots, or 'pixels'. The maximum size of a sprite varies from machine to machine, but will generally be several characters wide and deep. On the Commodore 64, for example, the maximum is 24 pixels wide by 21 pixels high.

The best way to construct a sprite is to draw up a

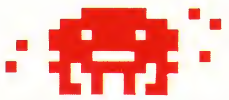
On Different Planes
Using sprite graphics, a complex image can be built up by placing component parts on different planes, which are stacked behind each other. Though they are viewed on the computer's screen as just one plane, the tremendous advantage of this system is that objects on different planes can move completely independently of each other. Planes are assigned an order of priority by the programmer, so that when two sprites overlap on the screen, the one with the higher priority is shown in front of the other — resulting in a three-dimensional effect.

Most computers with sprite graphics can also arrange for the program to be interrupted, whenever two sprites come into contact, to create the necessary explosion, or increase the player's score

Sprite Dimensions
Sprite graphics are available on only four of the popular home microcomputers, the Commodore 64, Sord M5, TI99/4A and the Atari range.
The 16 colours normally available are increased to 256 on the Atari because each colour comes in 16 degrees of 'luminosity'.
In games it is often useful to know when two sprites come into contact — for example when two spaceships crash — and for this a 'collision detection facility' is provided.
The 3-D effect is created by placing each sprite on a different plane; the more planes in a system the better. The maximum size of each sprite is expressed in pixels. Sprites can be made to expand or contract and can be moved about the screen.
To simplify the process of building up sprites special software 'utility packages' are available

MICROCOMPUTER	NUMBER OF COLOURS	COLLISION DETECTION	PLANES	SPRITE SIZE	SOFTWARE
Commodore 64	16	Yes	8	24 × 21	Yes
Atari 800	256	Yes	16	16 × 16	Yes
Texas TI 99/4A	16	Yes	28	32 × 32	Yes
Sord M5	16	Yes	32	8 × 8	No

grid of squares to the maximum size, and make the required shape by filling in the appropriate blocks. You will already be aware that computers work using only 1s and 0s (see page 28). The black squares should be represented as 1s, and the white squares as 0s.



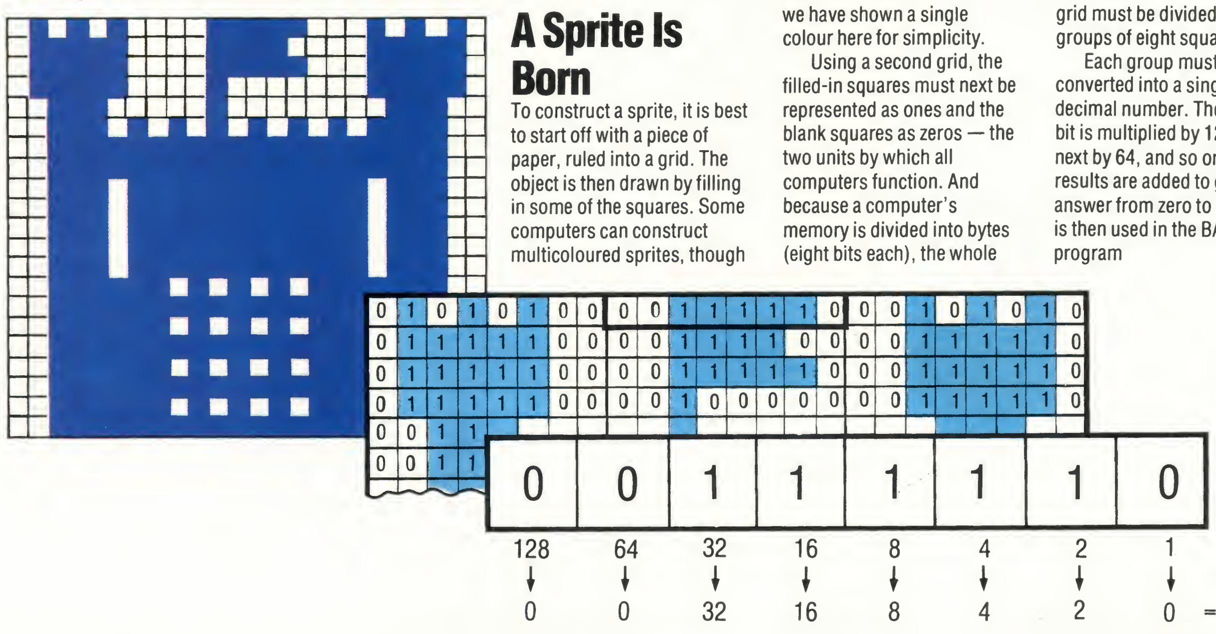
The computer handles most information in the form of bytes (a collection of eight bits). The computer's manual will show how the whole grid of dots making up a sprite needs to be broken up into groups of eight. Each byte in turn needs to be converted into a single decimal number, ranging from 0 to a maximum of 255, before it can be used in a BASIC program. This is done by multiplying the leftmost binary digit (bit) by 128, the next one by 64, and so on. The results are then added together.

The resulting collection of decimal numbers completely define the design of the sprite. These numbers are placed into memory locations using a BASIC program; the exact procedure varies with each machine. The computer then needs to be instructed as to where in memory it can find the specifications for each of the sprites required.

Everything else can now be achieved using simple commands to specify the current position of each sprite on the screen, change its colour, expand or contract it, and detect when two or more sprites overlap.

Software packages, called 'utilities', are available for most machines with sprites. These make the process of creating the image less tedious. They show the grid on the screen in large scale, and allow the image to be drawn simply by moving the flashing cursor around the grid. All the arithmetic is handled automatically, and the results are then placed in the appropriate bytes. Finally the grid disappears and the sprite itself appears on the screen for manipulation.

Sprites have revolutionised home computer graphics by providing a simple and efficient method of producing fast-moving and colourful displays.



A Sprite Is Born
To construct a sprite, it is best to start off with a piece of paper, ruled into a grid. The object is then drawn by filling in some of the squares. Some computers can construct multicoloured sprites, though we have shown a single colour here for simplicity.
Using a second grid, the filled-in squares must next be represented as ones and the blank squares as zeros — the two units by which all computers function. And because a computer's memory is divided into bytes (eight bits each), the whole grid must be divided into groups of eight squares.
Each group must be converted into a single decimal number. The leftmost bit is multiplied by 128, the next by 64, and so on. These results are added to give an answer from zero to 255. This is then used in the BASIC program

MARK WATKINSON

Steve Wozniak

APPLE COMPUTER INC.

