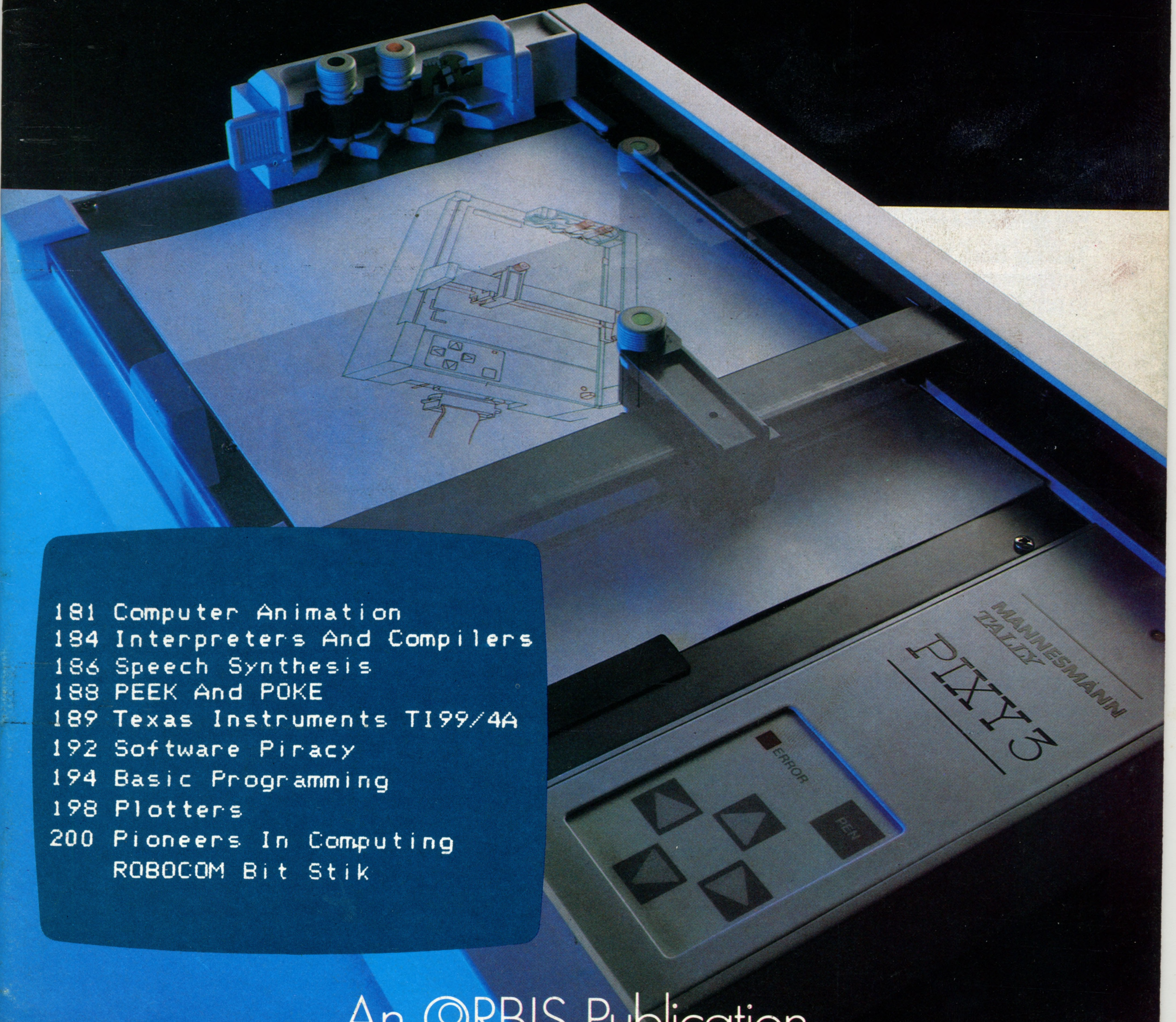
ISSN 0265-2919

R 80p

THE HOME COMPUTER COURSE 10

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

Hardware Focus

Software

Basic Programming

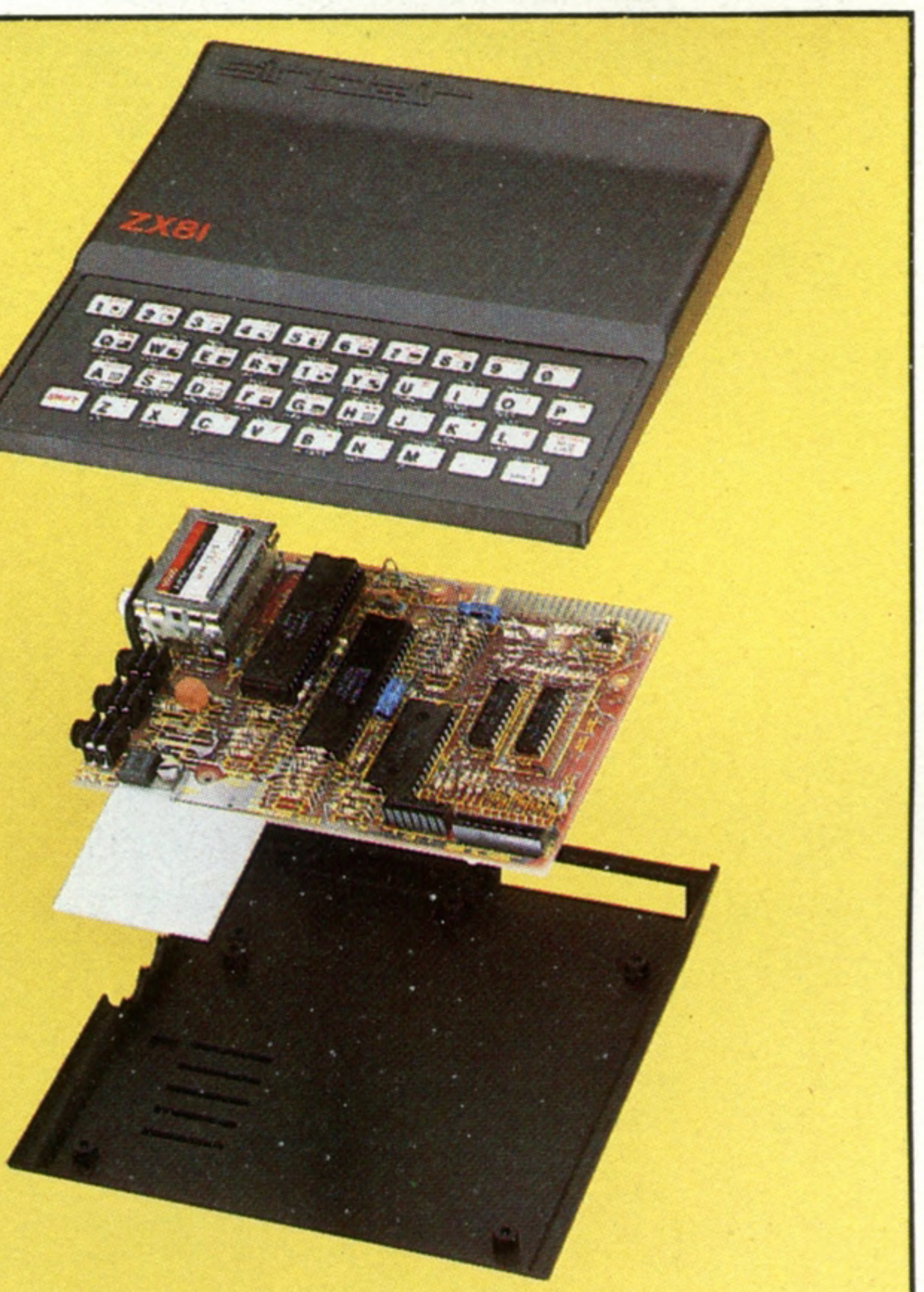
Insights

Passwords To Computing

Pioneers In Computing

Next Week

- The Sinclair ZX81, first mass-marketed micro and performing miracles on just four chips, can still be viewed as a triumph of design
- Flight simulators for pilot training make heavy use of computers and cost thousands of pounds. Some of the work put into their development is now appearing in the form of sophisticated games for the micro
- One way of sending data from your computer to another over the telephone is by means of an acoustic coupler. We explain how this device works



Editor Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistants** Liz Dixon, Safu Maria Gilbert; **Sub Editor** Tracy Ebbetts; **Researcher** Melanie Davis; **Contributors** Tim Heath, Richard King, Henry Budgett, Brian Morris, 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; Editorial Office** 39 Goodge Street, London W1; **© 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: Details of how to obtain your binders (and of our special offer) are in issues 4, 5, 6, and 8. 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.

Moving Pictures

Take the graphics on your computer, multiply the quality by a thousand, and you have a computer animation system

The entire process of producing moving pictures, whether on film or for television, relies on the brain's inability to 'freeze' an image. By presenting the eye with a rapid succession of images an impression of motion is created.

The first attempts at producing the illusion of movement in pictures involved piercing a drum with slots, pasting a strip of drawings around its inside, and spinning the drum. Looking through the slots, one sees a crude representation of one picture or 'frame' after another. The Zoetrope, as it was called, predates the science of photography, but naturally photographs soon replaced the drawings on the inside of the drum. The next stage, the motion picture, required relatively fast-acting photographic emulsions, capable of recording an image in less than one sixteenth of a second, since the early films were projected at 16 frames per second.

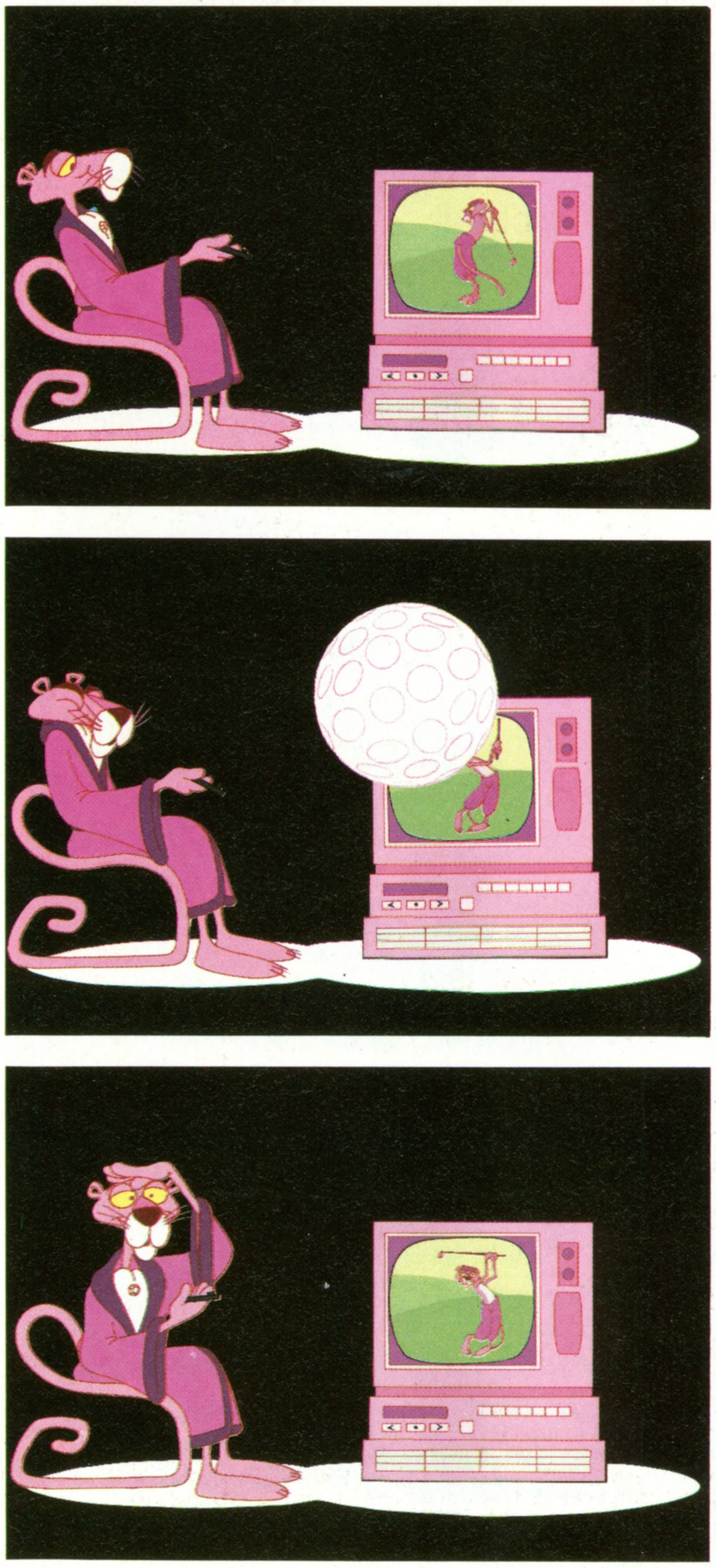
Simulating Movement

Strangely, it was quite some time before the film industry conceived the idea of hand-drawing each frame, photographing the drawings and then projecting the result to produce animated cartoons. Bearing in mind that each second's viewing requires the creation of 24 separate drawings (the projection speed of modern film), it is clear that the production of even a five-minute film requires a tremendous amount of work — 7,200 frames in this case. It is not surprising that the style of illustration is formalised — the most important requirement is precise repeatability. It wouldn't do to have Bugs Bunny looking different from one second to the next!

Repetitive and precise tasks like these are readily performed by machines. When the computer takes over the job of animation — adjusting speed of movement, changing perspectives and geometry, lighting and shading, changes in volume, rhythm and pace — the artist is then free to concentrate on the quality of the image. At this point animation changes to being a true graphic art, where the artist's time is spent in creating the image that the computer will cause to move.

In its simplest form, this process uses sprite graphics (see page 152) to create the 'cast of characters', which are then transferred onto the screen and moved about, producing the sort of animation used in simple video games. To create the illusion of change as well as movement (for example, someone walking) it is necessary to repeatedly substitute one sprite for another. As we saw, the creation of sprites is a comparatively slow business, given the graphic quality of the results, and the image has to be nothing more than a very simple two-dimensional representation.

The next stage of animation requires the animator/programmer to construct an algorithm that introduces a feeling of depth into the image according to the rules of perspective. Objects can then be defined on the screen in terms of their X, Y



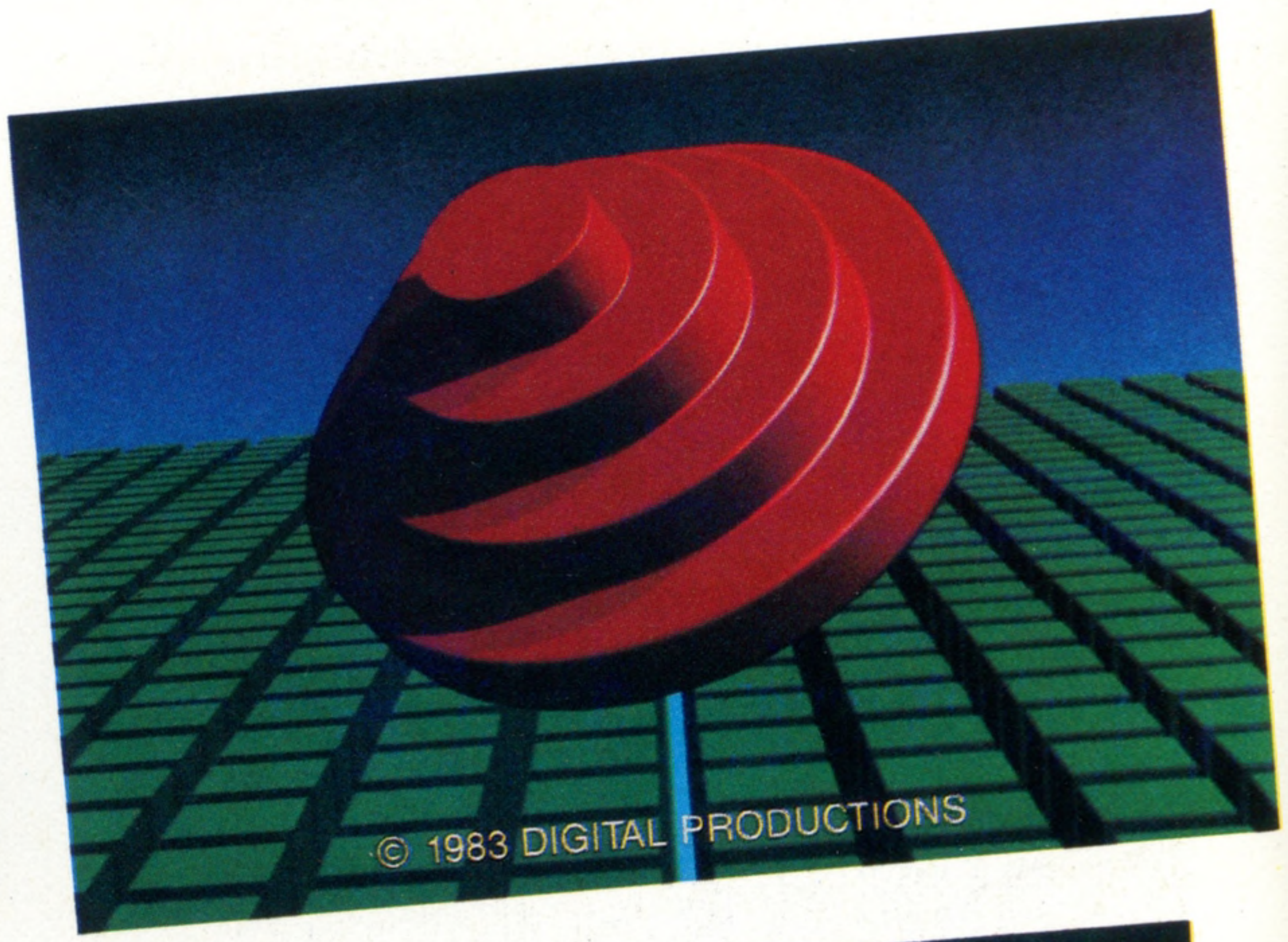
COURTESY OF RICHARD WILLIAMS ANIMATION LTD.

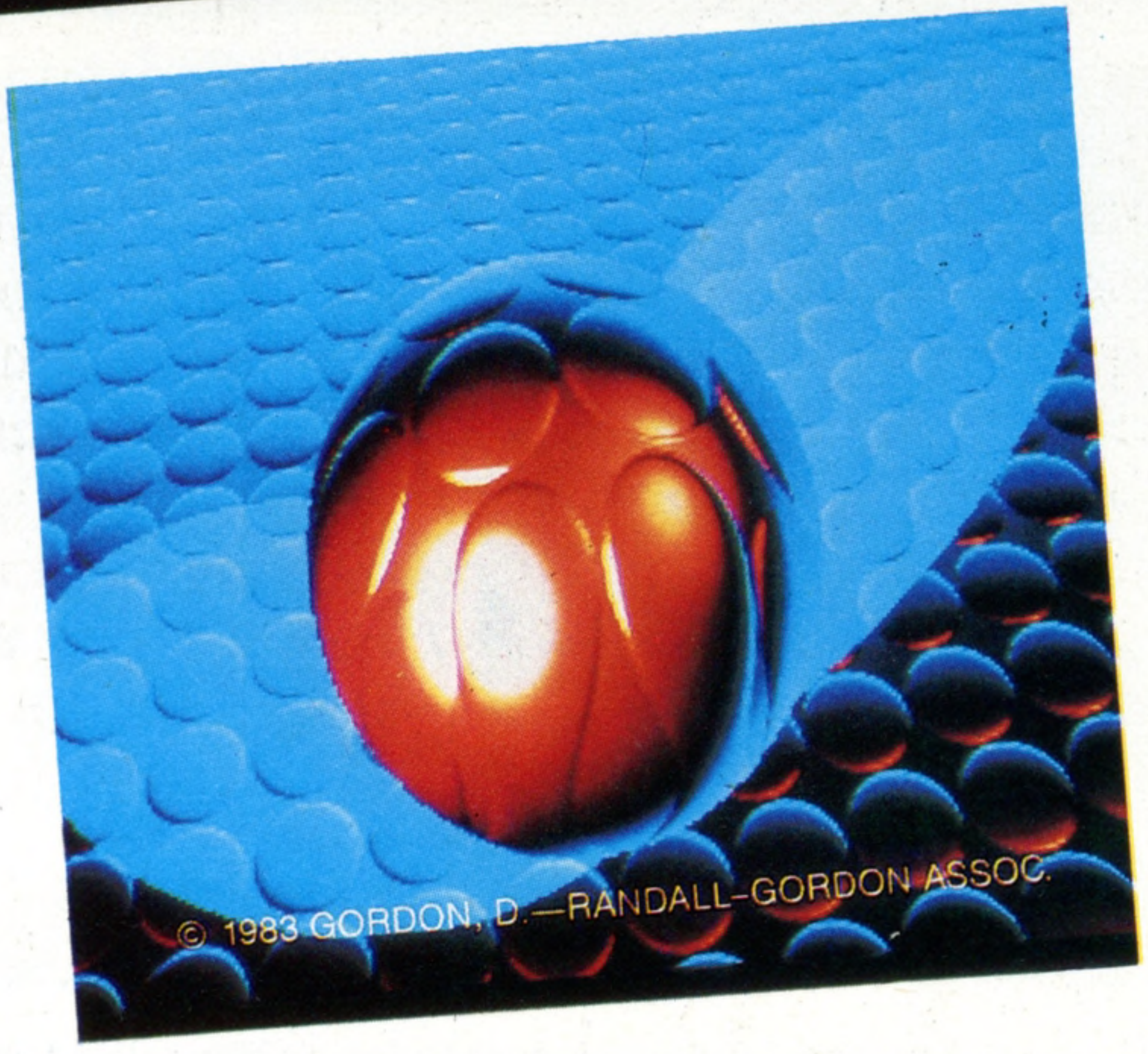
Frame By Frame
Conventional animation, like these frames from 'The Pink Panther', requires the artist to draw each picture separately — though common features need not be redrawn unless they change their appearance or position. Transparent film is used so that the entire image can be made up from a series of overlays. The artist will concentrate his attention on the key frames of the sequence, leaving the intervening sections to be filled in by assistants known as 'in-betweeners'. The finished drawings are then photographed using a rostrum camera, in the order that they will be seen

and Z coordinates. At this point it is extremely useful if the program does not reproduce 'hidden' lines, and by this means introduces opacity into what has been, up to now, a 'wire frame' model or representation.