From electronic prankster to multi-millionaire: the story behind the creation of the top-selling Apple computers

In the United States, Steve Wozniak is best known to the public for staging vast outdoor rock concerts, not as the man who built Apple I and II singlehanded, in his garage. But in the computer business he is known as the 'electronic wizard' who did more to simplify and popularise the microcomputer than anyone else. Wozniak's machines were the first to have colour, graphics, a keyboard and a video, all as standard features. And the Apple II has been so popular that one million have already been sold.

Steve Wozniak's meteoric rise from garage engineer to dollar billionaire reads like a modern Californian fairy tale. He was born and brought up in the now famous birthplace of 'the chip', Silicon Valley, California. His father was a professional engineer. Although he did teach his son the most elementary rule of electronics, 'Ohm's Law', in the main Wozniak taught himself electronics.

His early years were spent playing with electronic parts and using his technical genius to play pranks at school. On one occasion he built an electronic device called a blue box: an idea he got from a technological outlaw character in a magazine story. The device could mimic certain tones in the telephone system. These particular tones meant that the caller had inserted the required amount into the telephone's coin box. This enabled Wozniak to make free telephone calls all over the world. He talked to people in England, and even rang the Pope . . . direct!

Wozniak was never trained as a professional engineer; he excelled in maths and electronics at school, but dropped out of college. His first job was as a technician at the huge American corporation Hewlett Packard, designing calculators. They said he was not trained to do what he really wanted to do, which was design computers. So he started working on his own, usually at night, and designed a microcomputer that Hewlett Packard turned down. Undeterred, he left the company, and with his school friend (and partner in pranks) Steve Jobs, made and sold 50 of the design. The Apple I was born. They called the computer and the company Apple, simply because Jobs had once worked in an orchard.

Between 1975 and 1976 Wozniak locked himself in his garage. Here his creative bursts of work lasted days and nights on end before he finally produced the Apple II. He was 26 years old. The experts still regard the Apple as an amazing feat of brilliantly simple circuitry and design.

They say Steve Wozniak can read the system in the timing and circuit diagrams of a chip as easily as some people read fortunes in tea leaves. One of the important innovations in the Apple II was the simplifying of the disk drive. Before Wozniak it had needed 30 chips, but he redesigned it for a microcomputer to use only five. It was not so much that Wozniak invented anything completely new, but that he simplified all the parts and packaged them together so that everyone and anyone could use the computer at home.

Wozniak never intended to go into business. It was his partner Steve Jobs who was responsible for selling the Apples and for setting up the Apple Corporation. The company now has 3,300 employees all over the world and is said to have made at least 50 people millionaires. Wozniak owns just four per cent of Apple and has never been interested or involved in the management side. He still prefers to play with computers and dream up new ideas.

In summer 1983, after a two-year break, mainly organising rock festivals, Wozniak went back to Apple to work on some new projects. No one knows exactly what he is working on. Apparently he is writing some applications software. He is also involved in an Apple II project for a home video production and editing system with high quality graphics that can also do cartoons. But Wozniak is not just interested in making existing machines faster and better. He also believes that in the future it will be possible to design a very intelligent microcomputer. Using only a simple program, it would be able to learn anything. We shall have to wait to see what this imaginative and unconventional engineer comes up with next.

The Chairman
Steven Jobs, chairman of Apple Computer Inc. was Steve Wozniak's partner and school friend. If Wozniak was the electronic genius of the team, Jobs was the boy wonder businessman. In 1975 when the Apple I was designed, Jobs was 20 years old. He saw the potential for selling the machine, first to computer hobbyists, and then quickly saw it could be packaged for a new personal computer market. Wozniak was the Apple's creator, but Jobs was responsible for mass producing and selling it. Apple II was packed in a box complete, ready to be plugged in at home so anyone could use it. When Wozniak and Jobs were still in the garage workshop, the first money for the company was $1,350 from selling Jobs' VW van and Wozniak's programmable calculator. Soon though, Mike Markkula, a young millionaire, gave them his financial backing for what looked like an interesting project. When the company went public in 1980 it was the hottest issue around, and lived up to its potential by selling $583 million worth of computers in 1982 alone.

The phenomenal growth of the Apple Corporation has won it acclaim and awards in the United States. Steve Jobs made the cover of the prestigious 'Time' magazine, as one of America's youngest and most successful businessmen

Magic Wands

Light pens are computer add-ons that can be used to draw or mark on the screen. We discover how these remarkable devices work

'Keyboard phobia' is one of the commonest reasons why people are reluctant to get to grips with computers, either at home or in the office. Because the keyboard resembles a typewriter and they haven't learnt to type, and worse still it contains several keys with unfamiliar signs on them, they are afraid of making fools of themselves. The light pen is one solution to this problem (along with devices such as voice-input) though it does have other uses.

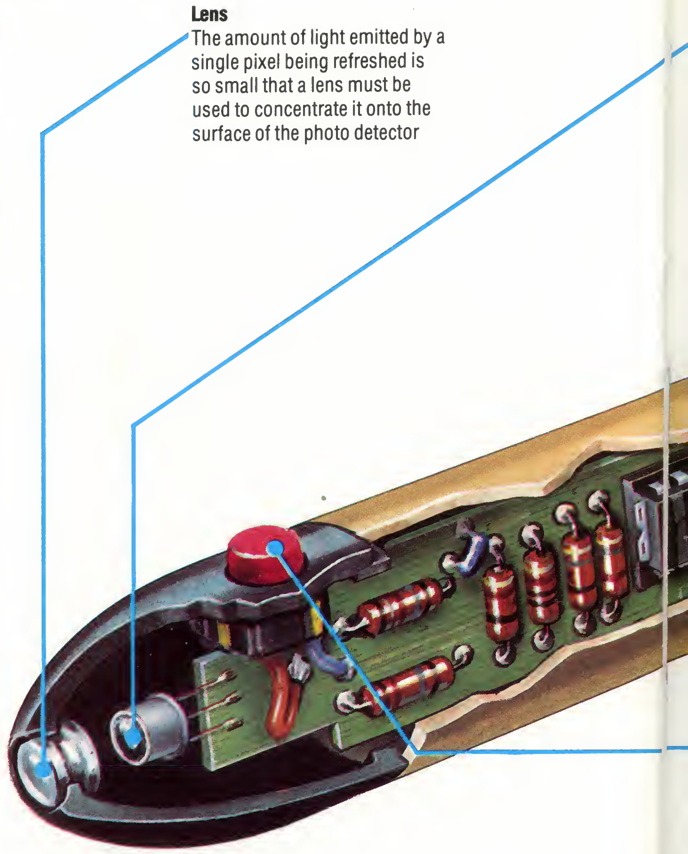
A light pen is a cylindrical device (looking much like an ordinary pen, from which its name is derived) with a telephone-style coiled wire coming from one end. At the other end of this wire is a plug, which fits into the back of the computer. Whenever the light pen is pointed directly at the screen (some systems require that the pen is pressed onto the screen, in order to activate a switch inside the pen), the computer can detect the exact position on the screen at which it is being pointed.

What is actually happening is that a photo-detector in the tip of the pen is responding with an electrical pulse every time it is passed by the point of light that continuously scans the whole screen to create the image. Circuitry inside the video controller chip calculates where the scanning point was at the time when the light pen emitted its signal.

The most common use for a light pen is therefore to select an item that is displayed on the screen. Knowing the dot at which the pen is pointing, the computer can deduce the character or symbol of which that dot is a part. Many applications programs feature 'menus' as part of their operation. A menu is simply a list of options displayed on the screen, from which the user must make a choice — just like in a restaurant. In a home finances program, the menu might be:

1) Make a payment
2) Examine bank balance
3) Enter a receipt
and so on.

Normally, the user would indicate his chosen course of action by pressing a single key (1, 2 or 3) or typing in a command word. With a light pen, he merely has to point to the required option. The computer usually responds by flashing that particular option to indicate that it has accepted the input. Some quite sophisticated programs are operated almost entirely by means of such menus (these are described as menu-driven), in which


Lens
The amount of light emitted by a single pixel being refreshed is so small that a lens must be used to concentrate it onto the surface of the photo detector

case the user need only touch the keyboard when actual data is needed, such as someone's name and address.

Special Routines

The difficulty is that such a program has to be written specially to work with the light pen, as distinct from the keyboard. This is only really a matter of writing a small routine that takes the co-ordinates indicating the current position of the light pen from the video controller, and works out which of the options occurs in that position on the screen. Unfortunately, few manufacturers of software supply versions of their packages that work with a light pen.

However, in addition to selecting items, a light pen can be used to create images on the screen. Most home computers for which a light pen is available have a suitable software package for this purpose. The user is presented with a blank screen on which to draw his diagram, picture or doodle, and a separate section (usually along the bottom of the screen) that provides a series of special functions to help in the creation process. One of these will be a palette of colour, which works just like the palette used by an oil painter. The light pen is placed over the colour next required by the user, and wherever the pen is subsequently moved over

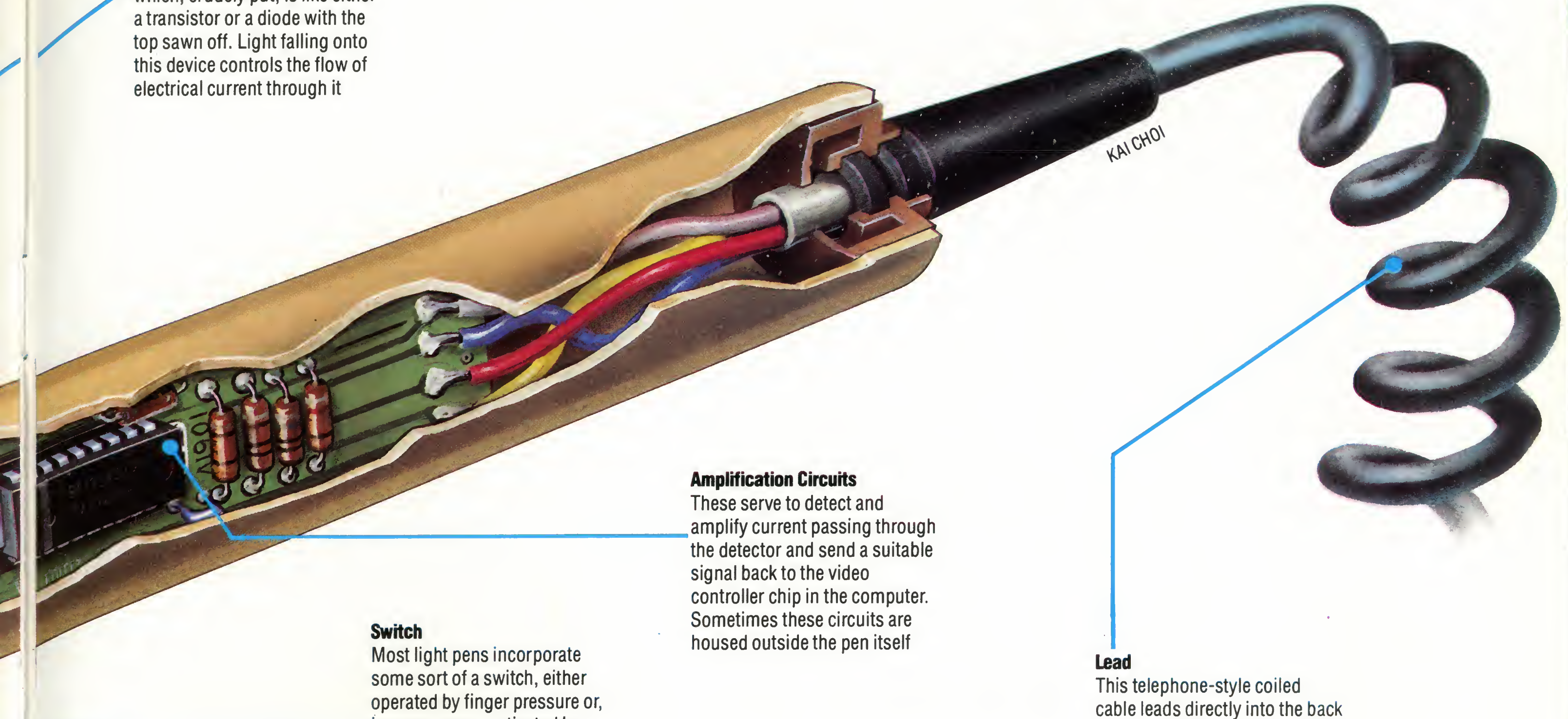
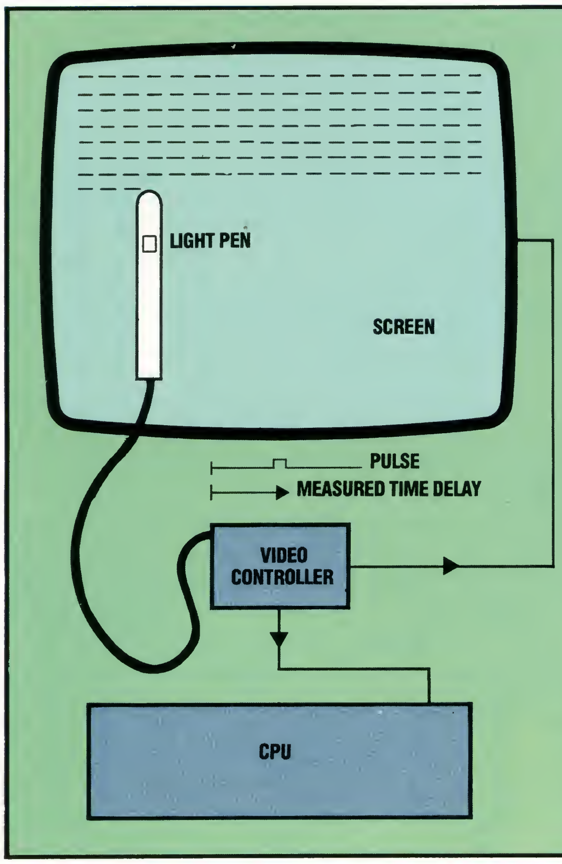