The next desirable refinement is for 'curve smoothing'. A curved line is specified by only three points — both ends, and the point farthest away from the straight line between them. Of course, a complex curve (an 'S', for example) needs to be split up into its simple components in order for this procedure to work adequately, and it's important to have some simple way of indicating to the machine that the line in question is a curve that requires smoothing, and not just a straight angled line.

Next comes the ability to introduce light and shade into the drawing. First of all it is necessary to specify the position of the light source. The part of the drawn object that lies facing the source will then be highlighted, and progressive shading added to help define the object's shape. Sophisticated software will allow the use of more than one light source and cope with the reflection of light off one object onto another.





© 1983 LISTER, D.—OHIO STATE UNIV.

Magic Colouring Box
Until recently, the method of originating artwork for film and television stills and animated sequences followed closely that used for magazines — the design was executed on paper or transparent film, and then photographed. Quantel's Paint Box system, however, cuts out the use of paper completely, composing the artwork digitally within the computer and then recording it directly onto videotape

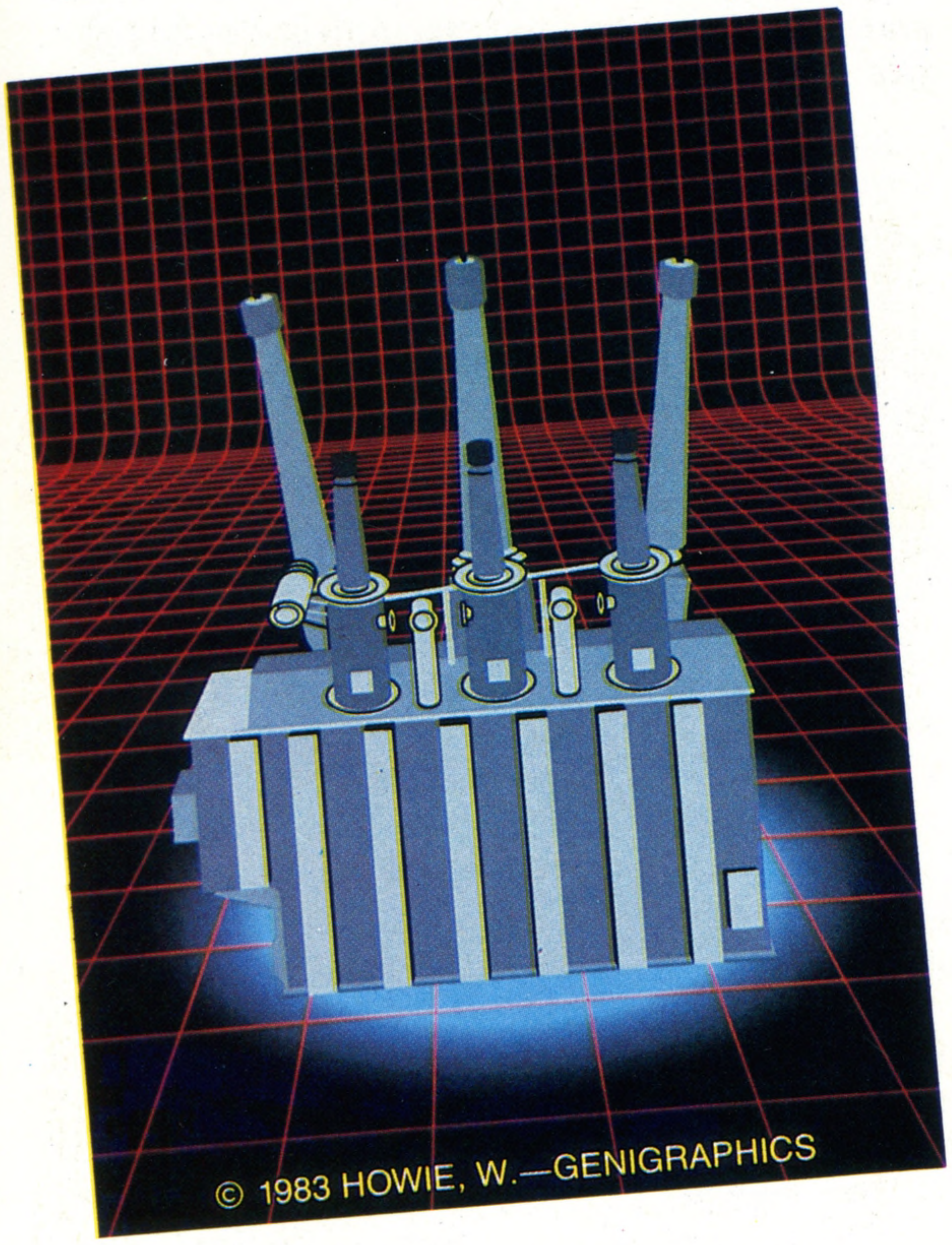
Along with shading goes the use of colour. Even the simplest of home computers now offers eight or perhaps 16 colours, but professional quality graphics computers generally allow at least 4,096. Some are limited simply by the number of binary digits in the computer's 'word'. If this is 24 bits for example, the computer has some 16.7 million colour options. The shading and colouring facilities are combined into one.

Let's now look at the problem of simulating movement. It is relatively easy to reduce movement to its individual components if we think of it as a problem in continuous solid geometry, even when the object represented is as complex as a human hand. The determining factor is the size and power of the computer that is being used. Bear in mind, however, that in order to produce a high-quality image, we will require a monitor capable of resolving something like 1,000 × 1,200 pixels. Each one of these pixels will require at least one eight-bit byte to hold the information that defines its colour and brightness. That means more than a megabyte per screen. Generation of high-quality moving pictures is therefore not possible on home computers. Indeed, professional animators use some of the largest and most powerful computers in the world, and their fees reflect this, being upwards of £1,000 per second of final film.

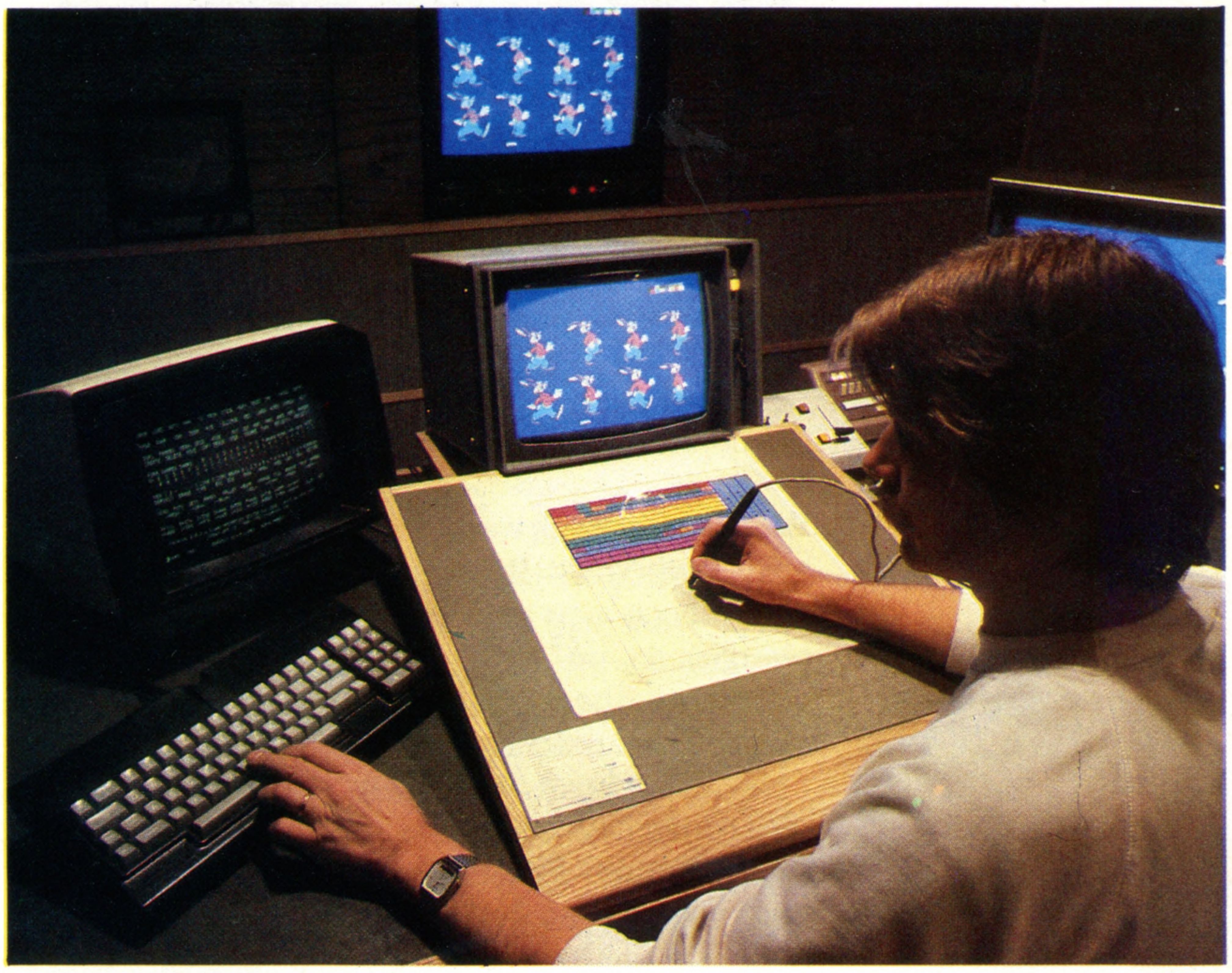
If we take as our starting point a simple object, like a cube, it is relatively easy to understand how we can cause it to move around the screen, tumbling, perhaps, as it goes. A cube can be defined by the coordinates of the eight corners alone, but exactly the same principle applies to more complex objects. The only difference is the amount of memory required to store all the coordinates and the processor power needed to be able to manipulate that information fast enough to generate 'real time' movement. In this application, like all others, there is the inevitable trade-off between quality and the amount of available space and power. The smaller the drawing unit one defines, the greater will be the storage requirements. But it is essential to work in the smallest possible detail to achieve a high quality of reproduction.

A domestic television set — with around 625 × 1,000 pixels — is marginally more 'coarse-grained' than the high power monitor we spoke of earlier. So we can be confident that any work in this detail or better will look as realistic as an ordinary television picture. Even with available techniques we can create an accurate impression of reality by means of animated pictures.

In order to create the image in this sort of detail both sophisticated software and purpose-built or specially adapted hardware are needed. The most popular method uses a device known as a 'bit-pad digitiser', which is rather like a large drawing board that contains a wire matrix. This mesh is used to sense the position of a stylus passed across it. The computer then displays the resulting line or point on the monitor screen. It is possible to trace from existing artwork, draw freehand, or use conventional drawing instruments, just as if one were working on paper. The image is digitised (its X-Y coordinates are worked out), written in memory and displayed by the computer. The character of the mark that appears on the screen can be defined by the user, just as one might choose to use a pencil, a pen or a brush. Likewise, the colour can be defined by calling up the palette — an array of colours at the bottom of the screen looking much like an oil-painter's palette. If the colour one wishes to use is not standard, it can be mixed, exactly as one would when using paint. The stylus can also be used as an eraser, and 'drawings' can be laid over one another.

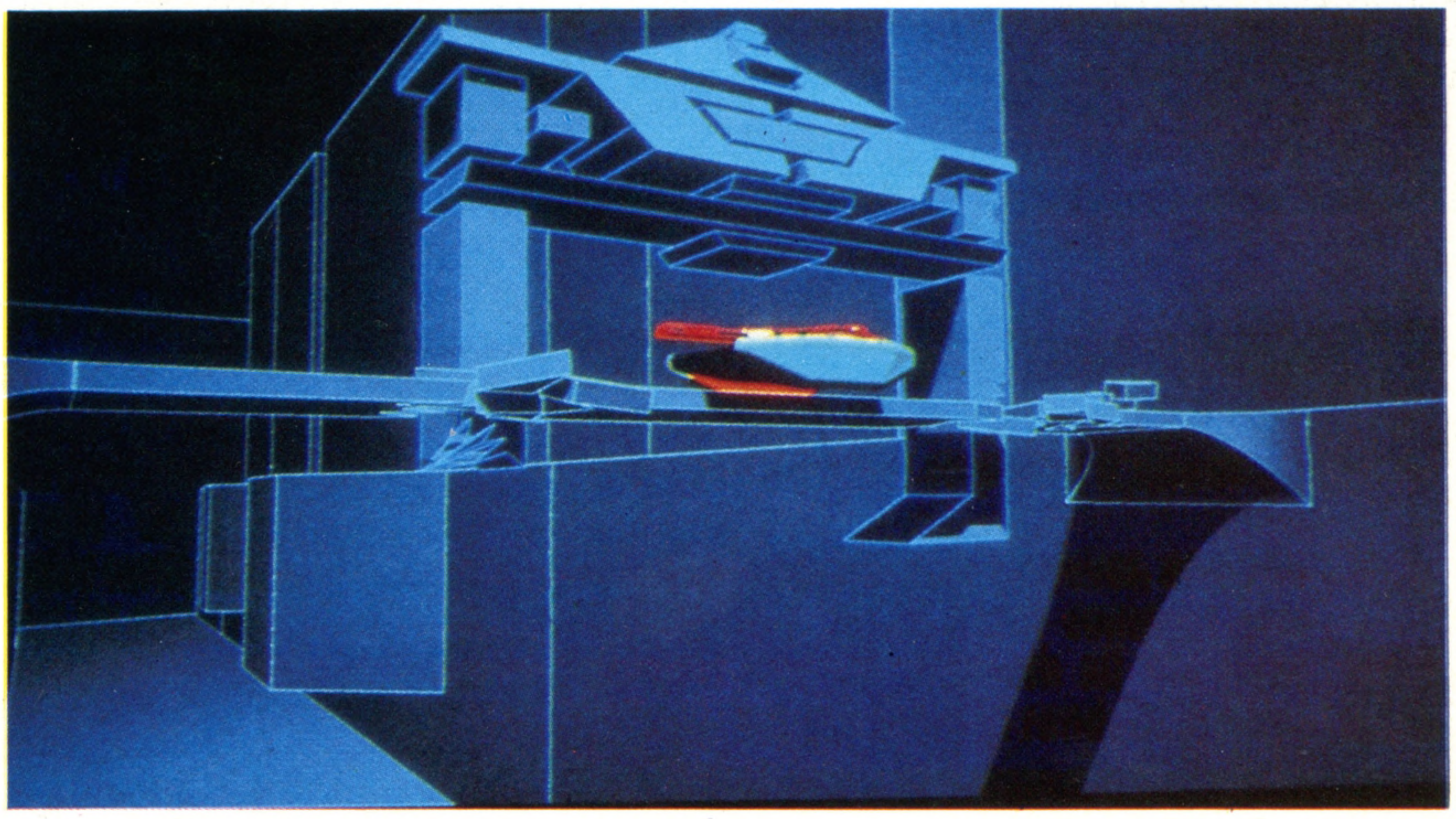


So, having created a single image, how does one go about making it move? One method is simply to mechanise the conventional process, by using the computer system to file material, to colour frames and perhaps to show roughly made-up sequences. Even this approach will speed up the task, but clever programming techniques make it possible to do much more. Just as curves can be automatically smoothed, so can whole blocks of action be fabricated by specifying the first and last frames of a sequence. This process, known as 'tweening', is performed in a conventional animation studio by an assistant known as an 'in-betweener'. Indeed, most of the work of animation is performed by assistants, and it is these that the computer system replaces. We noted earlier that the introduction of computers into the animation process releases the artist to concentrate on the quality of the image. Most of the hand animator's effort goes into creating the illusion of movement, but this task is precisely definable on a computer. Once the rules are stated, simply obeying them will produce the desired result — once again, a sure sign that the job is appropriate for computerisation.


IAN DOBBIE AT RESEARCH RECORDINGS

Seeing Is Believing
Using the ultra-fast processing power and huge storage capacity of modern computers it is possible to create on film or on the TV screen an image that is actually indistinguishable from a photograph. Then, using programming techniques developed for statistical and numerical problem-solving, it is possible to manipulate these created images in such a way as to make the viewer believe them to be real

TRace ON
First of a new generation of feature films that use these techniques of computer-assisted image generation was Walt Disney Productions' 'Tron'. Set partly in reality and partly inside a giant computer, 'Tron' uses a mixture of computer-animated imagery and special effects photography to create a stunning realisation of a fantasy world


COURTESY OF WALT DISNEY LTD.

Alternative Translation

Computers 'think' in machine code; programmers prefer to write in a high level language such as Basic. Compilers and interpreters offer different methods of translation between them

When computers were first developed they didn't have keyboards. Program instructions had to be entered one step at a time by setting each of eight switches to 'up' or 'down', to represent a single operation. These patterns of 'up' and 'down' were examples of machine code.

It was logical to replace the switches by a typewriter keyboard, and replace the patterns of switch settings by real English words. The result was the 'high-level' language such as BASIC, replacing the low-level machine codes.

As processors, however, computers did not change, but continued to work on the original patterns of switches (and still do), so programmers had to develop programs written in the original low-level notation to translate these high-level programs into patterns that the processors could work on. These low-level programs came to be called interpreters or compilers, according to their method of translation.

In computing (as elsewhere), any gain in power or speed has to be paid for — in money, time or freedom of action. So it is with interpreters and compilers. Together they provide all the program translation facilities that a programmer needs. Interpreters are strong in some areas and compilers in others, but each pays for its advantages with compensating disadvantages.

Interpreters, usually built into the home computer, are the cheap way of translating high-level language programs into something a computer can understand. They don't use up much memory — leaving more space for your programs.

Micros costing less than about £400 almost invariably feature a BASIC interpreter: you type in a BASIC program, type RUN, and either the program works, or it stops with an error message from the system — something like:

```
SYNTAX ERROR ON LINE 123
```

So you type LIST, find the error, correct it, type RUN, and it either works or stops again, and so on. Note that some of the more sophisticated BASIC interpreters actually check for syntax errors as each line is entered.

You may have done this sort of thing hundreds of times without having given a thought to the interpreter. Its chief virtue is precisely that it is an invisible device that allows you to work on your program without ever bothering about where it is in memory or how to execute it — the program is at your fingertips, and you can RUN it, LIST it, or EDIT it immediately.

The interpreter is easy to use, but not very sophisticated: every time you type RUN, the interpreter has to find your BASIC program in memory and translate and execute it line by line. If your program contains this loop:

```
400 LET N=0
500 PRINT N
600 LET N=N+1
700 IF N<100 THEN GOTO 500
```

the interpreter has to translate and execute lines 500 to 700 a hundred times, as if it had never encountered them before.

Compilers are different. They're expensive, difficult to write, and occupy and use a lot of memory. They are almost always disk-based software, so the user needs an expensive system.

What they offer is flexibility, power and speed; faced with the four lines of BASIC above, a compiler would translate them all once, then execute that code a hundred times.

This allows quite a saving in time – but at a price. Suppose you have a BASIC compiler and you want to enter and run a BASIC program.

First you load and run the File Creation Program (called the Editor), which allows you to type in the program and save it to disk as a 'source file'.

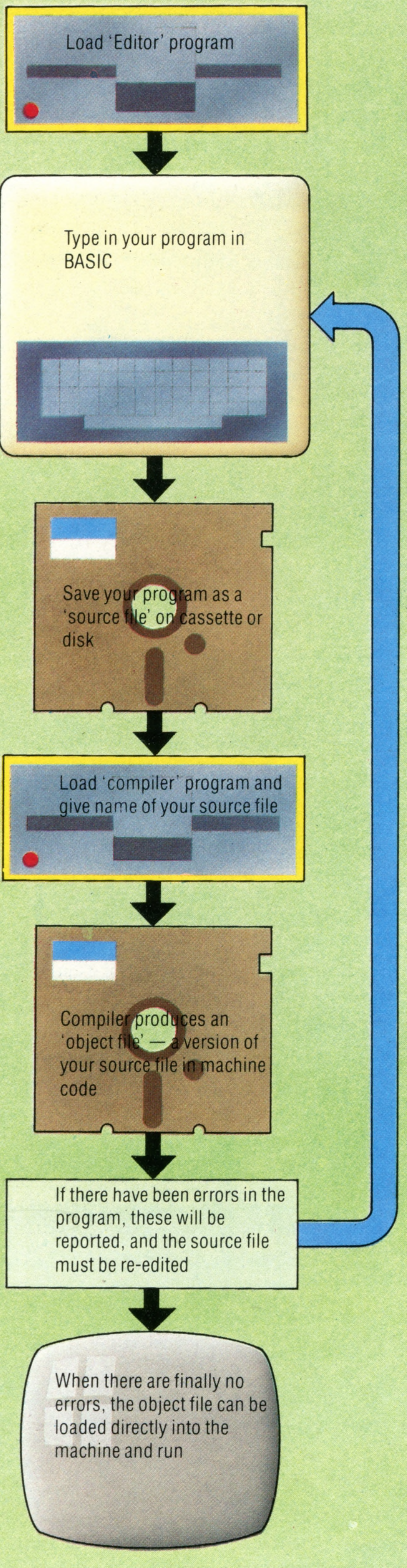
Files must be named so that you can find them once you've created them (just like files in a real filing cabinet), so the Editor asks you to name the source file. File names often consist of two parts: the first is a label, any name you choose – say MYPROG – and the second part is usually a three-letter code indicating the nature of the file contents; this code is the 'extension'. A BASIC file might have the code BAS as its extension. Your source file is now on disk under the name MYPROG.BAS. Now, typing:

```
COMPILE MYPROG.BAS
```

will cause the computer to LOAD and RUN the BASIC compiler on a BASIC source file called MYPROG.BAS.