Photo Detector
This is a semi-conductor device, which, crudely put, is like either a transistor or a diode with the top sawn off. Light falling onto this device controls the flow of electrical current through it

Amplification Circuits
These serve to detect and amplify current passing through the detector and send a suitable signal back to the video controller chip in the computer. Sometimes these circuits are housed outside the pen itself

Lead
This telephone-style coiled cable leads directly into the back of the computer, and from there to the video controller chip

Switch
Most light pens incorporate some sort of a switch, either operated by finger pressure or, in some cases, activated by pressing the light pen onto the screen. The switch is needed so that the light pen doesn't react to light (such as room lights) when it is not being used to select an item on the screen



How It Works
The operation of a light pen is usually the responsibility of the video controller chip inside the computer. This chip produces video images on the television set or monitor, by scanning the screen many times a second with horizontal lines (this is called a raster scan). On British television sets there are 625 lines on the screen and the scan frequency is 50 per second. The output from the light pen also comes directly into this chip. When the light pen is pointed at an item on the screen and the switch is activated, the video controller merely measures the length of time from when it last started drawing the lines at the top of the screen, to when a pulse comes from the light pen. From this information it can compute which line the light pen is on, and how far along that line it is. This data is then converted into the X and Y coordinates needed by the program

the main screen, it leaves a line in that colour.

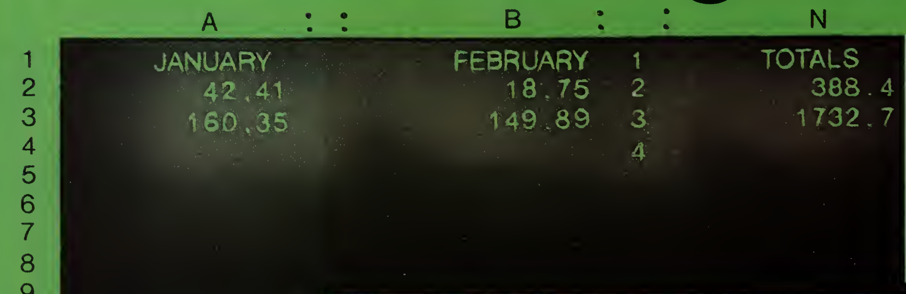
The user can also select different line widths or textures from the bottom of the screen to paint with, and has the ability to draw circles and squares. In short, everything that we discussed in 'The Electronic Artist' (see page 26) can be achieved with a light pen and probably a great deal faster than with a keyboard.

Games that make use of light pens are starting to appear on the scene, too. Zapping alien monsters is a great deal easier with a light pen than through the keyboard, so the games have to be made considerably harder. Rather more sedate games such as computerised chess can benefit — point to where you want the knight to move and the computer looks after the rest.

Perhaps the largest group of light pen users, however, is in engineering and design offices. Computer Aided Design (CAD) systems are featured widely in advertisements for new cars — in fact they are just like other computer systems, but with specialised software, and high-quality graphics. If a CAD system is being used to design a new electronic device, then the screen will feature representations of all the components that the designer may want access to, and these can be 'picked up' by the designer and placed where they are required anywhere on the screen.

The light pen is one of the best examples of a computer add-on that is both fun to use and of great practical value.

Planning Ahead



```
        A       : :        B       :  :      N        :
 1    JANUARY            FEBRUARY   1      TOTALS
 2       42.41              18.75   2       388.4
 3      160.35             149.89   3      1732.7
 4                                  4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
                                                FORM:SUM(A2:M2)
```

Fields Of Play
A spreadsheet is divided into rows and columns, and the intersection of a row and column is called a 'field' or 'cell' that can be addressed by its coordinates (A1, B3 etc.). Each field may contain a title (e.g. JANUARY), a number (e.g. 149.89) or a formula. Field N2 contains the formula 'SUM(A2:M2)', which is the sum of the top row of figures from January to December. The result of this calculation is displayed: 388.4. Note that the spreadsheet has been divided into two windows so that March to December are not currently visible

A 'spreadsheet' program can help you put your computer to profitable use. Planning, budgeting and forecasting are its main applications

It has been estimated that managers spend anything up to 30 per cent of their time preparing budgets – an activity that always calls for many 'what if...' questions to be asked. Traditionally, a sheet of paper, usually the size of a whole double-page spread in The Home Computer Course, was used. This was ruled vertically into a dozen or more columns, each of these columns was headed with a month number, and all the types of expenditure entered down one side. In each column the month's expenditure in the various categories was entered. By this means, one could sum the columns to arrive at a total expenditure by month, and add up the rows to find the total spending under one heading for the whole year.