You wait a few seconds, depending on the length of your program while the compiler translates your program into an 'object file', which it saves on disk under the name MYPROG.OBJ – the OBJ extension indicating that this is the object file, a machine code translation of a source file.

Compiling A Program
Writing a compiled program is not nearly as simple as using the interpreter on your home computer. However, once you have the program working, it will execute many times faster. This is the main flow of events:


Load 'Editor' program
Type in your program in BASIC
Save your program as a 'source file' on cassette or disk
Load 'compiler' program and give name of your source file
Compiler produces an 'object file' — a version of your source file in machine code
If there have been errors in the program, these will be reported, and the source file must be re-edited
When there are finally no errors, the object file can be loaded directly into the machine and run

While the compiler translates your file, it checks it for syntax errors. If it finds any, then you'll get a message like this:

```
100 REED X:IF X=3(N+2) LET P=Q
     1                2    3

FATAL ERROR:-
1) //REED// UNRECOGNISED COMMAND
2) //(// ILLEGAL OPERATOR HERE
3) ??'THEN' OR 'GOTO' EXPECTED HERE
```

You get this kind of message for every line that contains an error. In other words, the error reporting is far more comprehensive than on a BASIC interpreter. Now you must load and run the Editor again, recall the source file from disk, make the changes and try to compile again. If there are no more errors you can type:

```
RUN MYPROG
```

and it either works as you expect or it doesn't. There are no syntax errors at this stage, because you've corrected them, but you might still want to change the program anyway, in which case you load and run the Editor, change the source file, recompile it...and so on.

The virtues of a compiler are not obvious in the program development stage, though informative error reporting is valuable. Compilers start to earn their keep after you've got a working program and typed RUN, which is precisely where interpreters start to let you down.

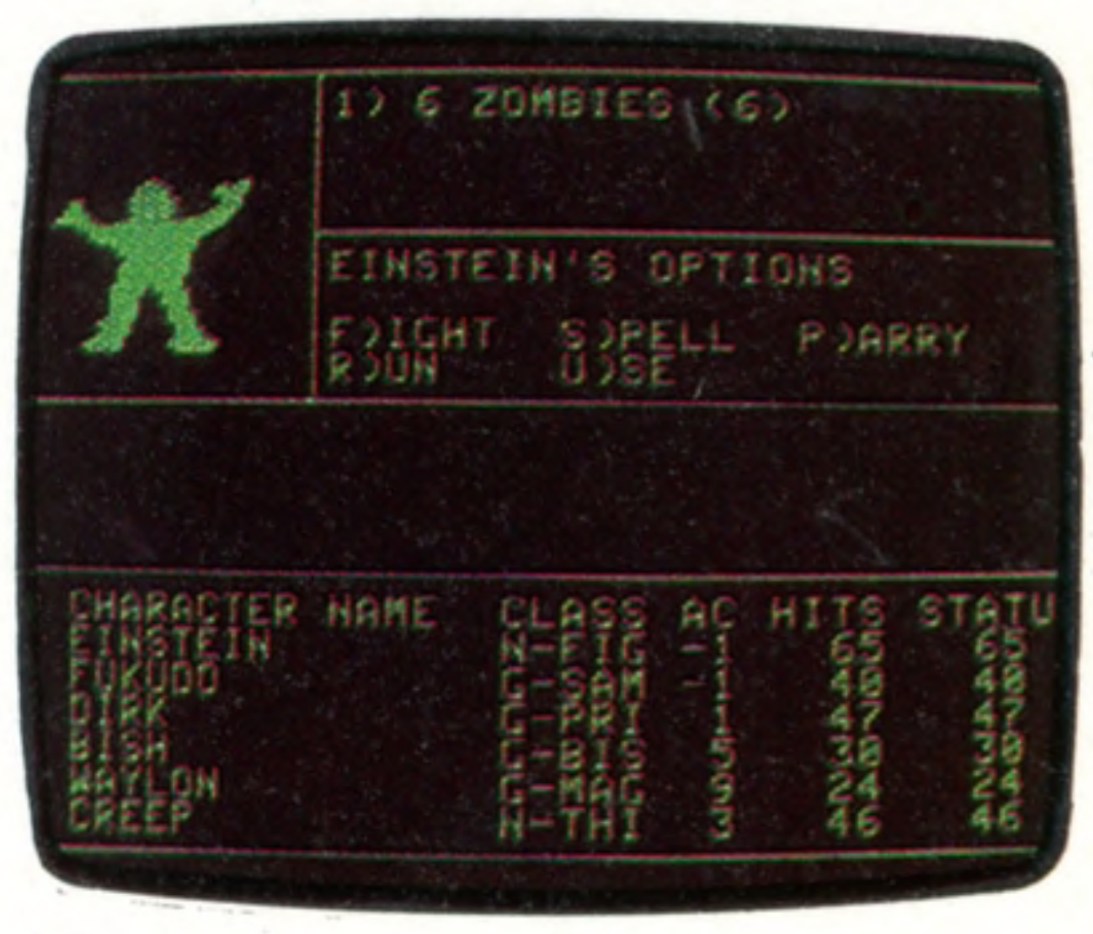
Compiled programs are fast — anything from five to 50 times faster than interpreted programs, depending on the efficiency of the compiler, but the compiled program's speed of execution is bought at the expense of its speed of program development.

Comparing compilers and interpreters by contrasting typical sequences of user commands like those above is unfair on compilers, since they are written mostly for more powerful, less specialised machines, the users of which might want to write and run programs in many different programming languages.

COBOL (for writing commercial data processing programs to handle accounts, payroll and inventory), for example, was invented with compilation in mind, whereas BASIC really demands an interpreter. If you're going to compare a Jensen with a Jeep, you ought to do so on both ploughed fields and metalled roads.

Once you've developed and compiled a program, you don't need the source file except for reference. So the source program can be fully commented on and written with readability in mind, while the object file may be a much smaller file, occupying less space on disk and memory.

The fact that the object files created by a compiler consist of unreadable machine code, can, surprisingly, be an advantage. If you're marketing software you don't sell the source file but only the object file, which makes it much harder to pirate, copy or alter.

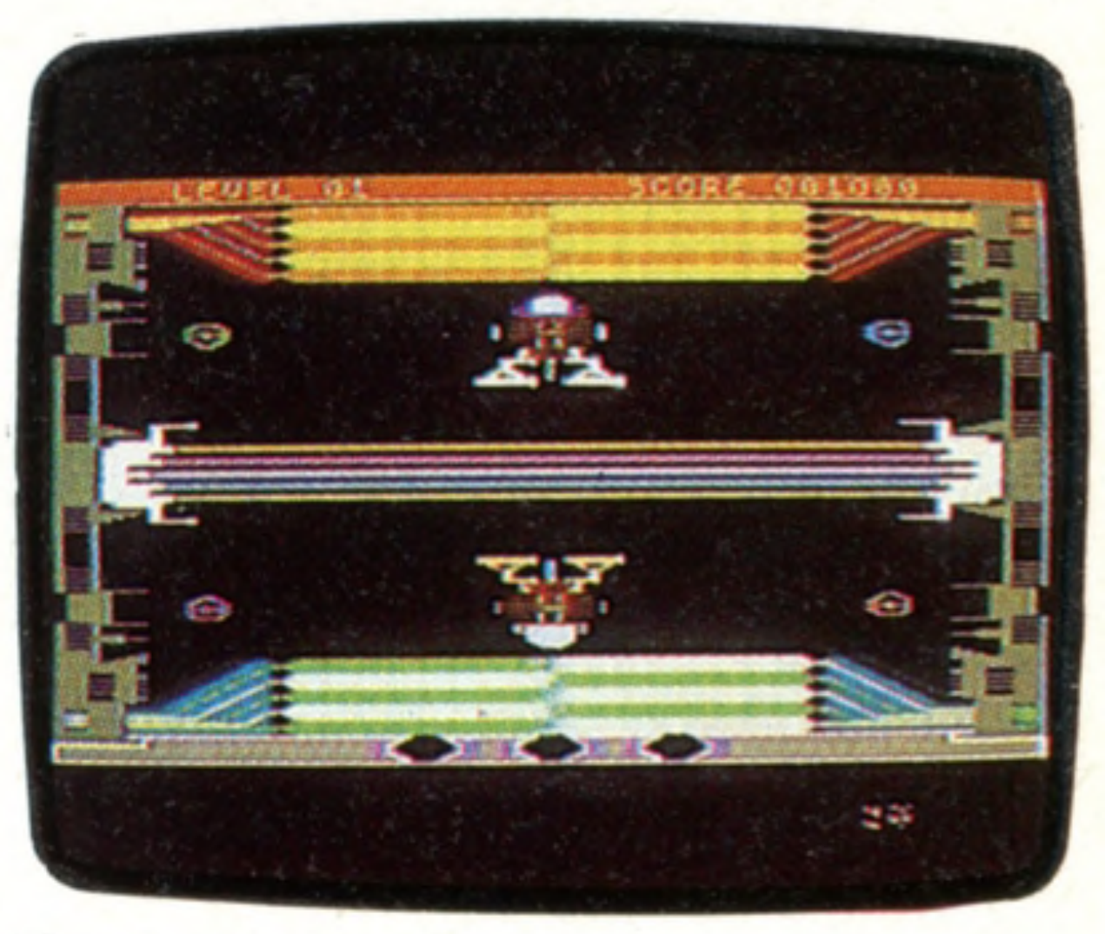


```
1) 6 ZOMBIES (6)

EINSTEIN'S OPTIONS

F)IGHT  S)PELL  P)ARRY
R)UN    U)SE

CHARACTER NAME   CLASS AC HITS STATU
EINSTEIN         N-FIG -1  65   65
FUKUDO           G-SAM -1  48   48
DIRK             G-PRI  1  47   47
BISH             G-BIS  5  38   38
WAYLON           G-MAG  9  24   24
CREEP            N-THI  3  46   46
```

Slow
In an Adventure-style game speed is not critical, and most of the program consists of manipulating strings of text. Therefore it is written in BASIC and interpreted as it runs on the computer

Worksheet Range Copy Move File Print Graph Data Quit
Global, Insert, Delete, Column-Width, Erase, Titles, Window, Status

Faster
Many business programs (particularly spreadsheets) are difficult to write in machine code because they involve a lot of mathematics. However, an interpreted language would be too slow, so they are often written in BASIC and then compiled



Fastest
For fast action arcade-style games, which involve the manipulation of graphics, even a compiled program would not be nearly fast enough. Such packages have to be written directly in machine code — a slow and painful task

Talking Back

It used to be pure science fiction. Now with a speech synthesiser your computer really can talk to you. And it needn't sound like a Dalek, either

While the science of speech recognition has yet to be fully developed, the generation of electronic speech has been mastered. Until recently, however, the computing power and memory capacity needed to produce human-like utterances were substantial. Now, with the aid of a suitable add-on, almost every home computer and electronic toy is capable of talking back. The rapid advances in technology and the fall in the cost of computer components have made the talking computer commonplace.

When people talk, sounds of three distinctly different types are produced. The first is 'voiced' or vowel-type sound — *oo, ar, ee* and so on. These are produced by the vibration of the vocal chords in the throat, the frequency of this vibration determining the vowel sound. The second is the 'fricative' or unvoiced sound, such as *ss, sh, t* and *ff*. Here air from the lungs bypasses the vocal cords and the frequency of the sound is controlled by the positioning of the lips and tongue. The third 'sound' is silence or — to be more precise — the gaps occurring within words like six, eight and so on. You may not realise that there are gaps in these words, but if you try to pronounce them slowly you will realise that it is impossible to run smoothly from the sound of *i* into *x*.

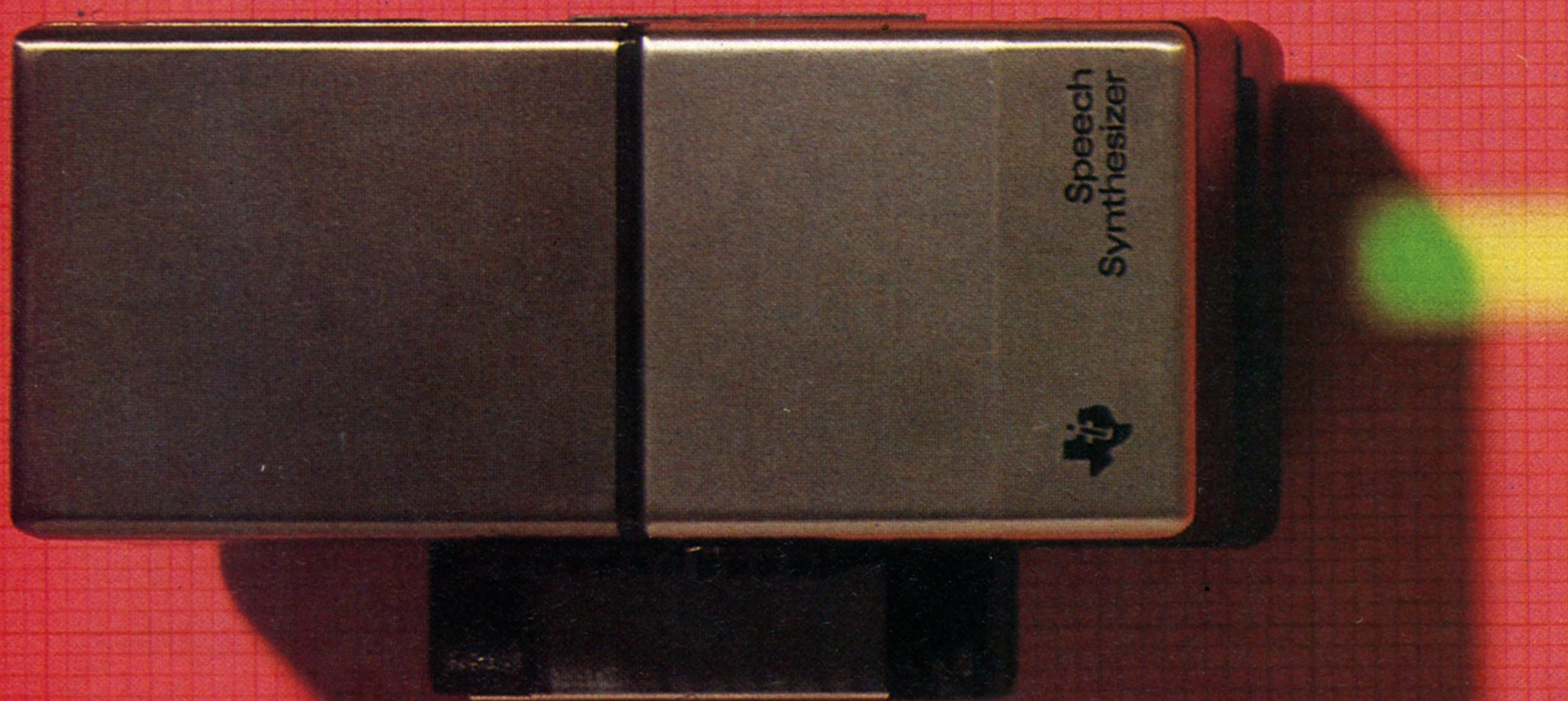
Building Blocks Of Sound

There are two ways of generating speech-like sounds electronically. The first, until recently the most common, is that of synthesis by rule. By analysing the frequencies contained within speech it is possible to devise a system of rules that allows us to create any given sound from its components. For example, the word 'too' could be defined as so many milliseconds of the mixture of frequencies that make up the sound *t*, followed immediately by the *oo* frequencies.

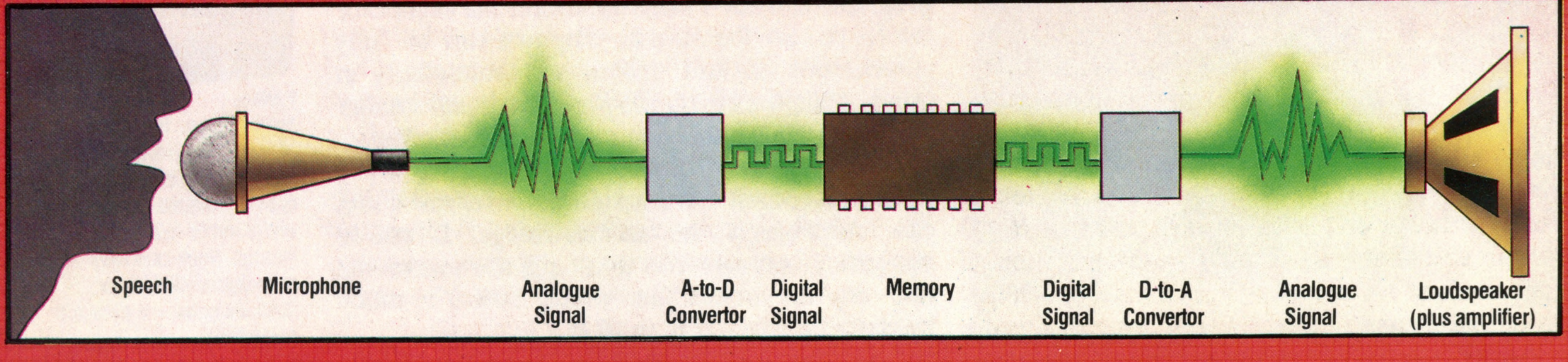
These individual building blocks are called 'phonemes' and by using them in various combinations any word can be constructed. The individual characteristics of a human speaker tend to be lost when speech is generated in this way, but the words can be recognised and understood. Because the rules for generating the phonemes are built into the equipment itself, the user is able to enter a list of the phonemes into the system. These are reproduced through a small speaker. With a little practice it is possible to generate complete sentences instantly by calling up sequences of phonemes, which can usually be stored in BASIC strings.

The second method of speech synthesis relies

IAN McKINNELL



The Flow Of Sound
Speech can be digitised and stored in memory, either RAM or ROM. Electrical output from a microphone is passed through an analogue-to-digital convertor. The output from this chip is a digital pattern of 1s and 0s. The speech can be recreated using a digital-to-analogue convertor, an amplifier, and a loudspeaker



KEVIN JONES

on the human ear and brain to fill in gaps. For example, the range of frequencies that can be transmitted over a telephone line gives only one-fifth the quality we would expect from a reasonable hi-fi system, yet the speech we hear through the ear-piece is perfectly understandable. This is because our brain fills in the gaps.

The second method of synthesis, called 'digitised speech', uses the same phenomenon. With the reduction in cost of computer memory it is now possible to convert speech into digital information by means of an analogue-to-digital converter. The resulting data is then compressed many hundreds of times and stored in a ROM — thereby creating the gaps which your ear can compensate for.

To cause any of the stored words to be spoken we simply give the computer the address of that word in the ROM, and the digital information is recovered and converted back into sound. Because the original speaker's words are stored, the personal characteristics remain. Acorn's speech chips for the BBC Micro, for example, can be clearly identified as the voice of newscaster Kenneth Kendall.

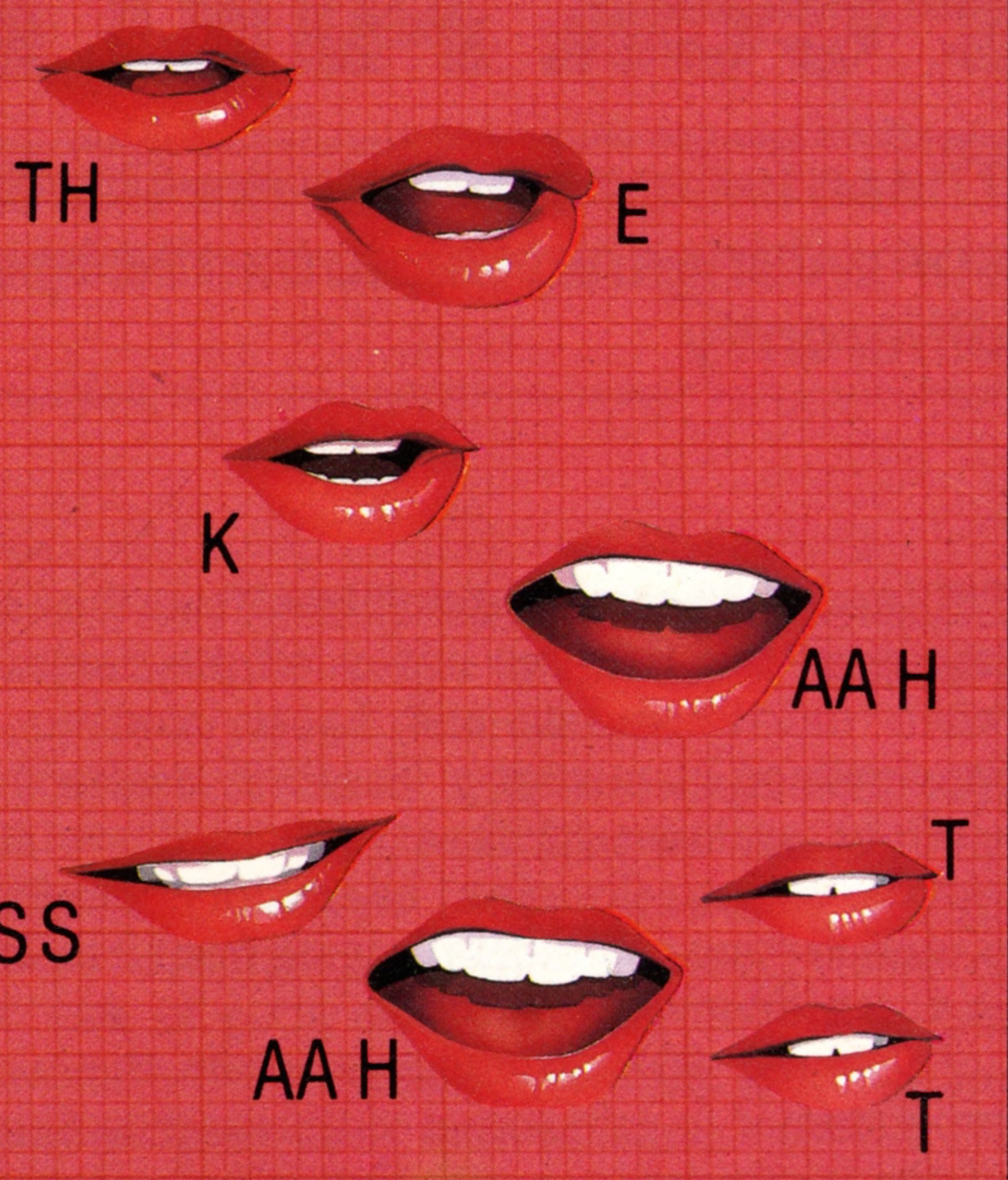
Some computers, notably the Sirius 1, feature built in hardware and disk-based software to allow the user to digitise his own voice using a microphone. The resulting data is stored on disk — one second of speech occupies about one Kbyte — to be recalled from an applications program as verbal messages and warnings.

The uses for speech synthesisers are so many and varied that it is almost impossible to list them. To start with, speech synthesis can replace taped announcements at railway stations, airports and other terminals. In the USA it is widely used on the telephone system to inform callers of wrongly dialled numbers, engaged numbers or withdrawn services. Many automated ordering systems now feature speech response. An order number is keyed in to a computer, which speaks the description as a double-check. The computer can also inform the customer of the current stock level or the likely waiting period so that the order may be modified at the time it is placed.

Speech synthesis units are now incorporated into cars — the BL Maestro, for example — as part of the standard instrumentation. More than a mere sales ploy, the synthesiser provides warnings that the driver can hear and act on without having to take his eyes off the road.

In the home computer and electronic games market speech synthesis is used to enhance games: scores are called out and warnings of enemy attack can be given verbally, leaving the player free to concentrate on the tactics of the game rather than having to consult messages printed at the bottom of the screen.

Finally, there are educational devices such as Texas Instruments' Speak'n'Spell, which recites a word that must then be spelt correctly, and foreign language dictionaries that speak the words as they are displayed.



Figures Of Speech

If we take a short and familiar English sentence

'The cat sat on the mat'

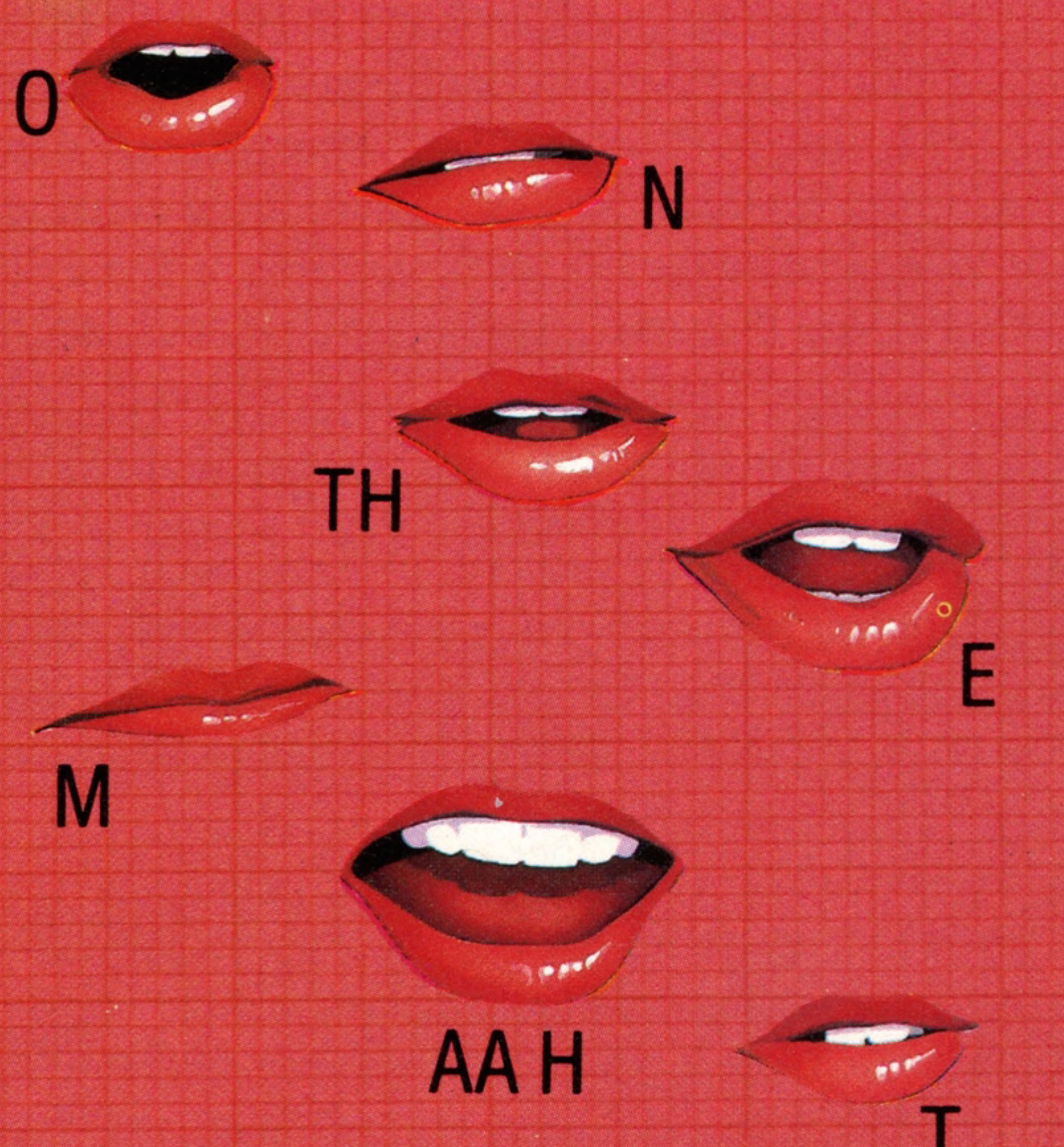
it is possible to break it down into a series of phonemes, as follows:

TH EE / K AA T / S AA T / O HN / TH EE / M AA HT

Different chips require that phrases are broken down and specified in different ways. For the same sentence, another chip might require the following phonemes:

THV E / K A T / S A T / UH_3 N / THV E / MAT

The Votrax chip, for example, contains some 60 phonemes and rules for using them, which can be directly accessed by a simple number. To make the system more usable, a set of programs is provided that allows the user to type in the utterance required in the form of the phonemes it contains, as in the example above. However, with devices such as the Braid Speech Synthesiser or Votrax's own Personal Speech System, which incudes a dedicated microprocessor and some sophisticated software, the user can type in the plain English text and get the spoken equivalent back



TONY LODGE

Peek And Poke

These two commands are used whenever you want to program something that Basic can't cope with, but every machine uses them differently

PEEK and POKE are two 'statements' from the BASIC language used in more advanced programming when individual bits and bytes need to be manipulated in memory. The PEEK statement is used to examine (peek at) the contents of a specific address (location) in memory, and POKE is used to store a number (ranging from 0 to 255) in a specific memory location.

PEEK and POKE statements allow the BASIC programmer to gain access to the inner workings of the computer in a way that is not otherwise possible. Normally, the built-in BASIC in your computer takes care of the actual locations where such things as variables and the data defining the characters to be displayed on the screen are stored. Although we do not usually worry about where such things are in the memory, occasionally we need to find out. The PEEK statement allows us to do this.

A short program to examine any memory location can easily be written:

```
10 REM LOOKING AT MEMORY LOCATIONS
20 PRINT “ENTER MEMORY LOCATION IN
   DECIMAL”
30 INPUT M
40 P = PEEK(M)
50 PRINT “CONTENTS OF LOCATION ”;M;“ ARE ”;P
60 GOTO 20
70 END
```

This will print the contents of the specified address expressed as a decimal number. (In fact, of course, the computer stores it in binary.) If you would like to see what the contents are equivalent to in terms of 'printable' characters, BASIC includes a function to convert decimal numbers into their character equivalents. This is the CHR$ function and changing line 50 will print character equivalents of the memory locations instead:

```
50 PRINT “CONTENTS OF LOCATION ”;M;“ ARE ”;
   CHR$(P)
```

To examine the whole of memory, a FOR...NEXT loop can be added by deleting line 30, changing line 20 to FOR X = 0 TO 65535 and replacing line 60 with NEXT X.

To give enough time to see each character as it is printed, you may need to add a delay loop after the PRINT statement and before the NEXT X statement. Note also that the upper limit of the FOR...NEXT loop assumes you have a 64 Kbyte memory. This number can be changed for smaller memories: 16 Kbytes requires 16383 in decimal, 32 Kbytes requires 32767, and 48 Kbytes requires 49151. A full listing of this program is:

```
10 REM PEEKING AND PRINTING ALL MEMORY
   LOCATIONS
20 FOR X=0 TO 65535
30 LET Y=PEEK(X)
40 PRINT “LOCATION ”;X;“ = ”;Y;“ = ”;
50 PRINT CHR$(Y)
60 FOR D=1 TO 200
70 NEXT D
80 NEXT X
90 END
```

Although the CHR$ function converts decimal numbers into their character equivalents, printable characters are represented by the numbers 32 to 127. Most computers use the numbers between 128 and 255 (the largest number representable in a single byte) for special graphics characters. Many of the numbers between 0 and 31 have special screen control functions. When these are encountered in memory as the program is run, they will be converted by CHR$ into curious screen effects. These may make the screen go blank, for example, or cause the cursor to move to the top left-hand corner of the screen.

The POKE statement is essentially the opposite of PEEK. It allows you to 'write' a byte of data (any number between 0 and 255) into any memory location. POKE must be used with care: if you POKE a number into the wrong part of memory you could 'crash' the computer by corrupting part of an essential program. The only way to recover from this is to reset the computer (switching it off and then on, unless it has a reset button), and this risks destroying one of your programs. Before using POKE, therefore, check the manual to find an area in the memory map designated a 'user area'.

Most home computers make the video memory (the memory used for storing the characters to be displayed on the screen) available to the user. Normally, the computer gets the shape of the characters to be displayed from a special ROM called a character generator, which stores the patterns of dots for each character. But it is usually also possible to use RAM as well. When the pattern codes for characters are stored in RAM, new patterns, specified as decimal numbers, can be POKEd to the appropriate RAM location and used to define completely new displayable characters.

POKEing

The POKE statement needs to be used with care as it changes the contents of memory locations and this could affect the running of the computer. No damage can be caused by this, but it could mean the loss of a program. Here are a few 'safe' POKE statements for you to try.

On the Atari 400 or 800, POKEing a 1 into location 751 will turn the screen cursor off; try POKE 751,1.

On the Commodore 64, try POKE 1024,1. 1024 is the address of the first screen location.

On the Sinclair Spectrum, try:

```
100 FOR N = 0 TO 6 STEP 2
110 POKE USR“A” +
    N,BIN01010101
120 POKE USR“A” + N +
    1,BIN10101010
130 NEXT N
140 PRINT “AAAAAAAA”
```

The As in line 140 must be typed in the graphics mode. Running the program will produce a line of miniature checkerboard symbols. However, it should also result in some interesting interference patterns on your TV set

TI99/4A

Texas Instruments' home computer is a Mercedes among Volkswagens – it has a high standard of construction, but the add-ons are expensive

In terms of design and construction, Texas Instruments' TI99/4A is one of the most professional of home computers. TI's withdrawal from the home computer market was a blow to hobbyists, but the machine is still being sold, and devotees regard it as still worth the trouble of seeking out.

It uses a 16-bit microprocessor, the TMS9900, designed and made by Texas Instruments, who make semiconductors, calculators, microprocessors and minicomputers. The TMS9900 was one of the first 16-bit chips but it failed to gain widespread popularity.

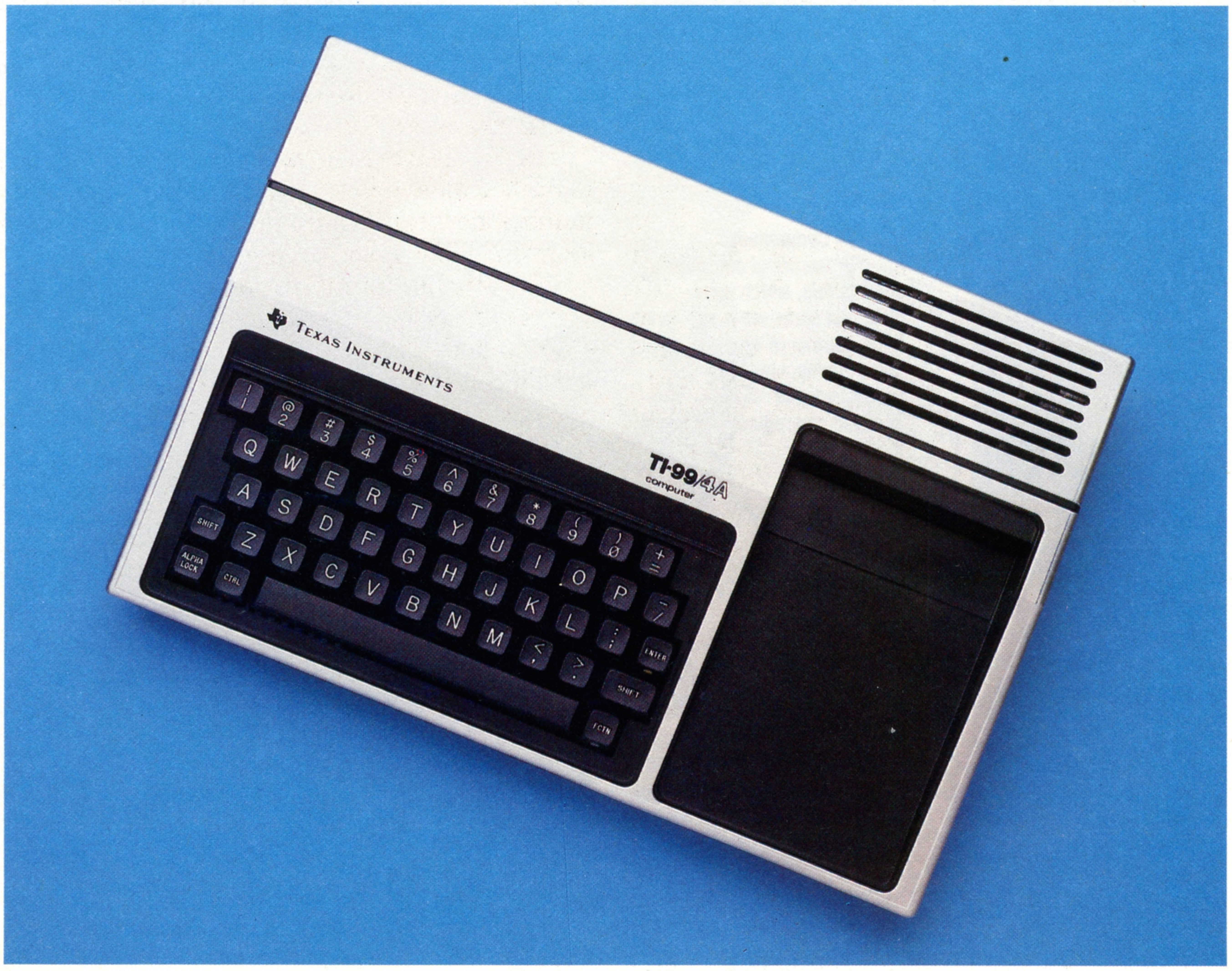
The TI99/4A has a 48-key keyboard, which by the general standards of home computer keyboards is very good to type on. There is a space to the right of it that receives the software cartridges, which Texas refer to as 'solid state software'. A similar connector on the right-hand edge of the case permits hardware expansion. The expansion modules, which are large plastic boxes, contain disk drive controllers, memory expansion and a serial (RS232) interface and are connected via an expansion box, a unit which is essential if you wish to extend the machine.

The screen display is in 16 colours with high-resolution graphics, and there is also a sound generator capable of producing three independent notes or 'voices' at once. However, the lack of good documentation makes writing machine code programs to use the graphics and sound facilities fairly hard to learn.

Almost every hardware add-on costs £90 or more and there are virtually no peripherals made by suppliers other than Texas.

The computer is designed for new users to computing, BASIC being the resident language and LOGO the most popular add-on language. In America it has been widely used in schools, and once competed with the Apple II for the position of top-selling educational micro.

When it is switched on, a menu is displayed on the screen offering the user a number of choices. If a software cartridge is plugged into the computer



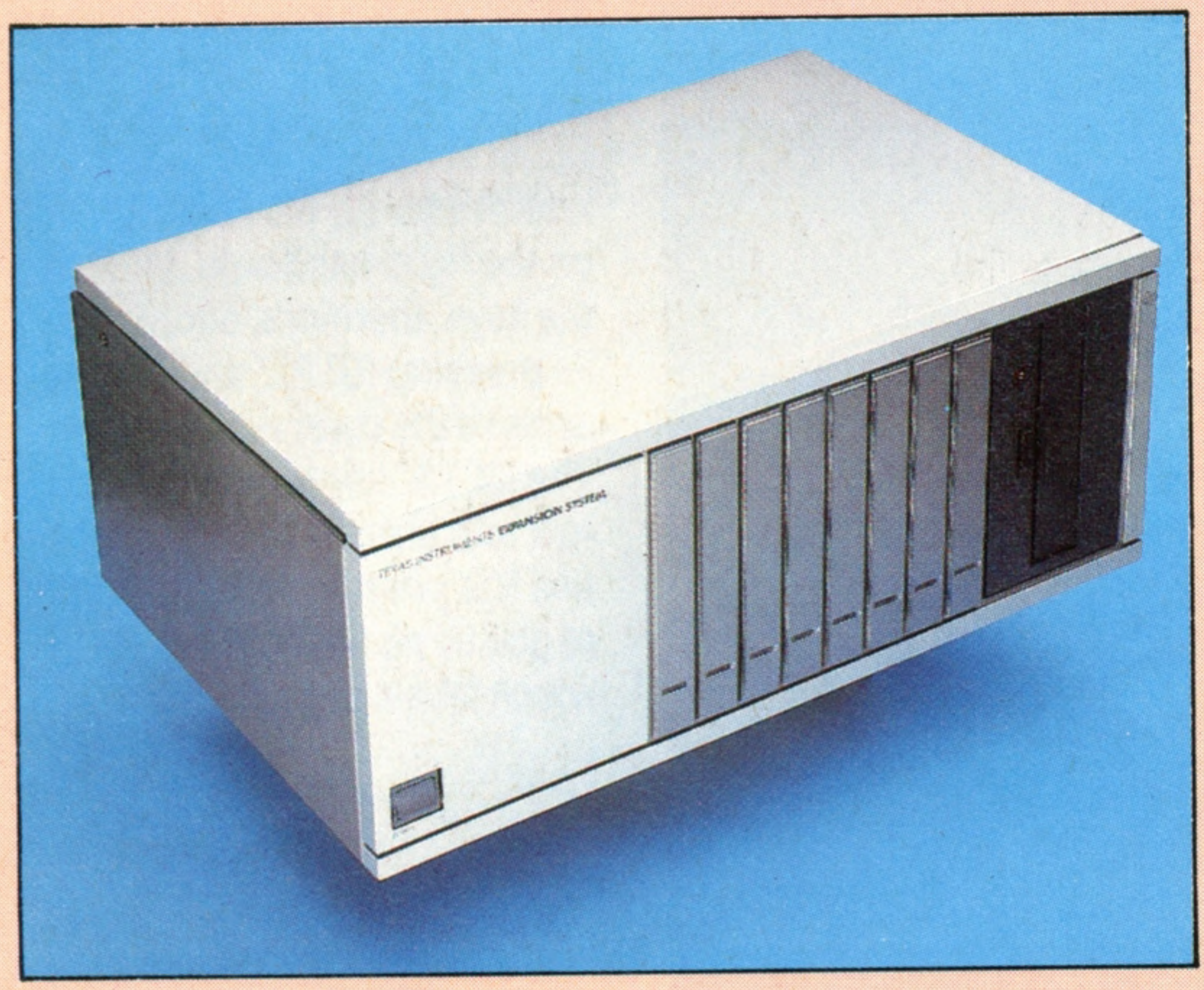
CHRIS STEVENS

TI99/4A Keyboard
The keyboard is of a higher standard than on most home computers, though some users have commented that the 'bounce' on each key is too stiff. The number of keys is also rather limited, presumably to make room for the cartridge slot on the right-hand side. Most of the keys, therefore, double up — pressing 'CTRL' and 'E' will achieve the cursor-up function. The 'FCTN' key turns the top row into user-definable keys, and it is possible to insert a strip of plastic above this row, on which labels can be written

the user is presented with the option of running the new software or of running BASIC. The built-in BASIC is limited in its abilities, but an 'extended' BASIC cartridge is available, which brings the facilities up to and beyond Microsoft standard, giving formatted print commands (see page 53), sprite graphics and the ability to operate a speech synthesiser. The synthesiser costs £34.95 but needs either the extended BASIC or the Speech Editor cartridge to operate it.

The TI99/4A has many hardware and software extensions. Every sort of peripheral is available and many programming languages can be purchased. But although the basic computer is cheap, most of the extras are expensive. Nevertheless, the overall system is easy for novices to use, and its robust construction makes it popular with children.