The problem came when one had planned to spend too much — or worse still, too little — and had to go back and alter a large number of figures, and re-calculate the row and column totals accordingly.

By using a spreadsheet program, one is able to re-calculate the entire page of figures every time a single basic element is altered. If, for example, the cost of Transport in January is changed, this alteration will cause the total expenditure for that month to change, and the whole of the expenditure under that heading to change with it — at the touch of a key. No wonder that spreadsheet packages are the world's largest selling type of software!

In common with most pieces of commercial software, spreadsheets are normally written with 'overlays' — that is not all of the program is actually resident in the machine all the time. If you think of the program as being divided into subroutines (see page 77), a subroutine that is not required in the current operation will not be called up from backing storage (disk, or perhaps tape) until a 'call' is made into it. The operating system will then *lay* that subroutine *over* one that has become redundant (hence, overlay). As you can imagine, this method of stretching available memory is very useful indeed, but it does mean that one is often waiting while the back-up storage medium is transferring information into main memory.

Spreadsheet packages — you can usually spot them by their name, which more often than not ends in 'calc' — are available for a wide variety of home and business computers. Most popular, in terms of the number sold, is Visicalc, originally written for the Apple II and made available in mid-1979. The world of microcomputer software is never slow to react to the success of one of its members, and here was no exception. Before you could say 'spreadsheet' they were everywhere, for every type of machine, appropriate to the purpose or not.

To be of any real use a spreadsheet has to have two attributes — size (not necessarily what you see on the screen, because, as we will see later, what you see is just a 'window' on the whole), and a good range of formatting and control commands.

This means that there are severe limitations on the sorts of machine that could hope to run a spreadsheet program to its best advantage. As a rule of thumb, 32 Kbytes of RAM and an 80-character screen are minimum requirements for a business application, though a 40-character screen would probably suffice for domestic purposes.

Home computer users will find that many of the packages available for cassette-based machines like the Spectrum or Vic-20, while naturally being limited in size and power, will prove very useful.

Because spreadsheets have the ability to answer 'what if...?' questions, they can obviously be used to set up simple computerised models (see page 101). We gave there an example of the work of a systems analyst, and should you use a spreadsheet package yourself, the need for similar careful planning will soon become obvious. Databases, we have noted, consist of a mass of raw information that is ordered according to the user's requirements when the data is retrieved. Word processors, the other big-selling type of software, exist to allow the user to shift around single words or whole blocks of text at run time. But spreadsheets are a little different, in that they really do require the user to go through a planning process.

For instance, if you are using a spreadsheet to

The Bottom Line
The cursor is the rectangular block currently occupying field N2. If something is typed it will appear in the field where the cursor has been positioned. The full contents of that field will also be displayed on the command line of the spreadsheet, which in this case is at the bottom of the page

analyse your living costs, you might well wish to group together all expenses relating to the house — rent or mortgage repayments, rates, insurance, etc. and then use the result of that calculation in a larger table within the same sheet. You must be careful to sum all the household expenses before carrying forward any figure into the larger table.

Each individual location of the spreadsheet, known as a cell, is addressed and located by its X and Y coordinates. Horizontally they use the letters A – Z, AA – AZ, and perhaps BA – BM, to allow for the total possible width of the sheet — 65 cells in popular versions. Vertically, numbers 0 – 256 might be used. Each cell may contain a label (such as 'Sales' or 'Profit'), a value that is either entered or derived from a calculation (such as 1,000) or the formula for that calculation, such as B4+B6*B5. Formulae, because they often exceed the displayed size of the box, are usually displayed on a separate line at the top of the screen. On starting a fresh spreadsheet, the size of the cell will be preset, perhaps to eight or nine digits or characters. This is known as the default size. You are normally allowed to shorten or lengthen the cells to suit the type of calculation you are doing. Some packages will allow the leftmost column (usually your titles or descriptions) to be wider than the others. And you don't have to decide immediately how big you want your cells to be. Most versions allow you to expand or contract them even when you've got data in them. Should you reduce the size below that of the length of the contents, the non-displayed part is not erased, but only lost from view.

The last major component of the sheet is the command line, which appears at the top or bottom of the screen in response to the 'command' key: / for example. These commands are for use in formatting and manipulating the layout of the piece itself, not the data, although they may well affect how it appears. Most popular versions of spreadsheet software allow a wide variety of operations on the database. You can, for instance, clear, move or copy whole rows or columns. You can split the window to display together parts of the sheet that are normally too far apart for the window to cover, and you can then scroll (move across) these windows separately.

Normally, movement between cells is by means of the cursor control key, but yet another command key allows a jump to a specified cell. Loading and saving, clearing, protecting are all performed by the use of command keys, and while on this subject it is worth stressing yet again the importance of regularly saving one's work. It takes a longish time to set up a spreadsheet — normally a lot longer than it does to enter the data. As a general rule, *always* save a sheet before you start to enter into it. Then, if you do make some ghastly mistake, you will have minimised your loss.

Results are transmitted to the printer on a command key, but care must be taken to define which part of the sheet one wants printed by use of the parameters. Just as the screen is a window, or portion of the whole sheet, so, of course, will be the printed page from an output printer. If you need to print a sheet that is wider than your printer, the recommended course is to do two print runs and then stick the pages together.

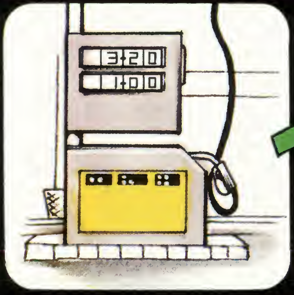
The use of windows, as we have noted, allows one to have two different parts of the sheet displayed at the same time. A sheet with a split window can also be printed. This is particularly useful when entering information, as one can refer back to any earlier entry. Most packages allow the user to 'hold' either or both the top and first lines, useful for the same reason, as these usually contain the titles or labels.