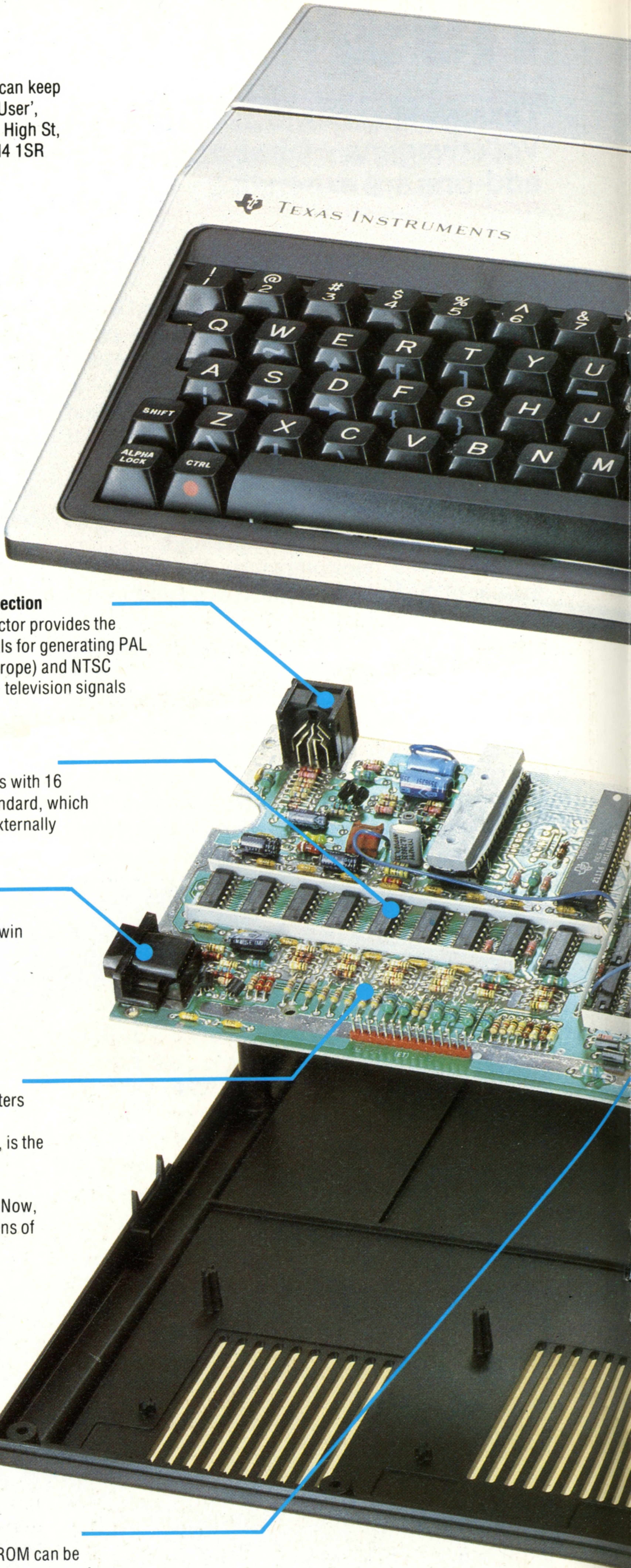
The Joystick
Texas Instruments' joysticks (they call them 'Wired Controllers') come as a pair of units, wired together onto one plug for connection with the computer. Inside each device are four switches, which are not unlike the connectors underneath some keyboards



Peripheral Expansion Box
This case contains a power supply, connections and space to contain all the modules for memory expansion, disks and printer interfaces. These modules are large plastic cases that contain circuit cards with an edge connector at the base, a 'power on' light at the front and any cables coming out of the back. There are eight 'slots' in the box. The left-hand one has to contain the module that connects the expansion box to the computer and the right-hand slot has to be for the disk drive electronics module. This leaves six slots for memory and serial port expansion. Only one extra 32 Kbyte memory module can be added, which gives 52 Kbyte maximum RAM. The serial interface module allows serial devices, such as printers and modems that use the RS232 format, to be connected to the TI99/4A

CHRIS STEVENS

TI99/4A enthusiasts can keep in touch through 'TI User', c/o Galaxy Video, 60 High St, Maidstone, Kent ME14 1SR



Video Connection
This connector provides the basic signals for generating PAL (UK and Europe) and NTSC (American) television signals

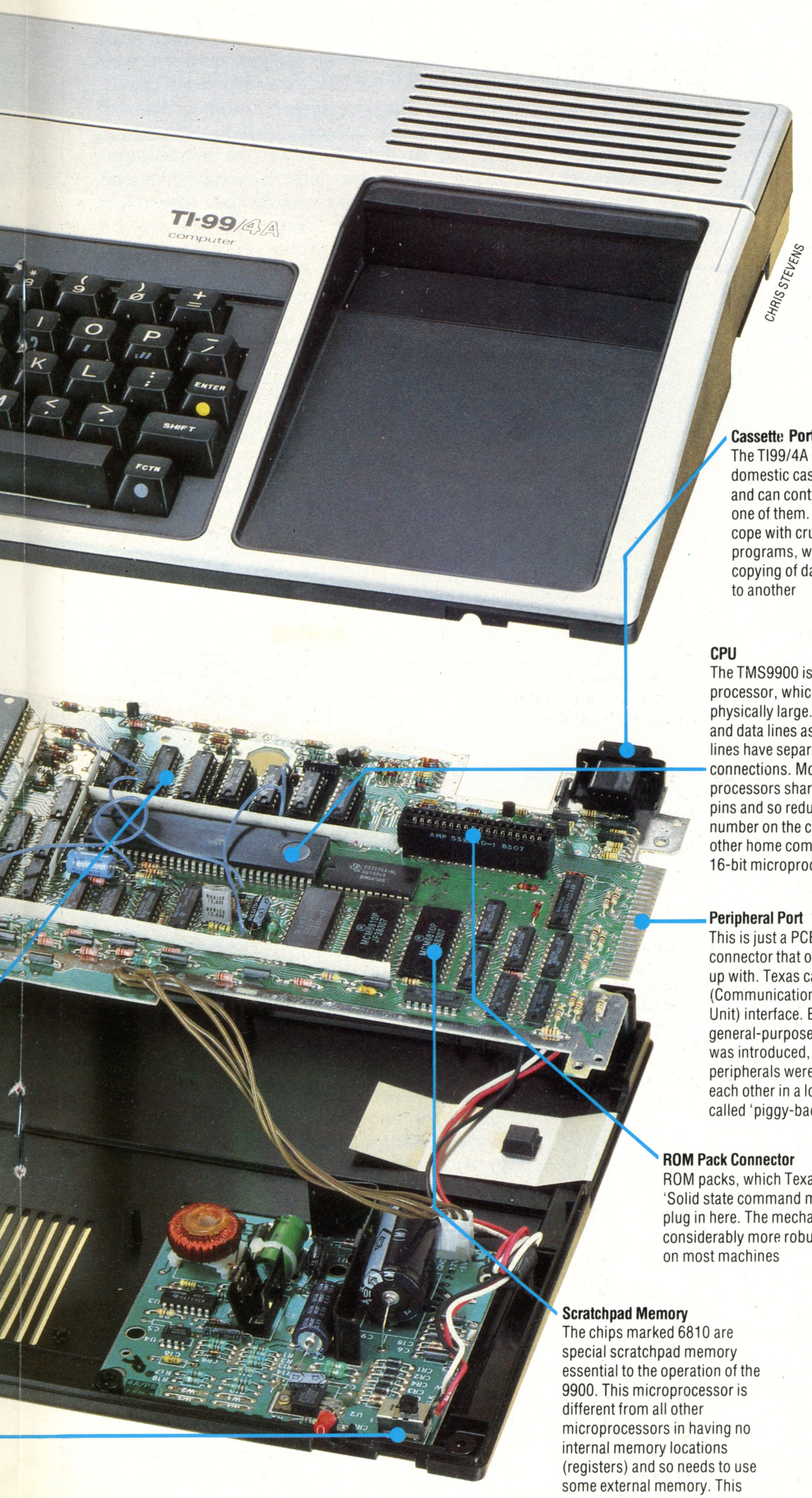
RAM
The machine comes with 16 Kbytes RAM as standard, which can be expanded externally

Joystick Connectors
This single multiple pin connector can cope with twin joysticks made by Texas themselves

Discrete Components
Another feature of computers such as this, which were designed some years ago, is the large number of discrete components, such as transistors and resistors. Now, one chip can replace dozens of these

ROM
The onboard ROM can be supplemented by means of plug-in cartridges. For example, extended BASIC will enhance the range of commands available

On-Off Switch
This incorporates an LED power-on indicator



CHRIS STEVENS

Cassette Port
The TI99/4A can work with two domestic cassette recorders, and can control the motor of one of them. This means it can cope with crude business programs, which require the copying of data from one deck to another

CPU
The TMS9900 is an early processor, which is why it is physically large. All the address and data lines as well as control lines have separate connections. More modern processors share functions of pins and so reduce the total number on the chip. Unlike other home computers this is a 16-bit microprocessor

Peripheral Port
This is just a PCB edge connector that other units link up with. Texas call it their CRU (Communications Register Unit) interface. Before the general-purpose expansion unit was introduced, individual peripherals were plugged into each other in a long line. This is called 'piggy-backing'

ROM Pack Connector
ROM packs, which Texas call 'Solid state command modules' plug in here. The mechanism is considerably more robust than on most machines

Scratchpad Memory
The chips marked 6810 are special scratchpad memory essential to the operation of the 9900. This microprocessor is different from all other microprocessors in having no internal memory locations (registers) and so needs to use some external memory. This scratchpad memory is not accessible to normal programs

TI99/4A

PRICE
Obtainable from about £60

SIZE
380×260×70 mm

WEIGHT
1.8 Kg (4lbs)

CLOCK SPEED
1MHz

MEMORY
26 Kbytes ROM, 16K user RAM, 8K graphics RAM. There are an extra 256 bytes of 'scratchpad' RAM not normally available to the user. These are used for the internal registers of the 9900, most CPUs have them built in

VIDEO DISPLAY
Character display of 24 rows of 32 columns. There are 16 colours which can be used as foreground and background colours. No user graphics are available on the basic machine but individual 8 × 8 character cells can be defined with a sequence of 16 characters

INTERFACES
Cassette, joystick, video (not TV), a cartridge slot and a connector for the expansion bus

LANGUAGES SUPPLIED
BASIC

OTHER LANGUAGES
Extended BASIC, TI LOGO, UCSD(University of California at San Diego) PASCAL, TI FORTH, and Assembler

COMES WITH
Power supply adaptor, TV adaptor, cassette connector and manuals

KEYBOARD
Typewriter-style with 48 moving keys, including control and function keys. The numeric keys double as function keys, depending on the added software cartridges

DOCUMENTATION
There is one main manual with an addendum for the UK market, which describes how to connect up the computer and how to use the 'solid state command modules'. This introduction is very short and has many diagrams but no photographs. There is a detailed list of commands available in the BASIC, a section giving some example programs, and a short glossary at the end of the manual

Forging Ahead

Piracy is the thorn in the flesh of the software industry. That's why suppliers go to such lengths to protect their programs

TONY SLEEP

TONY SLEEP

Slave Driver
Software cassettes, like music cassettes, are duplicated using a high-speed tape copier. This consists of a master deck, into which the original is placed, and a number of slave units which make recordings simultaneously. Copying both sides of a program cassette takes a matter of seconds. Disks have to be copied individually using normal disk drives

A Hundred To One
Just as it is technically illegal to make copies of other people's music cassettes, copying of programs represents software piracy. Unfortunately for the suppliers, piracy is not only difficult to prevent, but equally difficult to detect and prosecute. Some suppliers claim that for every copy of their programs bought legitimately a hundred illegal copies are made

Software piracy can be defined, simply, as the unauthorised copying of programs. In common with the music business, which the software industry is starting to mirror, piracy happens in different ways and at different levels. At the lowest level, piracy is committed every time a home computer user makes a copy of a program that has been borrowed from a friend. Even the fact that some programs (especially those written in machine code) can't be SAVEd using the normal BASIC commands provides little deterrent, because it is always possible to link two cassette recorders together and copy the program from one to the other — without the need for a computer at all.

Some games suppliers claim that for every copy of a title they sell, up to 100 illegal copies are made. Though it might be argued that some of them can well afford the loss, it must also be remembered that there are a good many people who earn their living from program royalties and who don't drive around in Rolls Royces!

There has been a great deal of controversy concerning dealers who offer software on loan, rent, or the try-before-you-buy schemes — since they make it easier for those who copy programs. Less scrupulous dealers will take this a stage further, and give away pirated copies of popular titles to someone buying a home computer to increase its effective value.

There are even cases of a distributor reproducing programs in quantity and selling them to other dealers — not as risky as it sounds if the package's suppliers are in another country. These products are therefore equivalent to 'bootleg' copies of well-known rock albums.

Hereafter, the piracy becomes more sophisticated and more difficult to pin down. Someone takes an existing program, for example, makes some modifications to it and markets it as his own. The new version may offer a substantial improvement in performance or additional facilities, or may simply feature a change to the 'credits' displayed when the program is first run, and the layout of information on the screen, in order that the package isn't immediately recognisable. This practice is more common with business programs than games.

Whether this process of modification is software piracy in the same sense as pure copying is arguable, which is why so many people get away with it. Software publishers receive little protection from the law and the existing laws of copyright do not protect programs from piracy or modification. Copyright, it seems, applies only to printed material (with special exceptions for music) and therefore computer programs that are stored only in RAM or on cassette are not covered. As with most legal matters, precedents have to be established and they take time and money.

The most woolly area is where a company take an idea from a popular program, and reproduce their own version of it. Note that they aren't copying any of the program code, they are merely taking accurate note of how the game appears on the screen and reacts to the user's input, and then writing a program from scratch to achieve the same effect. The most noteworthy example of this has been PacMan — the arcade game that started out on coin-operated machines, was made available on Atari's own home computers and Video Cartridge System, and subsequently appeared in different variations from a score of software publishers. Each looked slightly different, but each featured the familiar little creature gobbling his way around the maze. Over a period of months Atari successfully managed to eliminate most of these competitors, either by court action or, in the case of smaller operations, simply with the threat of court action.

Generally, software authors and suppliers have to resort to means outside the legal system to protect their program code and royalties. Some suppliers take the laudable view that if they sell their products cheaply enough, there is less incentive for people to copy. On more sophisticated programs, a well-produced manual and attractive packaging afford some degree of protection.

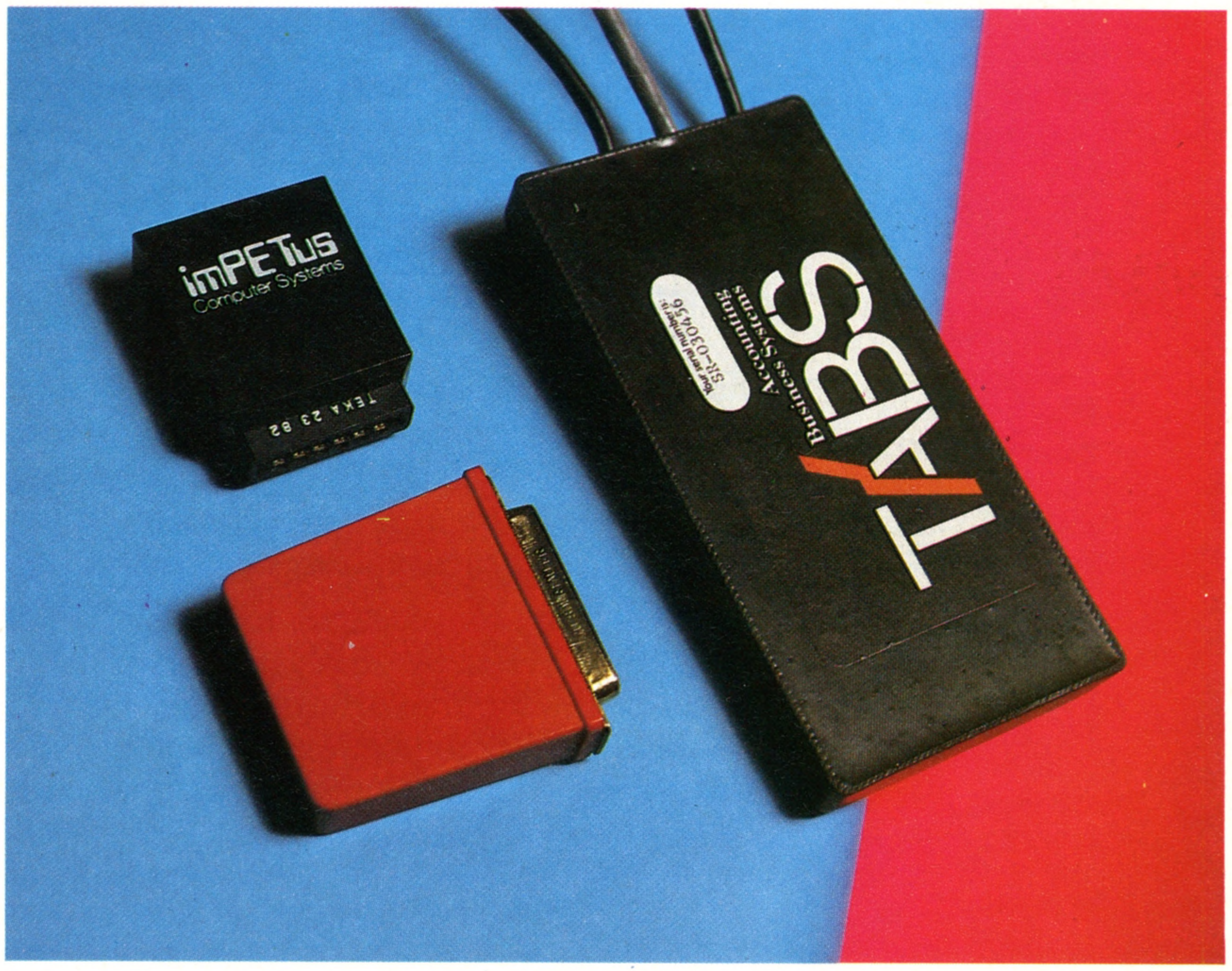
'User-registration' is one means by which more expensive business software is protected: unless you have returned the card from the owner's manual, you won't be able to obtain help and support on the telephone.

So-called 'hard' methods of protection usually involve a matchbox-sized device, called a 'dongle', which must be plugged into one of the computer's interface ports in order for the program to run. The dongle's circuitry incorporates a short electronic code, usually a pattern of ones and zeros burnt into a ROM. At frequent intervals the applications program addresses the dongle; if it doesn't receive the correct code back it will refuse to continue. The code may well be individual to each dongle, which means that each copy of the package must be matched to the dongle it will be sold with. The only way to make illegal copies is by forging the dongle, or re-writing the program code to remove the sections that refer to the dongle — by no means impossible, but well beyond the capability of most home programmers.

A lot of research has been put into methods of achieving the same protection, without additional hardware. The idea, aptly known as 'watermarking', is to have a magnetic code superimposed on the cassette or disk 'behind' the recording of the program itself, which will not transfer to a copy, so the program won't run on any disk or cassette other than the original.

The only economically viable 'hard' protection for the games suppliers is the ROM cartridge, which generally commands higher prices because it avoids the long loading times of cassettes. Nevertheless, even the cartridge is not impregnable — devices now exist which can copy a cartridge either onto a cassette, or onto a new kind of cartridge that can be programmed or re-programmed by the user.

Software piracy is a cops-and-robbers style battle with the protagonists constantly trying to leapfrog each other in ingenuity. It is unlikely ever to be eliminated; at best it can be made sufficiently costly to be only a marginal activity.



Dongles
These are small hardware devices used to protect certain programs against illicit copying. Such programs will not run unless the correct dongle is plugged into one of the computer's interfaces. The electronics inside are usually encased in solid resin, so it's very difficult to interfere with them

IAN McKINNELL

Another Dimension

One-dimensional arrays, as we have seen, store a collection of data that have something in common. Two-dimensional arrays are used for tables and charts

So far we have considered two types of variables, simple variables and subscripted variables. Simple variables are like memory locations where numbers (or character strings) can be stored and manipulated by referring to the variable 'label'. Simple variables can store just one value or string and have 'simple' variable names — N, B2, X, Y3 are examples. Subscripted variables, sometimes called one-dimensional arrays, can store a whole list of values or strings. The number of values or strings that can be held is specified at the beginning of the program using the DIM statement. For example, DIM A(16) establishes that the array labelled A can contain 16 separate values. It should be noted, however, that many BASICS accept A(0) as the first element, so that DIM A(16) actually defines 17 elements. These 'locations' are referred to by using the appropriate subscript. PRINT A(1) will print the first element in the array; LET B = A(12) assigns the value in the 12th element in the array to variable B; LET A(3) = A(5) assigns the value of the fifth element to the third element.

Sometimes, however, we need to be able to manipulate data that is best presented as tables. Note how closely this resembles a spreadsheet (see page 158). Such data could range from tables of football results to a breakdown of sales by item and department in a store. As an example of a typical table of data, consider this breakdown of household expenditure over a one year period:

	RENT	PHONE	ELECTR.	FOOD	CAR
JAN	260.00	25.10	41.50	161.30	50.55
FEB	260.00	35.40	43.75	145.90	46.20
MAR	260.00	29.05	50.70	151.20	43.40
APR	260.00	26.20	44.60	155.30	49.20
MAY	260.00	19.30	39.80	150.95	48.30
JUN	260.00	20.45	32.60	147.65	52.30
JUL	260.00	30.50	26.10	150.35	58.40
AUG	260.00	29.50	22.40	148.05	61.20
SEP	260.00	28.25	24.45	148.60	59.45
OCT	260.00	31.15	34.50	154.90	23.50
NOV	260.00	31.05	39.50	160.05	45.95
DEC	260.00	28.95	42.20	210.60	51.25

Arranging the information in this way allows it to be manipulated in a number of ways relatively simply. It is easy, for example, to find the total expenditure in March by simply adding up all the figures in the row for March. It is just as easy to find the total expenditure for the year on the telephone or the car by adding up the vertical columns. Similarly, it is easy to find monthly or yearly averages. This table is called a two-dimensional array. It has 12 rows and five columns.

Two-dimensional arrays such as this can also be represented in BASIC in much the same way as single-dimension arrays. The difference is that the variable now needs two subscripts to reference any location.

If we were writing a BASIC program using this table of information, the simplest thing would be to treat the whole table as a single two-dimensional array. Just as with ordinary subscripted arrays, we give it a variable name. Let's call it A (for 'Array'). Again, as with ordinary subscripted arrays, it will need to be DIMensioned. As there are 12 rows and five columns, it is dimensioned thus: DIM A(12,5). The order in which the two subscripts are put is important; the convention is that rows are specified first and columns second. Our table above has 12 rows (one for each month) and five columns (one for each of the five categories of expenditure), it is therefore a 12-by-5 array.

The DIM statement serves two essential functions. It sets aside enough memory locations in the computer's memory for the array, and it allows each of the locations to be specified by the variable name followed, in brackets, by the row and column positions. The DIM statement DIM X(3,5), for example, would create a variable X able to represent an array with three rows and five columns.

Look at the table and assume that the information has been entered as the elements in a two-dimensional array labelled A. Find the values present in A(1,1), A(1,5), A(2,1), A(3,3) and A(12,3).

It is possible to enter a table of information as an array in part of a program by using LET statements, for example.

```
30 LET A(1,2) = 25.1
40 LET A(1,3) = 41.5
50 LET A(1,4) = 161.30
:
:

610 LET A(12,5) = 51.25
```

But this is clearly a laborious way of doing things. A far simpler method is to use either READ and DATA statements or the INPUT statement with nested FOR...NEXT loops. Let's see how it could be done using the READ statement:

```
10 DIM A(12,5)
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
40 READ A(R,C)
50 NEXT C
60 NEXT R
70 DATA 260, 25.1, 41.5, 161.3, 50.55, 260, 35.4,
    43.75
80 DATA 145.9, 46.2, 260, 29.05, 50.7, 151.2, 43.4,
    260
90 DATA 26.2, 44.6, 155.3, 49.2, 260, 19.3, 39.8,
    150.95
100 DATA 48.3, 260, 20.45, 32.6, 147.65, 52.3,
    260, 30.5
110 DATA 26.10, 150.35, 58.4, 260, 29.5, 22.4,
    148.05, 61.2, 260
120 DATA 28.25, 24.45, 148.6, 59.45, 260, 31.15,
    34.5
130 DATA 154.9, 23.5, 260, 31.05, 39.5, 160.05,
    45.95
140 DATA 260, 28.95, 42.2, 210.6, 51.25
150 END
```

There are a number of important points to note about this program. The first is that the DIM statement is right at the beginning of the program. A DIM statement should be executed only once in a program and so it is usual to place it near the beginning or before any loops are executed. The second point to note is that there are two FOR...NEXT loops, one to set the 'row' part of the subscript and one to set the 'column'. These two loops do not follow one after the other; they are 'nested' one inside the other. Notice the limits chosen. FOR R = 1 TO 12 will increment the value for the row from one to 12; FOR C = 1 TO 5 will increment the value for the column from one to five.

Right in the middle of the nested loop is the READ statement. The crucial part of the program is:

```
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
40 READ A(R,C)
50 NEXT C
60 NEXT R
```

The first time through, after lines 20 and 30 have been executed, the values of R and C will both be one, so line 40 will be equivalent to READ A(1,1). The first item of data in the DATA statement is 260, so this value will be assigned to the first row and the first column of the array. The choice of eight elements to each DATA statement is purely arbitrary.

After that has happened, the NEXT C statement sends the program back to line 30 and the value of C is incremented to two. Line 40 is now equivalent to READ A(1,2) and the next item of data, 25.1, will be assigned to the first row and the second column of the array. This process is repeated until C has been incremented to 5. After that, the NEXT R statement in line 60 returns the program to line 20 and R is incremented to two. Line 30 will set C to one again and so now line 40 will be equivalent to READ A(2,1).

Nesting loops in this way is very useful, but care is needed. Each loop must be nested completely within another loop and the order of the NEXT statements must be carefully observed. Notice how the first loop, FOR R, has the second NEXT statement. When there are two loops, one nested inside the other, the first loop is called the outer loop and the second is called the inner loop. The whole of the inner loop will always be completed before the index of the outer loop is incremented. It is possible to nest loops to as many 'depths' as required by the program, but such programs can become complex and difficult to follow and debug. It is bad programming practice to put branching instructions inside loops and GOTOs are to be avoided.

Let's look at the DATA statements. Notice that commas are used to separate data items, but there must be no comma before the first data item or after the last. We have inserted spaces between each data item, but this is not normal. Mistakes when entering the data are easy to make and difficult to spot later. As many DATA statements as required may be used. Each new line needs to start with a DATA statement. The data is read in one item at a time, starting from the beginning of the first DATA statement and working through until all the items have been read. Be sure that the number of data items is correct or you will get an error message when the program is run.

The program presented so far does not actually do anything except convert appropriate data into a two-dimensional array. After the program has been entered and RUN, nothing will apparently happen and all you will see on the screen will be the BASIC prompt. To test that the data is correctly placed, try a few PRINT commands. (A command in BASIC is a keyword that can be immediately executed without having to be within a program and does not therefore need a line number. Examples are LIST, RUN, SAVE, AUTO, EDIT and PRINT). PRINT A(1,1) <CR> should cause the number 260 to appear on the screen. What will be printed by the following commands?

```
PRINT A(12,1)
PRINT A(1,5)
PRINT A(5,1)
PRINT A(5,5)
```

To make the program do something useful, it will need to be extended. As it stands it forms an adequate basis for a 'main program'. To use it as part of a larger, more useful program, modules can be written as subroutines to be called by GOSUBs inserted at suitable points before the END statement.

In the early stages of designing a household accounts program, it is best to start with a simple written description of the general requirements. We might decide that we want to be able to have totals and averages calculated for monthly expenditure or by category (electricity, for example). We can work out the details of how to derive these results at a later stage. If there is a choice to be made within the program about which subroutines we wish to be executed we will probably want to be prompted by a 'menu' which will direct control to the appropriate subroutines as a result of our response. An early sketch of the program at this stage might look like this:

```
MAIN PROGRAM
(DATA ENTRY)

MENU
(SELECT SUBROUTINES)

END
```

A little further refinement may show that we will need subroutines to calculate totals for months or for categories (MONTHTOTAL and CATTOTAL), average monthly expenditure (MONTHAV) and average yearly expenditure by category (CATAV). The reason for using one-word names for these subroutines is to help us to plan the program without having to worry about details such as line numbers at this stage. On reflection we may decide that even the main menu selection part of the program should be dealt with as a subroutine in order to keep the main part of the program as a separate module. The next stage of refinement of the program will look like this:

```
MAIN PROGRAM (DATA ENTRY)
   MENU (CALL SUBROUTINE)
END

**SUBROUTINES**

1 MENU
2 TOTALS
3 AVERAGES

(2) TOTALS
4 MONTHTOTAL
5 CATTOTAL

(3) AVERAGES
6 MONTHAV
7 CATAV
```

This sketch of the program shows that the MENU subroutine will give us a choice of either TOTALS or AVERAGES. Both of these will themselves be subroutines. The TOTALS subroutine will give a further choice of MONTHTOTAL or CATTOTAL. These will be the subroutines that perform the actual calculations.

The AVERAGES subroutine will give a choice of MONTHAV or CATAV, and again these will be subroutines to perform the appropriate calculations. At this stage it should be possible to see whether our 'program' will do what we want, without doing any actual coding (detailed program writing in BASIC). If we can be satisfied that 'so far so good', we are readv to tackle the writing of the modules (subroutines) themselves. The only change needed to the main program will be a subroutine call before the END statement, so we could add:

```
145 GOSUB **MENU**
```

Note that we are still using 'names' for subroutines rather than line numbers. Many languages, PASCAL, for example, allow sub-programs to be called by name, but most versions of BASIC do not and actual line numbers are needed instead. However, these 'details' can be incorporated later.

Let's see how the MENU subroutine could be written (line numbers have been omitted and you can add appropriate ones if you wish to implement this program).

```
REM THE **MENU** SUBROUTINE
PRINT "WOULD YOU LIKE T(OTALS) OR
   A(VERAGES)?"
PRINT "TYPE EITHER A OR T"
INPUT L$
IF L$ = "T" THEN GOSUB *TOTALS*
IF L$ = "A" THEN GOSUB *AVERAGES*
RETURN
```

Note: we are marking the subroutines called by enclosing them within *—–* marks. You will have to use line numbers instead. These can be inserted when you are in a position to know what they are.

Suppose you type T for TOTALS. The program will then call the TOTALS subroutine. This will then present another menu and could look like this:

```
REM THE **TOTALS** SUBROUTINE
PRINT "WOULD YOU LIKE TOTALS FOR"
PRINT "M(ONTH) OR C(ATEGORY)?"
PRINT "TYPE EITHER M OR C"
INPUT L$
IF L$ = "M" THEN GOSUB *MONTHTOTAL*
IF L$ = "C" THEN GOSUB *CATTOTAL*
RETURN
```

Suppose you selected M for MONTHTOTAL. Let's see how we could write a module to calculate the total expenditure for any month in the year.

```
REM THE **MONTHTOTAL** SUBROUTINE
REM THIS CALCULATES TOTAL EXPENDITURE FOR
REM ANY MONTH
PRINT "SELECT MONTH"
PRINT "1-JAN 2-FEB 3-MAR 4-APR 5-MAY"
PRINT "6-JUN 7-JUL 8-AUG 9-SEP"
PRINT "10-OCT 11-NOV 12-DEC"
PRINT "TYPE A NUMBER FOR THE MONTH"
LET T = 0
INPUT M
FOR C = 1 TO 5
LET T = T + A(M,C)
NEXT C
PRINT "THE TOTAL EXPENDITURE FOR MONTH"
PRINT "NUMBER ";M;" IS ";T
RETURN
```

The number representing the month is typed in and the INPUT statement assigns the number to the variable M (MONTH). M is used to specify the 'row' subscript of the two-dimensional array A. The FOR-NEXT loop increments the value of C (column) from one to five so the first time through the loop, if we had selected three for March, the LET statement would be equivalent to LET T = T + A(3,1). The next time round it would be equivalent to LET T = T + A(3,2) and so on.

This week we'll leave you to write the other subroutines, or try out the other exercises. Two-dimensional arrays are ideal for any program that involves tables of data, be they statistical, financial or any other quantity.

Answers To Exercises On Page 175

RND Function

```
    40 IF R >6 THEN LET R = 1
```

Loop And Average

```
    5 FOR L = 1 TO 100
    :
    80 LET T = T + R
    90 NEXT L
    100 LET A = T/100
    110 PRINT A
    120 END
```

Replace With Subroutine

Delete lines 5, 80, 90, 100, and 110 in the solution above. Change lines 10 to 70 to (say) 1000 to 1070. Check that line 40 is as in the RND Function solution above. Then add 1080 RETURN. Incorporate the result into the main program. Change lines 50 and 130 in the main program to read 50 GOSUB 1000 and 130 GOSUB 1000.

INKEY$

```
    10 PRINT "TYPE ANY KEY"
    20 LET A$ = INKEY$
    30 IF A$ ="" THEN GOTO 20
    40 PRINT "THE KEY YOU HIT WAS";A$
    50 END
```

(On the Spectrum add: 15 IF INKEY$ <>"" THEN GOTO 15)

Timing Loop

```
    5 PRINT "HIT THE SPACE-BAR AFTER 10 SECONDS"
    10 FOR L = 0 TO 1
    20 LET R = R + 1
    30 IF INKEY$ = "     " THEN GOTO 60
    40 LET L = 0
    50 NEXT L
    60 PRINT "THE VALUE OF R AFTER 10 SECONDS IS
       ";R
    70 END
```

IF...THEN

```
    10 GOSUB 1000
    20 PRINT "GUESS THE NUMBER"
    30 FOR G = 1 TO 5
    40 INPUT N
    50 IF N >R THEN GOTO 110
    60 IF N <R THEN GOTO 130
    70 IF N = R THEN GOTO 150
    80 NEXT G
    90 PRINT "NO MORE GOES. YOU LOSE!"
    100 GOTO 500
    110 PRINT "YOUR GUESS IS TOO LARGE"
    120 GOTO 80
    130 PRINT "YOUR GUESS IS TOO SMALL"
    140 GOTO 80
    150 PRINT "YOU ARE RIGHT,
       CONGRATULATIONS".
    500 END
    1000 REM **RANDOM SUBROUTINE**
    (Insert your subroutine here.)
    1020 RETURN
```

Errata

We regret that errors appeared in the Basic Programming course in Issues 5 and 7. Two of the LET statements on page 99, Issue 5, should have read:

LET X(5) = 31
LET X(6) = 30

On page 100 we should have said:

910 LET M = 2

On page 137, Issue 7, two lines in the Basic Flavours box, concerning the INSTR command, should be revised to read:

525 NEXT P

(for Commodore machines and the Oric-1), and:

540 FOR P = 1 TO L

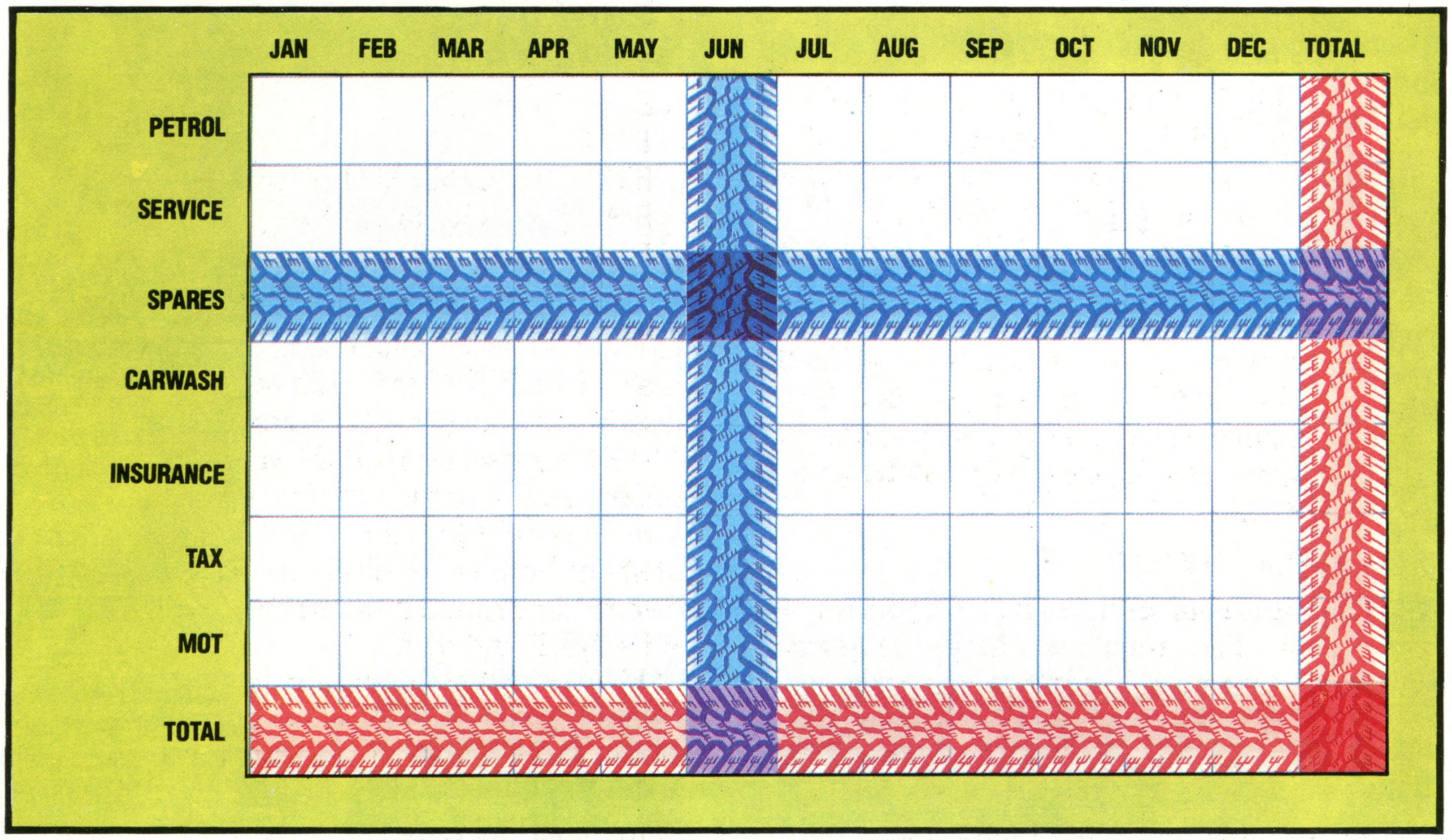
(for the ZX81 and Spectrum)

Exercises

■ **Assigning Values** Write a program that assigns values to the elements ('Petrol', 'Service' etc.) of the matrix (see illustration below). Next, write a subroutine that asks for a month, and an expense heading, and prints the contents of the box thus specified. Finally, write a subroutine that finds the sum of each column, and places the result in the bottom box, does the same across the rows, and then calculates the grand total, which it stores in the lower right box.

■ **Bugs** The following program would not run properly and would produce an error message. There are two mistakes. Find them and make appropriate corrections.

```
●10 DIM A(3,4)
 20 FOR R = 1 TO 3
 30 FOR C = 1 TO 4
 40 READ A(R,C)
 50 NEXT C
 60 NEXT R
 70 FOR X = 1 TO 3
 90 FOR Y = 1 TO 4
 100 PRINT A(Y,X)
 110 NEXT Y
 120 NEXT X
 130 DATA 2,4,6,8,10,12,14,16,18,20,22
 140 END
```



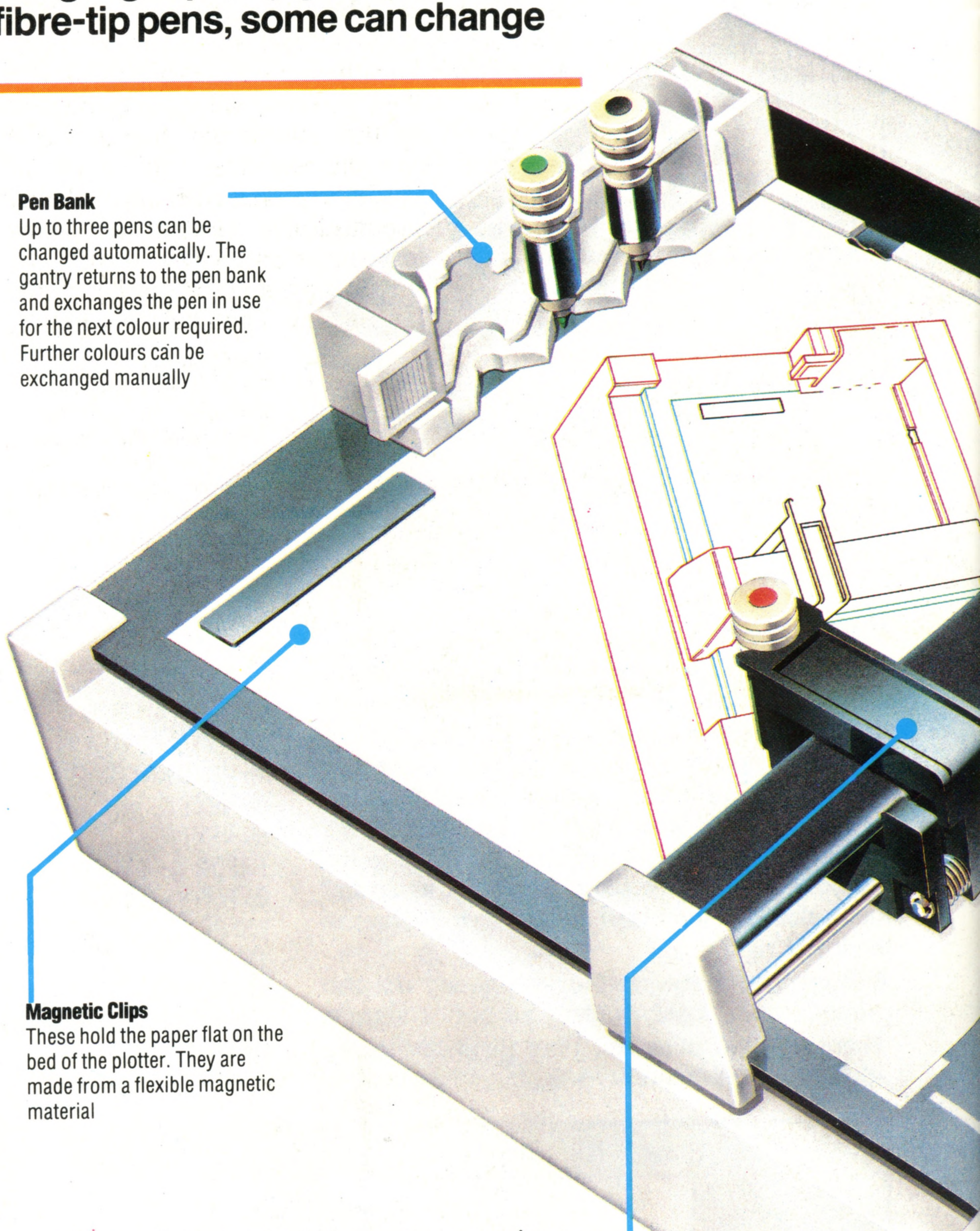
	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	TOTAL
PETROL													
SERVICE													
SPARES													
CARWASH													
INSURANCE													
TAX													
MOT													
TOTAL													

Car Expenses

The picture shows a grid of 8 × 13 squares. The rows represent different elements of the cost of running a car, and the columns represent the different months of the year. Follow the exercise on 'Assigning Values' to calculate the yearly cost of running a car

Intricate Plot

A plotter is the best means of producing high quality graphic output from your computer. Working with fibre-tip pens, some can change colour automatically



Pen Bank
Up to three pens can be changed automatically. The gantry returns to the pen bank and exchanges the pen in use for the next colour required. Further colours can be exchanged manually

Magnetic Clips
These hold the paper flat on the bed of the plotter. They are made from a flexible magnetic material

Pen Holder
The currently selected pen is clamped — in this case magnetically — into this holder, which moves down and places the pen in contact with the paper

The ability to create printed copies of diagrams that appear on a computer screen is an essential requirement for many serious computer users. Engineers, scientists, technical artists and businessmen all need accurate diagrams and charts that conventional printers are not capable of producing. The only device that can create these images is a plotter and, until recently, these have been too expensive for the home computer user.

However, with the introduction of devices like the four-pen printer/plotter mechanism used in the Tandy/CGP-115 and Oric MCP-40 printer, graphical output is at last within reach of the emptiest wallets. A whole range of plotters has recently appeared on the market that offer features previously only found in machines costing thousands of pounds.