Up until now, we have considered the sheet as a table, which one could access only serially (one item after the next, along a row or column), but if one were to dispense with the luxury of summing rows or columns, there is little reason why the sheet should not be laid out in any pattern, if that pattern helped ease the user's thought paths. Bear in mind, though, that such sophistication within the database will require an even more thorough job of systems analysis than ordinarily.

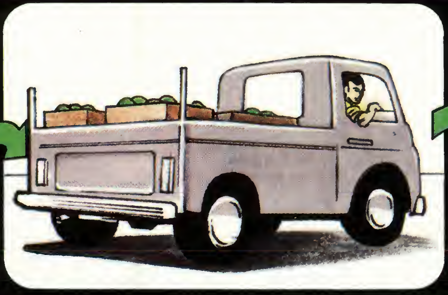
The business-oriented spreadsheets, like Visicalc, Supercalc and Masterplan, all offer the

MARK WATKINSON

What If . . . ?
If the contents of any field are changed, the spreadsheet will automatically recalculate all other fields which in some way depend on that figure. The speed and ease of this process, encourages the user to test out the validity of his budgets, forecasts and projections to see what happens to, say, the overall profit if certain conditions change. Take this example of a fruit seller...



If the price of petrol were to rise by X%...


Then the monthly transport costs would go up by Y%...

This in turn would increase the wholesale cost of apples...

Which would be passed on to the customer in the form of higher prices...

KEVIN JONES

user the ability to pass information from the spreadsheet to word processing or database management packages, and there are many complementary programs available to allow output to be in a variety of graphic forms: pie-charts, for example, or bar charts.

We have already mentioned one possible application of spreadsheets to the home computer user: analysing domestic expenses and budgeting. Another very appropriate use in the home might be in the installation of central heating, where there are a large number of variables to be taken into consideration: the type of fuel to be used, number and type of radiators, boiler output, etc. In fact, any decision-making process can be helped considerably by the use of a spreadsheet, not least because the user is almost forced to cover any and all possibilities.

Perhaps the most impressive feature of a business oriented spreadsheet run on an appropriate machine is the sheer speed of its operation. This is a function of machine code programming, and, not surprisingly, the speed of a package written in BASIC for a small home computer might, in contrast, give some cause for alarm.

It is perhaps worth considering some of the problems that one would encounter if one were to attempt to write such a program in BASIC, if only as an indication of the complexity of the task.

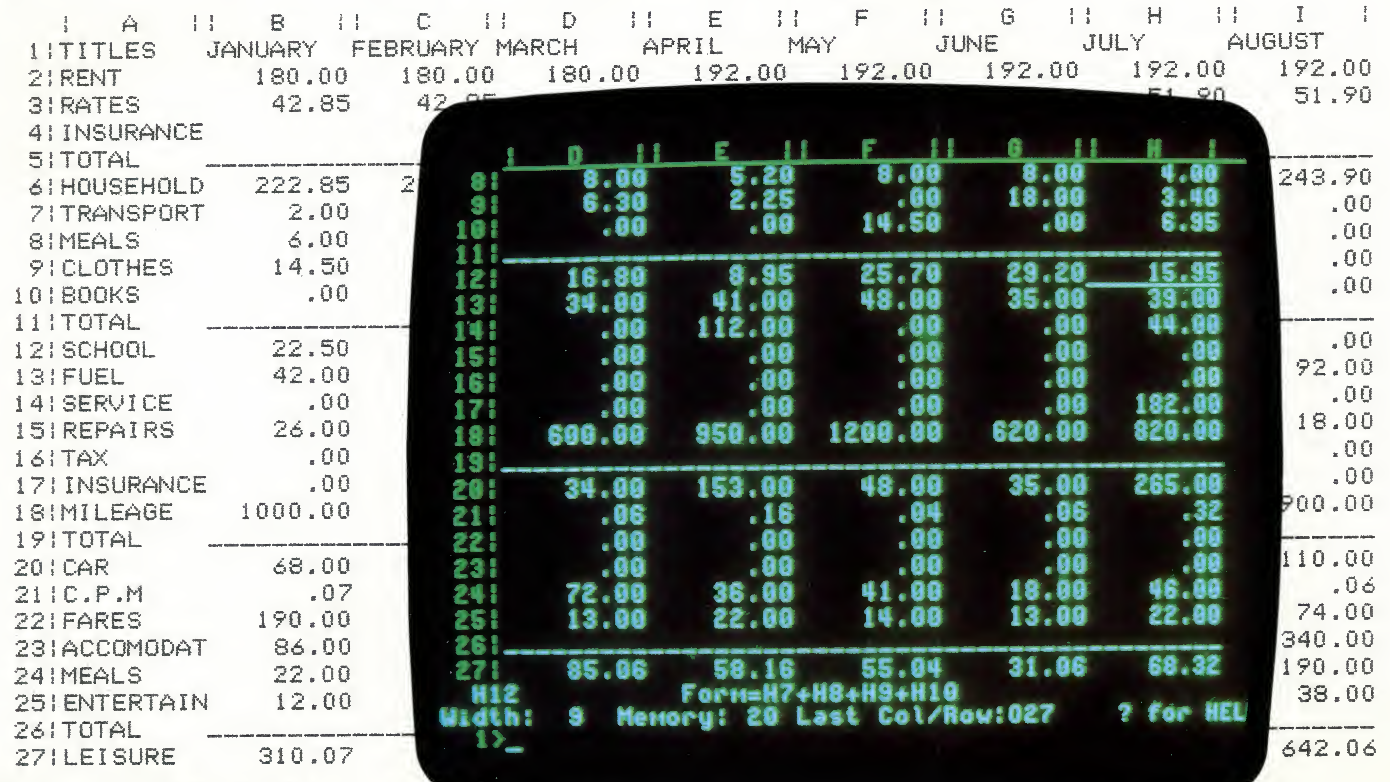
To start with, each cell must be defined in three ways. It must be capable of holding 'string variable' data, like 'January' or 'Rates'; it must be capable of holding numeric data for use in arithmetic operations, the amount of rates paid in January, for instance; and it may be required to hold a formula which is, in essence, a line of programming code, like 'annual rates/12' to give the monthly rate. Then, each cell must be expandable and contractable in size, but without losing the least significant part, so they must all be duplicated: one to appear on the screen, the other to always hold all the data, hidden away inside the program.