The need for a plotter is generally governed by the type of output being generated by the computer. An engineer or draughtsman will need accurate drawings of equipment and installations, a businessman might want charts and graphs showing sales figures. Producing these on conventional printers is a very laborious process and the results will appear only in black and white. The only other low cost option is to take a colour photograph of the screen and while this might suffice for business charts, it certainly won't be accurate enough for a designer or architect.

Plotters work in an entirely different way from printers: they draw lines between two points rather than creating their output from preformed characters or patterns of dots. The basic principle behind all the various systems is that of the X, Y coordinate. Just as a graph can be plotted by defining the coordinates through which the line must pass, so any shape can be broken down into a series of coordinates. To be able to join these coordinates together in order to recreate the shape, there must be some form of movement. So the pen is fixed to a travelling gantry that can move in the X direction (left and right) while the pen moves along the gantry in the Y direction (ıp and down).

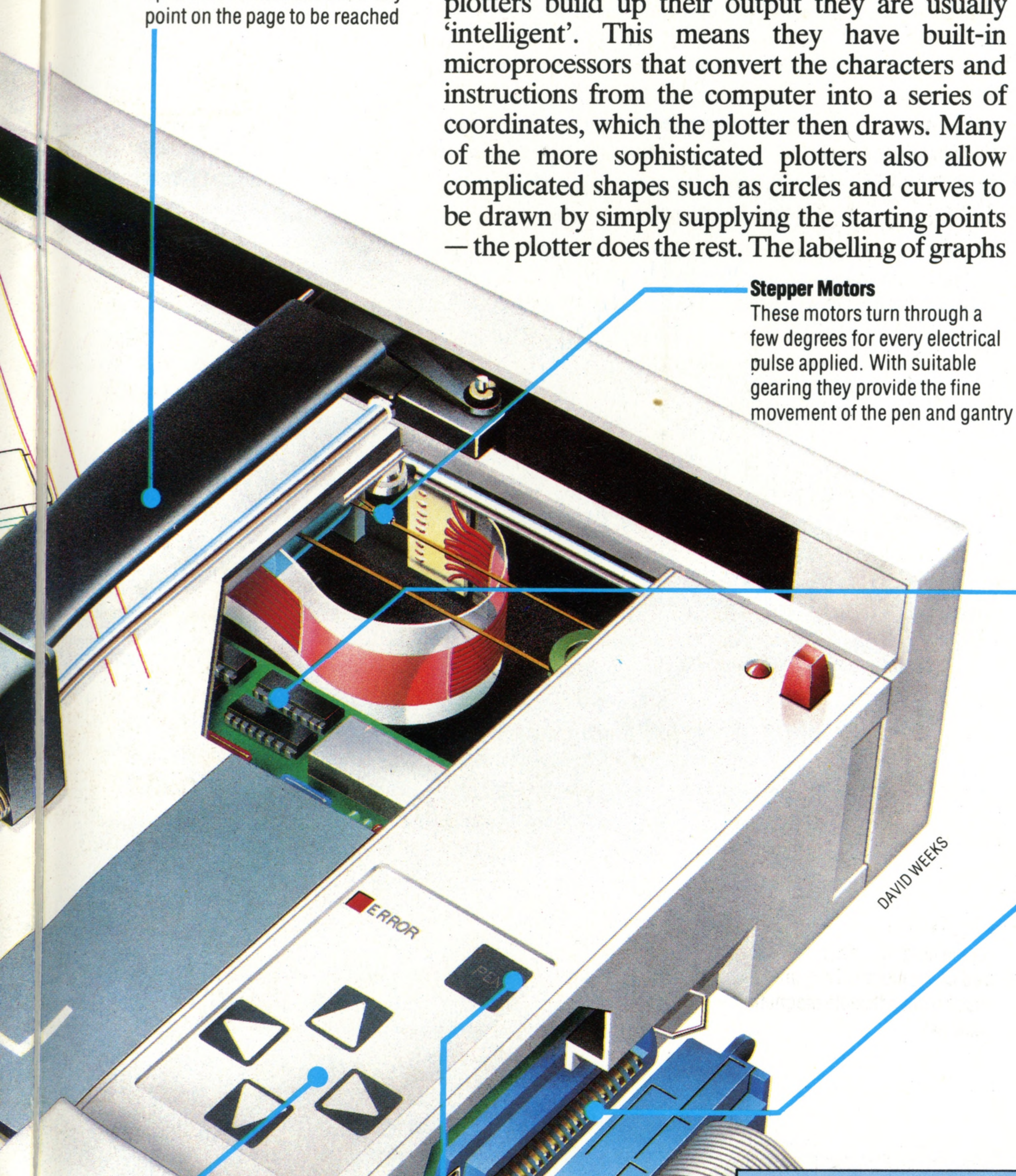
The traditional type of plotter is known as a 'flat bed' plotter because the paper is fixed to a flat plate with the gantry travelling over the top — this is shown in the illustration. Its disadvantage is that the plotter must be at least as big as the piece of paper.

One method of reducing the size is to adopt a large-scale version of the four-pen plotter idea (see illustration), in which the paper moves in one direction and the pen moves in the other. Examples of this are the Strobe 100 and the Hewlett Packard Sweetlips plotters. The movement of the paper must be as precisely controlled as the motion of the gantry in the flat bed type, and is achieved by using a stepper motor. A stepper motor is a very special type of motor that only rotates by a fraction of a turn for each pulse of power that is applied. It is mainly found in disk drives, where it controls the positioning of the head on the surface of the disk, and in robot devices (see page 176).

Connecting a plotter to a computer is generally the same as connecting a printer, at least in terms of the interface. Plotters are usually available with either serial (RS232) or parallel (Centronics or IEEE488) interfaces, which can be connected to the port normally used by a printer. The programming is often a little more complicated in that, instead of just sending the results of a program to be printed, information about the way the results are to be presented must also be sent. This is generally done in much the same way as a diagram would be built up on the screen.

Because of the complicated way in which plotters build up their output they are usually 'intelligent'. This means they have built-in microprocessors that convert the characters and instructions from the computer into a series of coordinates, which the plotter then draws. Many of the more sophisticated plotters also allow complicated shapes such as circles and curves to be drawn by simply supplying the starting points — the plotter does the rest. The labelling of graphs and diagrams and the colouring-in of pie charts and bar graphs are often automatic processes, making the programming much simpler.

Many plotters come complete with software that allows them to be used directly from within a program rather like a paper copy of the screen. If this type of program is not provided, the user will have to work out the necessary routines to translate screen information into the appropriate codes in order to drive the plotter. Some plotters don't feature built-in character sets, so even the codes for the letters and numbers will have to be created. This does at least allow the user to design his own characters and typefaces. Once a shape has been generated, it can be plotted at any position and in any orientation or size, so a library of shapes can be built up for repeated use. Routines to plot circles and curves and shapes in sections of graphs are often very useful, especially in the field of business graphics and these may also have to be created. However, the principles of creating a drawing from coordinates on the screen are just the same as those required to create the shape on paper, so the programming is usually quite simple.



Pen Gantry
The gantry can be positioned at any point across the page (the X axis) and the pen holder is then moved into position along its length (the Y axis). Combinations of left to right and up and down motions allow any point on the page to be reached

Stepper Motors
These motors turn through a few degrees for every electrical pulse applied. With suitable gearing they provide the fine movement of the pen and gantry

Circuit Board
Plotters are usually 'intelligent' devices – they can be given a high-level command such as 'draw a circle with specified radius and centre', and the plotter works out how to move the pen. The circuit board contains its own microprocessor, ROM and RAM

Interface Connection
Plotters connect to the computer by means of a standard interface such as RS232 (serial) or Centronics (parallel). To the computer it appears just like a printer, though different commands will be needed to drive it

Pen Lift Control
This allows the pen to be manually placed in contact or lifted off the paper

Pen Motion Controls
The pen can be manually positioned on the page by these controls

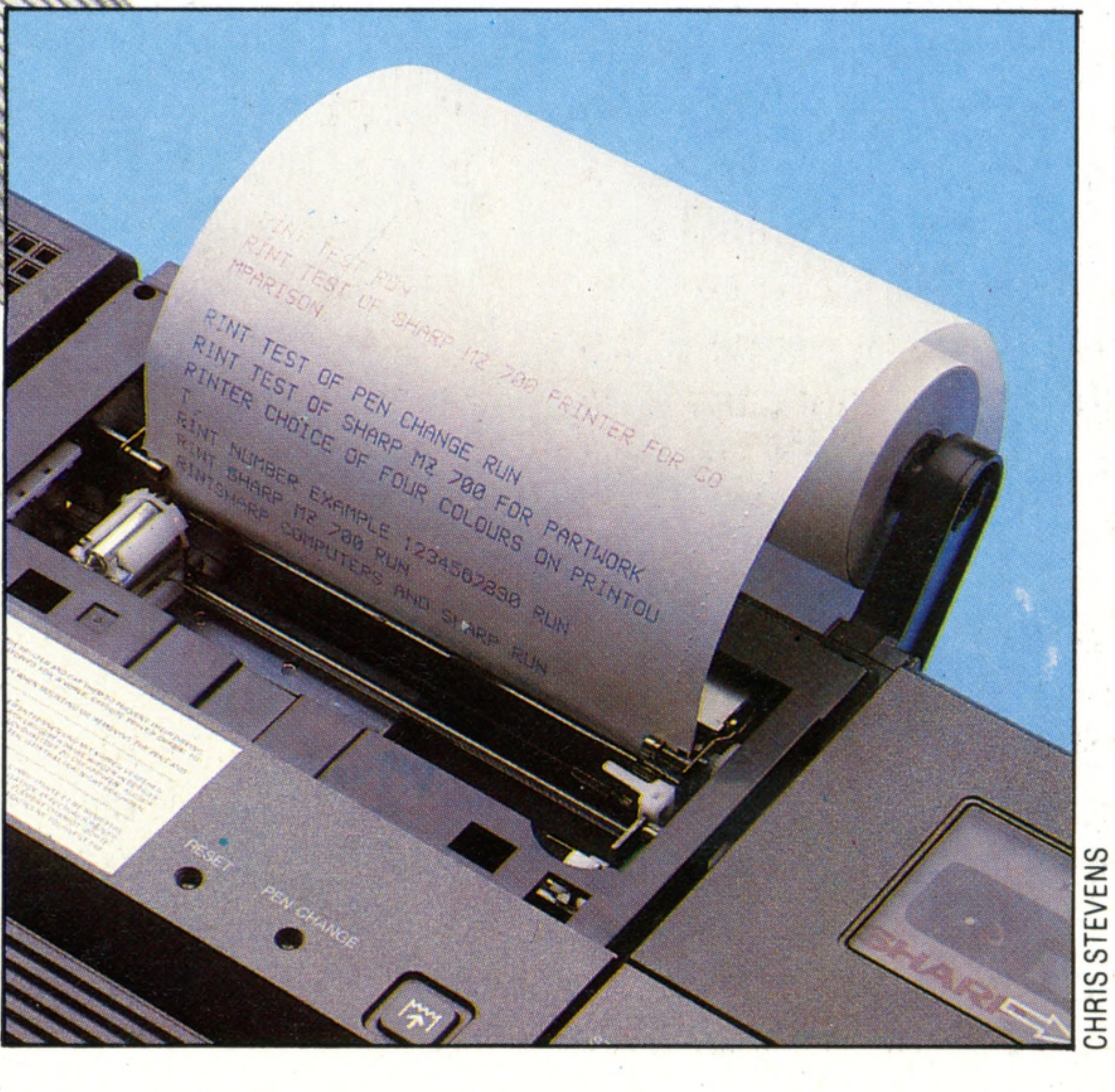
DAVID WEEKS

The Four-Pen Plotter/ Printer

This mechanism captured the attention of the micro industry when it first appeared in the Sharp CE-150 printer. Its bigger brothers in the form of Tandy's CGP-115 and the Oric MCP-40 have helped bring low-cost colour printing to the home computer user.

Like all good ideas the system is amazingly simple in concept. A roll of paper is pulled through the mechanism by a spiked roller. The paper is moved both backwards and forwards in very precise steps while a pen carrier holding four miniature ballpoint pens moves across the surface from left to right and vice versa.

To create the output, which can be text or graphical, the pen carrier is rotated until the correct colour is in position and then the pen is pressed against the paper. Horizontal lines are created by the pen moving while the paper is stationary, vertical lines use the movement of the paper with the pen fixed in place. Combinations of the two movements produce diagonals and curves. The quality of the printing is very high, although the restricted paper width makes it unsuitable for word processing and other serious uses


RINT TEST OF PEN CHANGE RUN
RINT TEST OF SHARP MZ 700 FOR PARTWORK
PRINTER CHOICE OF FOUR COLOURS ON PRINTOU
RINT NUMBER EXAMPLE 1234567890 RUN
PEN CHANGE
SHARP

CHRIS STEVENS

Alan Turing

This British mathematician gave his name to the accepted test for machine intelligence. Much of his work, however, was for military intelligence during the war

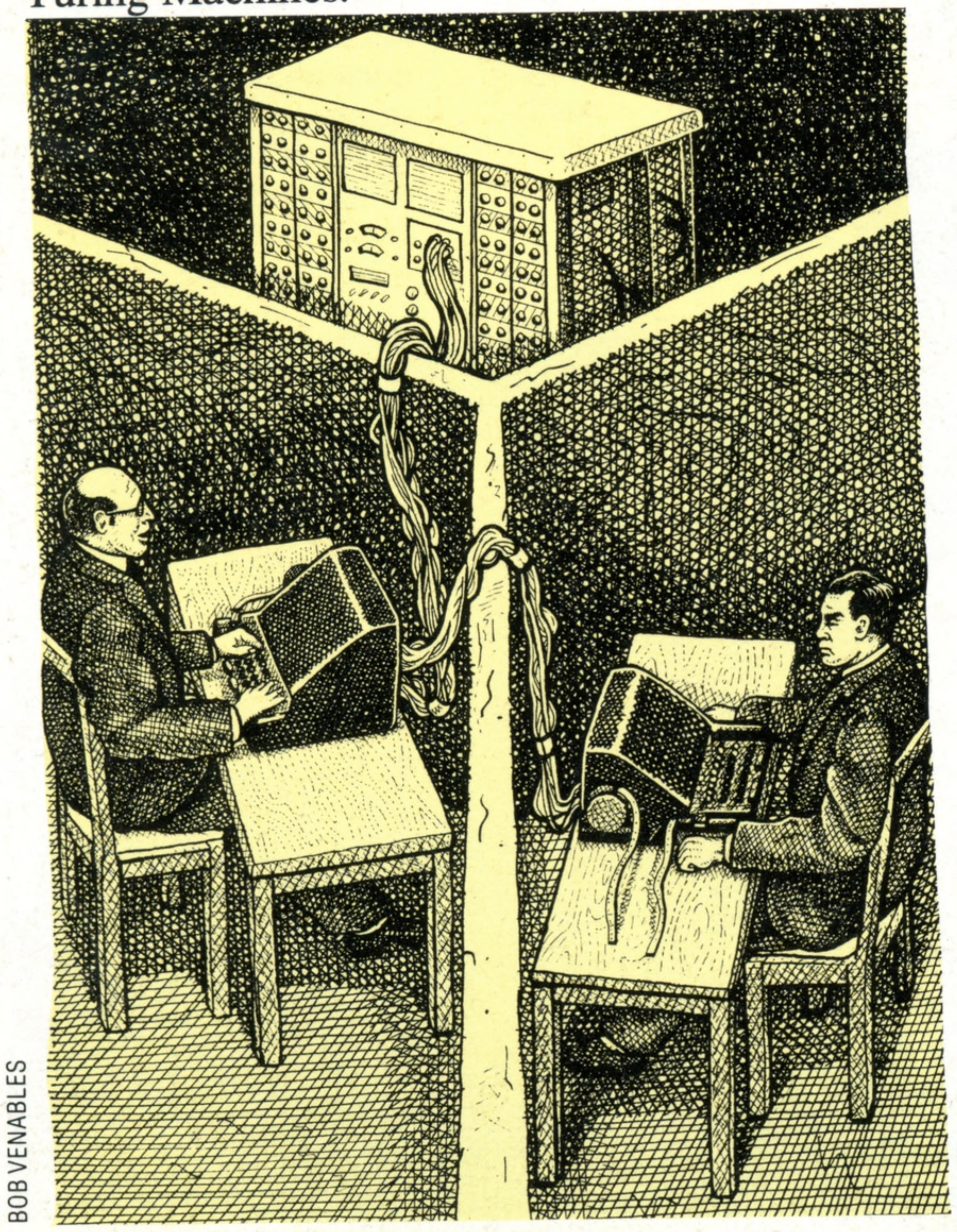
The young Alan Turing showed a remarkable insight into science. He wrote to his mother from school 'I seem always to want to make things from the thing that is commonest in nature'. Mathematicians show their talent early and as soon as Turing could read and write he was factorising hymn numbers and designing amphibious bicycles.

While his father was away in Madras working in the Indian Civil Service, Turing was winning school prizes and then the scholarship that took him to King's College, Cambridge. It was at Cambridge, first as a student and then as a fellow of King's, that his interest began to focus on the problems of mathematical logic.

In 1931 the Czech mathematician Kurt Gödel astonished the scientific world with the discovery that there were mathematical theorems that were true yet could never be proved. Alan Turing set out to investigate those which could be proved.

He proposed a machine, the construction of which he left to the imagination, that could carry out mechanically the processes usually performed by a mathematician. For each process there was one machine — for example, a machine to add, another to divide, and a third to integrate and so forth. These machines later came to be known as Turing Machines.

Mathematical Feat
Alan Turing (1912-1954) found inspiration and relaxation through long-distance running. He was intrigued by the effect of physical exertion on creativity and mental agility



Can Machines Think?
To answer this question, Turing proposed his famous test, called the Imitation Game, but which has subsequently become known as the Turing Test. A man is put into a room that features a teleprinter (keyboard-cum-printer). This is linked to a teleprinter in another room, operated by another man; and also to the computer under test. The first man is allowed to ask any questions he likes of either. If he is unable consistently to determine when he is communicating with the man and when with the computer, then the machine may be deemed to be intelligent. After all, the argument goes, we have no way of telling for certain whether other people do think or are conscious, except by a comparison of their reactions to circumstances with our own

BOB VENABLES

Turing investigated the workings of these imaginary machines and came to a remarkable conclusion. Rather than each mathematical process needing a separate machine, it was possible to design a 'universal' device that could be made to imitate any other of the specialist machines by being 'programmed'. Turing had stumbled upon the theory of the programmable computer.

When the Second World War broke out Turing was quickly recruited from the academic world to the Government School of Codes and Ciphers at Bletchley Park, Buckinghamshire. Had it not been for the war, his machines might have remained imaginary, but Bletchley Park was involved with the highly secret and urgent work of breaking German military codes.

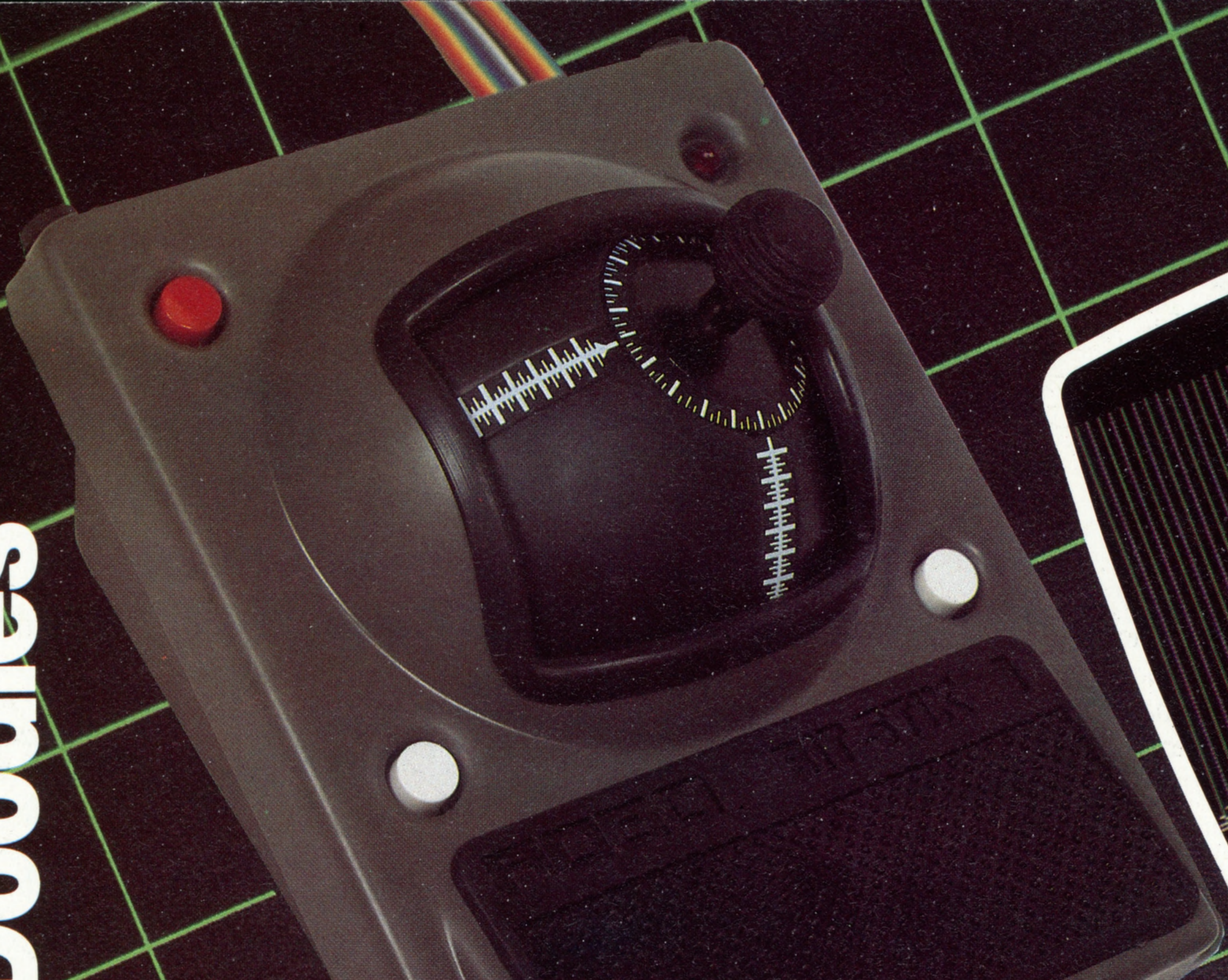
Because these codes could be changed each day, machines were needed to crack the ciphers before new ones were introduced. Bletchley Park became a huge information processing centre. In the middle of the war Turing was sent to America to establish secure codes for transatlantic communications between the Allies.

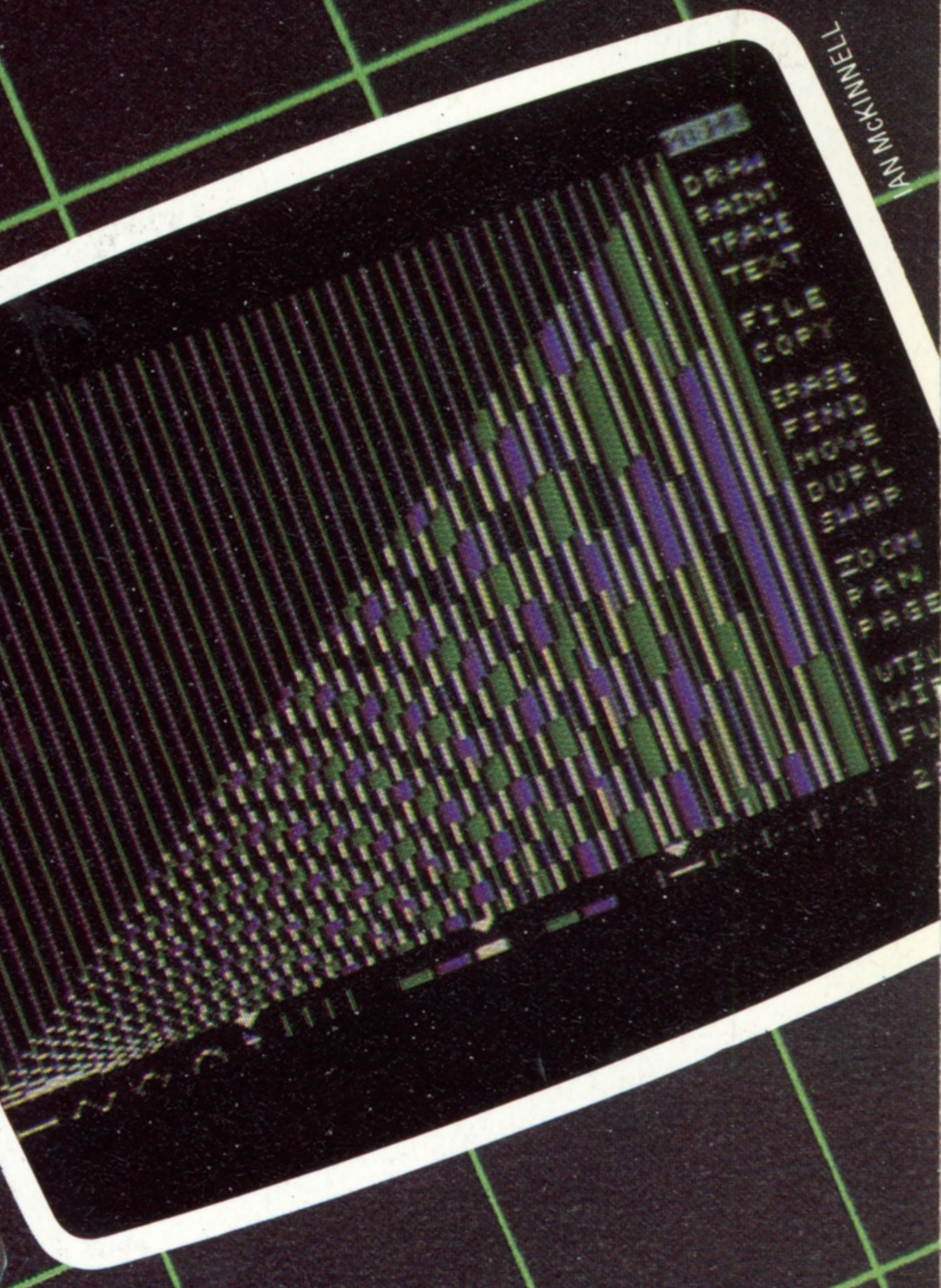
The secret nature of his work at this time means that few records of his movements are available. However, it is widely supposed that he met Von Neumann while at Princeton, New Jersey. Towards the end of the war Turing was asked to draw up plans for an all-British computer for the National Physical Laboratory, to be called ACE.

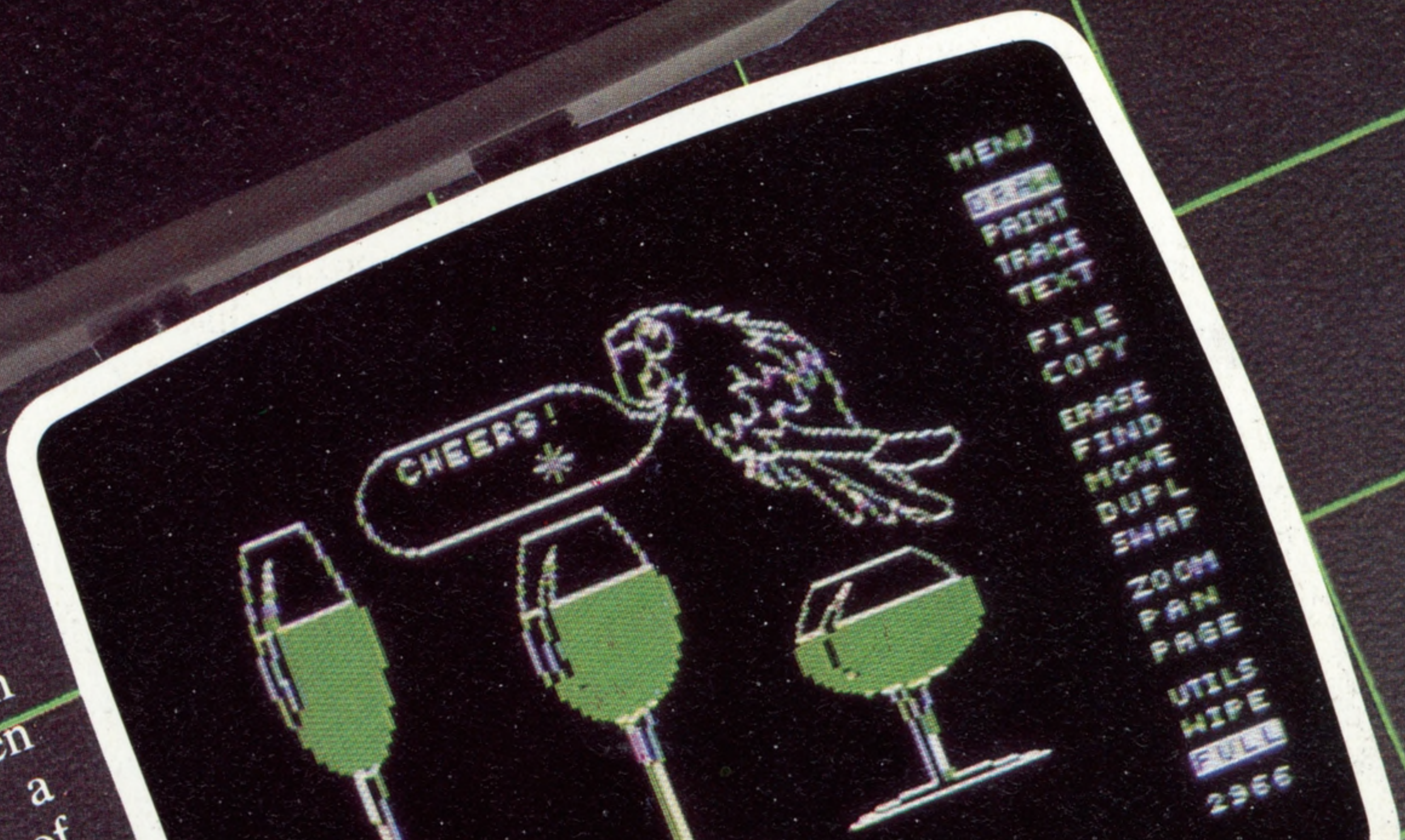
The Automatic Computing Engine was named partly in honour of Babbage's Analytical Engine. Like this pioneering machine, ACE took a long time to be constructed, but in many ways it was far in advance of ENIAC (see page 46). Frustrated at the slow progress, Turing resigned and moved to Manchester where he joined the university's computer project. At the same time he became a consultant to the Ferranti company and subsequently became involved in the first computers to be built in Britain.

Turing was an eccentric who pursued what he knew to be important without regard for social conventions or legal constraints. A friend said he was 'divinely retarded' when it came to seeing faults in others, but his scientific genius was flawless. In 1952 he was convicted on charges relating to homosexuality, and committed suicide two years later. Who can tell what a contribution Turing might have made to artificial intelligence, had he still been alive today?

Digital Doodles

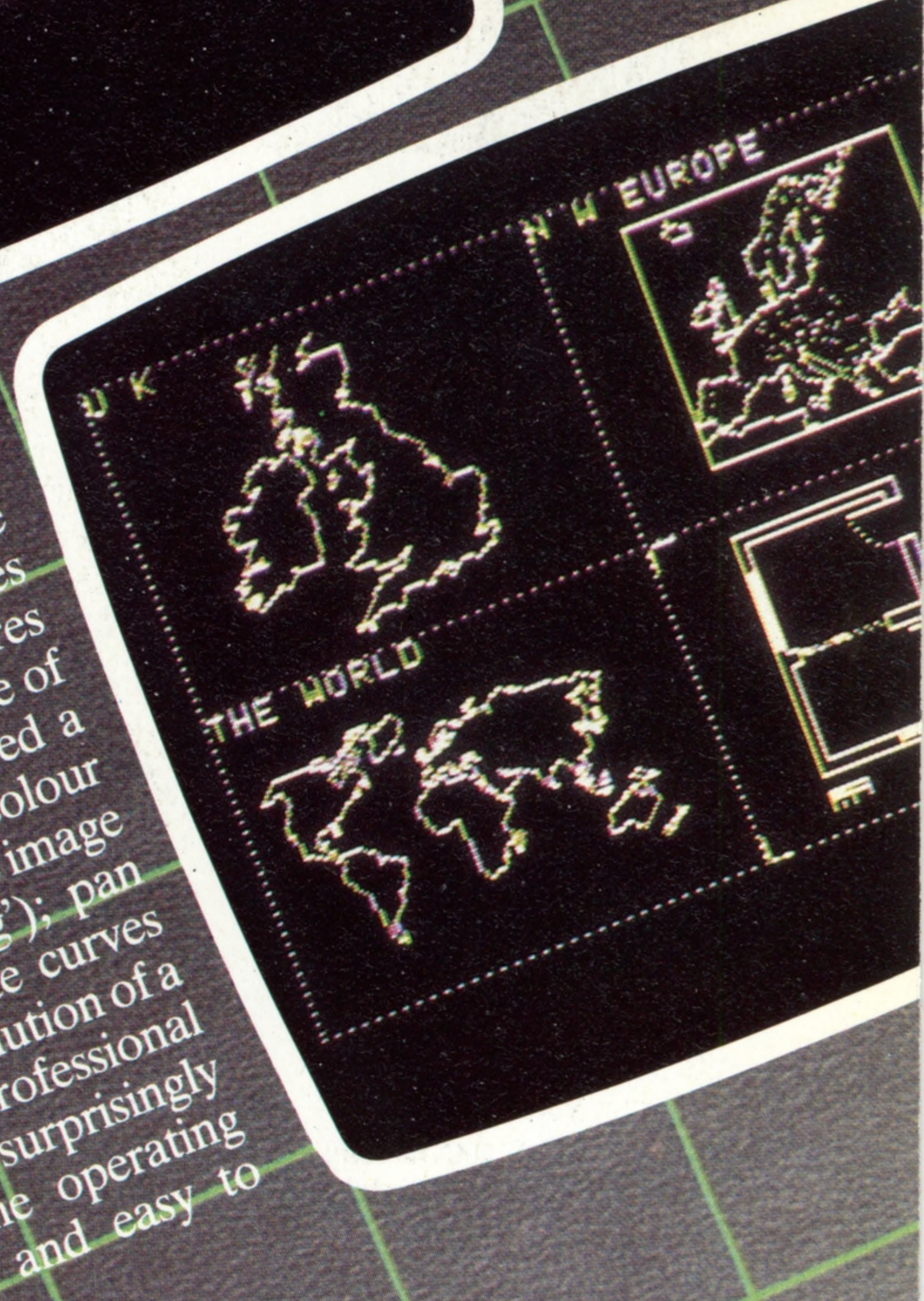


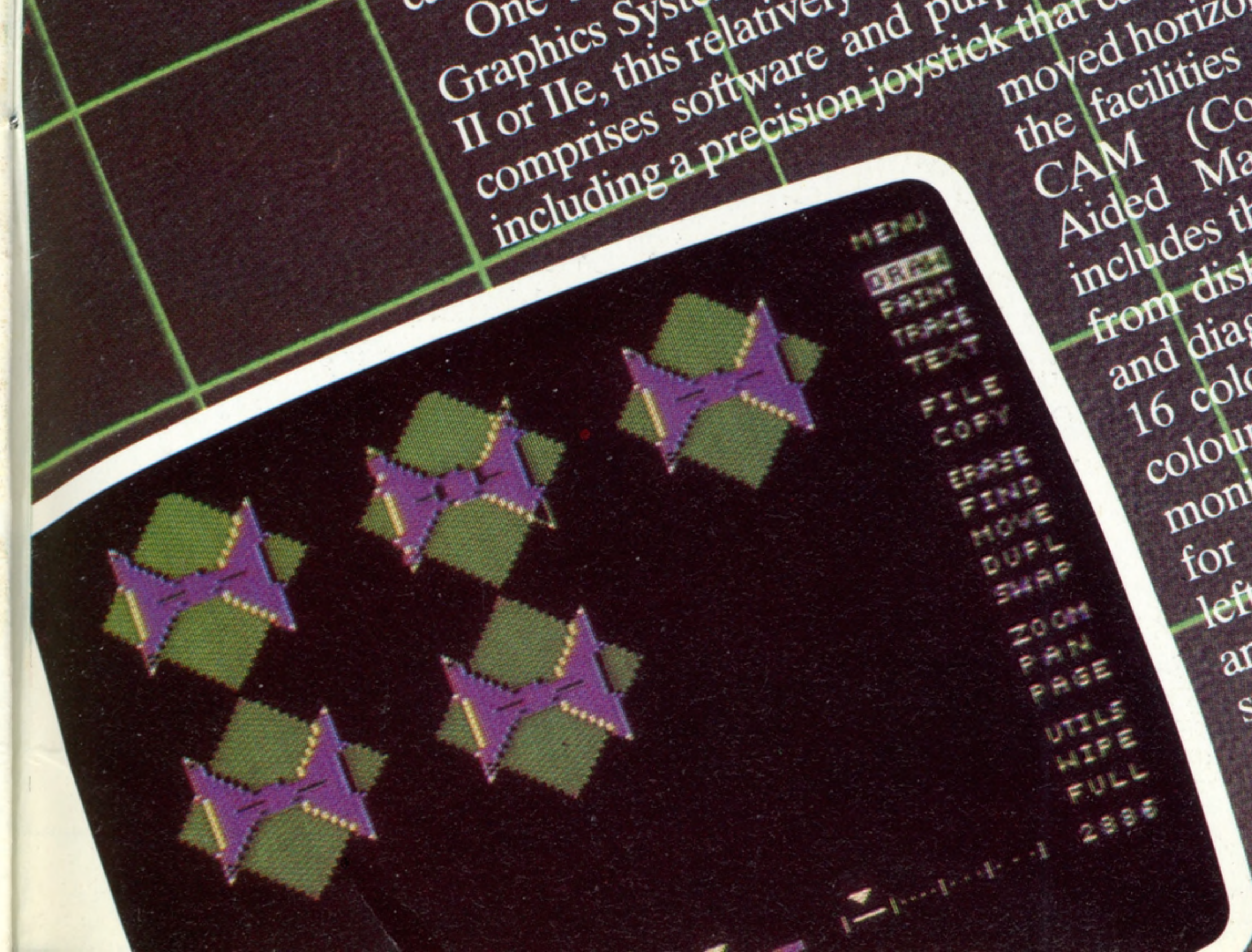




By far the most popular method of pointing to a particular location on the screen is by using a joystick or trackball (see page 56), but these peripherals were never conceived as precision devices. Sophisticated software, however, which manipulates individual dots or pixels on the screen can change the joystick or trackball into a remarkably accurate input device, capable of producing clear graphic representations. It enables the user to create a drawing in large scale on the screen and then reduce it down to the required size. What started as a games peripheral can become a powerful drafting tool.

One such device is the Robo 1000 Bit Stik Graphics System. Running on a 64 Kbyte Apple II or IIe, this relatively inexpensive system (£595) comprises software and purpose-built hardware including a precision joystick that can rotate or be moved horizontally or vertically. It offers many of the facilities found in mainframe-based CAD/CAM (Computer-Aided Design/Computer-Aided Manufacture) packages. The software includes the ability to: retrieve pre-drawn images from disk storage, and likewise to store pictures and diagrams for future use; fill areas with one of 16 colours — though of course you will need a colour card in your computer and a colour monitor to achieve this; magnify part of the image for drawing fine detail (called 'zooming'); pan left-right and up-down; and incorporate curves and circles. Given that the graphics resolution of a standard Apple is well below that of a professional CAD/CAM system, the results are surprisingly good and, perhaps best of all, the operating instructions are particularly clear and easy to follow.

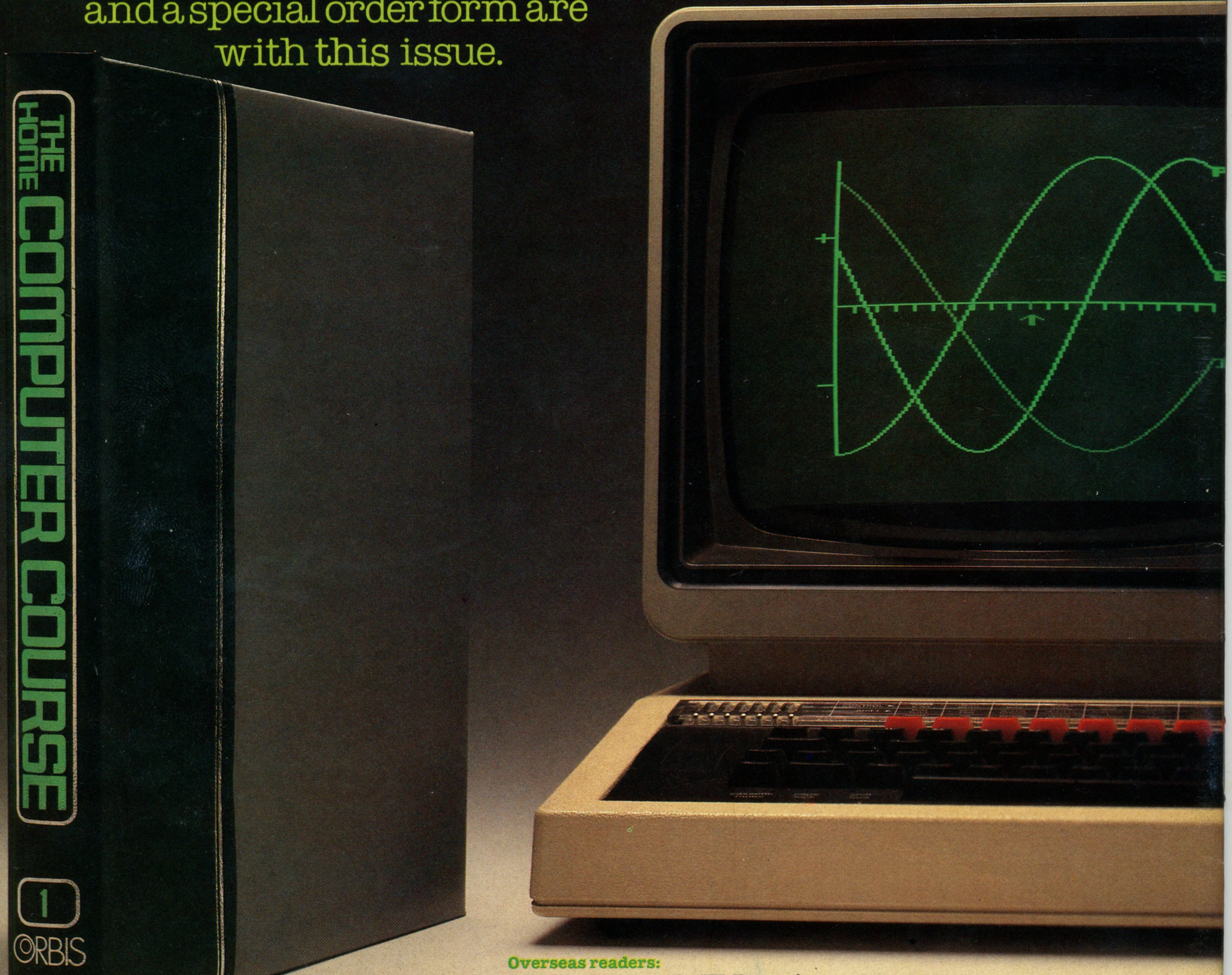




........THE HOME COMPUTER COURSE ....BINDER.

A BASIC NECESSITY.

To help you keep your copies immaculate we have made a very special binder offer. Details of this offer and a special order form are with this issue.



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.

NEXT TO YOUR COMPUTER..... YOUR COURSE MANUALS!