As you can see, the data handling alone is a very complex task — and remember that the more sophisticated packages may have up to 16,000 individual cells! The techniques used in writing such software are very much akin to the writing of interpreters for languages such as BASIC or FORTH, and similar techniques are also used in database management software.

All this goes some way towards explaining the high cost of such purpose-written business software. A package like Visicalc or Supercalc may cost hundreds of pounds, but one should perhaps bear in mind the savings that its use can bring about. The fairly simple application that we considered earlier, that of a manager compiling budgets, for instance, can lead to savings of perhaps 15 or 20 per cent in a year. The cost of the software is a very insignificant part of such an economy, even before one starts to think about the savings in time and energy. Indeed, the Visicalc package has probably been the Apple salesman's best friend.

Windows On The World
Just as we can move the cursor around the screen, left or right, up or down, under keyboard control, in a spreadsheet we can move the screen around the sheet. This sophistication allows an area much larger than that of the screen to be displayed.

Some portable computers with limited screen size, like the Epson HX-20 and the Osborne-1, use this same system in all their operations



```
  ¦    A     ¦¦    B    ¦¦    C     ¦¦    D    ¦¦    E    ¦¦    F    ¦¦    G    ¦¦    H    ¦¦    I    ¦
 1¦TITLES     JANUARY   FEBRUARY MARCH      APRIL      MAY         JUNE       JULY       AUGUST
 2¦RENT        180.00    180.00    180.00    192.00    192.00    192.00    192.00    192.00
 3¦RATES        42.85                                                                  51.90
 4¦INSURANCE
 5¦TOTAL      ---------                                                              ---------
 6¦HOUSEHOLD   222.85                                                                 243.90
 7¦TRANSPORT     2.00                                                                    .00
 8¦MEALS         6.00                                                                    .00
 9¦CLOTHES      14.50                                                                    .00
10¦BOOKS          .00                                                                    .00
11¦TOTAL      ---------                                                              ---------
12¦SCHOOL       22.50                                                                    .00
13¦FUEL         42.00                                                                  92.00
14¦SERVICE        .00                                                                    .00
15¦REPAIRS      26.00                                                                  18.00
16¦TAX            .00                                                                    .00
17¦INSURANCE      .00                                                                    .00
18¦MILEAGE    1000.00                                                                 900.00
19¦TOTAL      ---------                                                              ---------
20¦CAR          68.00                                                                 110.00
21¦C.P.M          .07                                                                    .06
22¦FARES       190.00                                                                  74.00
23¦ACCOMODAT    86.00                                                                 340.00
24¦MEALS        22.00                                                                 190.00
25¦ENTERTAIN    12.00                                                                  38.00
26¦TOTAL      ---------                                                              ---------
27¦LEISURE     310.07                                                                 642.06
```

```
   ¦    D    ¦¦    E    ¦¦    F    ¦¦    G    ¦¦    H    ¦
  8¦     8.00      5.20      8.00      8.00      4.00
  9¦     6.30      2.25       .00     18.00      3.40
 10¦      .00       .00     14.50       .00      6.95
 11¦------------------------------------------------------
 12¦    16.80      8.95     25.70     29.20     15.95
 13¦    34.00     41.00     48.00     35.00     39.00
 14¦      .00    112.00       .00       .00     44.00
 15¦      .00       .00       .00       .00       .00
 16¦      .00       .00       .00       .00       .00
 17¦      .00       .00       .00       .00    182.00
 18¦   600.00    950.00   1200.00    620.00    820.00
 19¦------------------------------------------------------
 20¦    34.00    153.00     48.00     35.00    265.00
 21¦      .06       .16       .04       .06       .32
 22¦      .00       .00       .00       .00       .00
 23¦      .00       .00       .00       .00       .00
 24¦    72.00     36.00     41.00     18.00     46.00
 25¦    13.00     22.00     14.00     13.00     22.00
 26¦------------------------------------------------------
 27¦    85.06     58.16     55.04     31.06     68.32
  H12           Form=H7+H8+H9+H10
Width:  9  Memory: 20 Last Col/Row:027      ? for HEL
  1>_
```

Double Vision

Most people think that we will have to wait for some major breakthrough before we get 3-D television. This is not so. With your computer, you can already produce quite respectable three-dimensional images on a perfectly normal colour television.

Admittedly, they are not in full colour, and you will only be able to show the edges and corners of an object, but you can certainly perceive true depth.

It should be made clear that the term 'three-dimensional' is a description often applied to commercially available programs. But in almost all cases, this is less than accurate: the product on offer is in reality an ordinary 'flat on the screen' game, to which has been added some perspective or shading.

The genuine three-dimensional display, however, adds a third direction of motion, in-out. This is achieved by using a pair of 'two-coloured' spectacles, where one lens is red, and the other is cyan (greenish-blue). It is also used in picturebooks and certain sensational movies.

The way to tell whether you are looking at 'fake' 3-D or not is by colour. Any true three-dimensional display will not make much sense if viewed without the glasses, since it will consist largely of lines drawn in two different colours, red and cyan. These will result in a confusing mess of intersecting lines.

An image that contains any colours but these, and in particular, one that is easily comprehended, is not likely to be truly three dimensional. Even if it says so on the packaging, there will be no true depth to the game. Motion will only be possible in two planes, up-down and left-right.

It is quite simple to produce an image that will persuade your eyes and brain that you are looking at a truly three-dimensional scene, though the mathematics can be a little involved.

The first necessity is to provide a slightly different image for each eye. Since the eyes will view the scene from separate positions, the two images are drawn with slightly different perspectives and angles. These images must not interfere with each other, even though they are both on the same screen. This is done by drawing one in red, and the other in cyan.

If the picture is viewed through a pair of two-coloured spectacles that have the cyan filter in front of the left eye, the red image (intended for the right eye) will become almost completely invisible to the left eye. The reverse applies to the cyan image (intended for the left eye). By this process, we can isolate the two pictures. The result will be a fairly convincing three-dimensional image.

The disadvantage of this system is that, though we have used colour to isolate the images from each other, the picture is effectively monochrome. Adding red to cyan will produce a more or less white image, which is what your brain perceives.

Another small problem becomes apparent when we consider what must happen when an object has to move 'into' or 'out of' the screen. An object at a distance of three metres will not appear twice as large as something that is six metres away. The sizes must be calculated by trigonometry, which can be done by using the tangent function (TAN in most BASICS).

Calculating values and converting them into coordinates for screen-drawing involves several calculations with decimal numbers. This is quite slow, so there is a limit to how complicated a display can become, or how fast it can move. Too many displayed objects and associated calculations can slow things up to the point that the motion becomes unpleasantly jerky and slow.

Another problem is that the resolution of the average computer screen is fairly low. In other words, there is a limit on how small an object can be before it collapses into a shapeless blob. At this distance the object has effectively reached infinity, and cannot be moved any deeper.

Not only that, but in order to isolate the two images, one must be displaced from the other by at least one dot, so that they can have different colours. Moving too 'deep' into the screen will produce the same values from the calculations for each image, and so the two will superimpose. When this happens, the object will lose all 'solidity' and become a flat object.

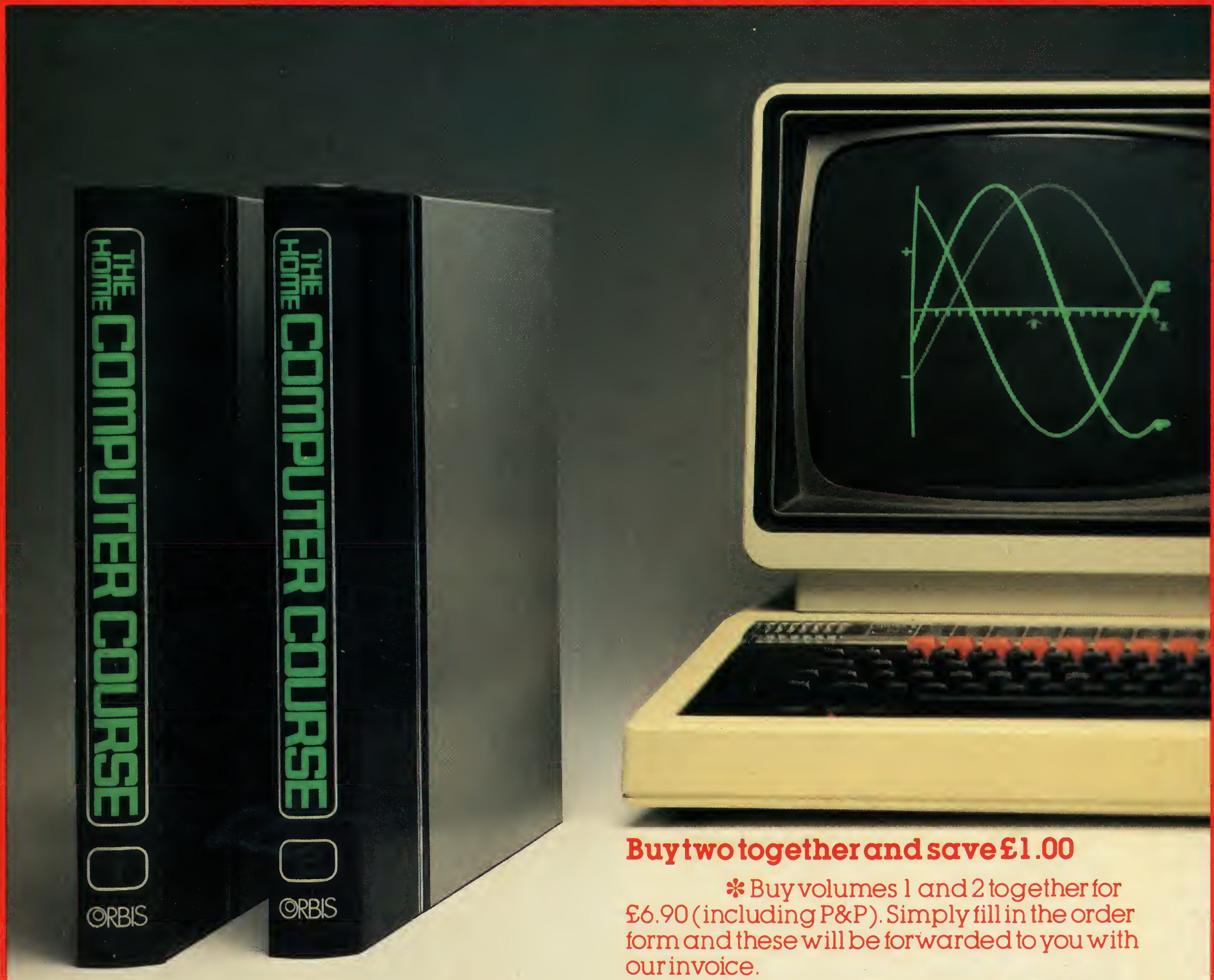
An important element in the production of high-quality, three-dimensional images is the quality of the screen display. The best results will be obtained with a monitor rather than an ordinary television. This is because the colours will be clearer and have sharper edges on a monitor. For more information, refer to the feature on monitors on page 132.

There are not many programs that use this technique. Owing to the limitations of the system, in particular the lack of colour and the need to wear the spectacles, it is unlikely to attain wide popularity. However, within these limits, very effective and entertaining displays can be produced.

Solid Gold
Perhaps Mike Singleton has spent so much time wearing red and green spectacles while developing 3-D games that his brain is now processing their coloured images as normal...In any event, he has really opened up a new dimension for home computer games players with his application of a 30-year old comic-book technique to computer graphics

TONY SLEEP/LAURIE-RAE CHAMBERLAIN

THE HOME COMPUTER COURSE BINDER



Now that your collection of Home Computer Course is growing, it makes sound sense to take advantage of this opportunity to order the two specially designed Home Computer Course binders.

The binders have been commissioned to store all the issues in this 24 part series.

At the end of the course the two volume binder set will prove invaluable in converting your copies of this unique series into a permanent work of reference.

Buy two together and save £1.00

✻ Buy volumes 1 and 2 together for £6.90 (including P&P). Simply fill in the order form and these will be forwarded to you with our invoice.

✻ If you prefer to buy the binders separately please send us your cheque/postal order for £3.95 (including P&P). We will send you volume 1 only. Then you may order volume 2 in the same way – when it suits you!

Overseas readers: This binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in Issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain their binders **now**. For details please see inside the front cover.

Binders may be subject to import duty and/or local tax.

THE LAST WORD IN LOGIC