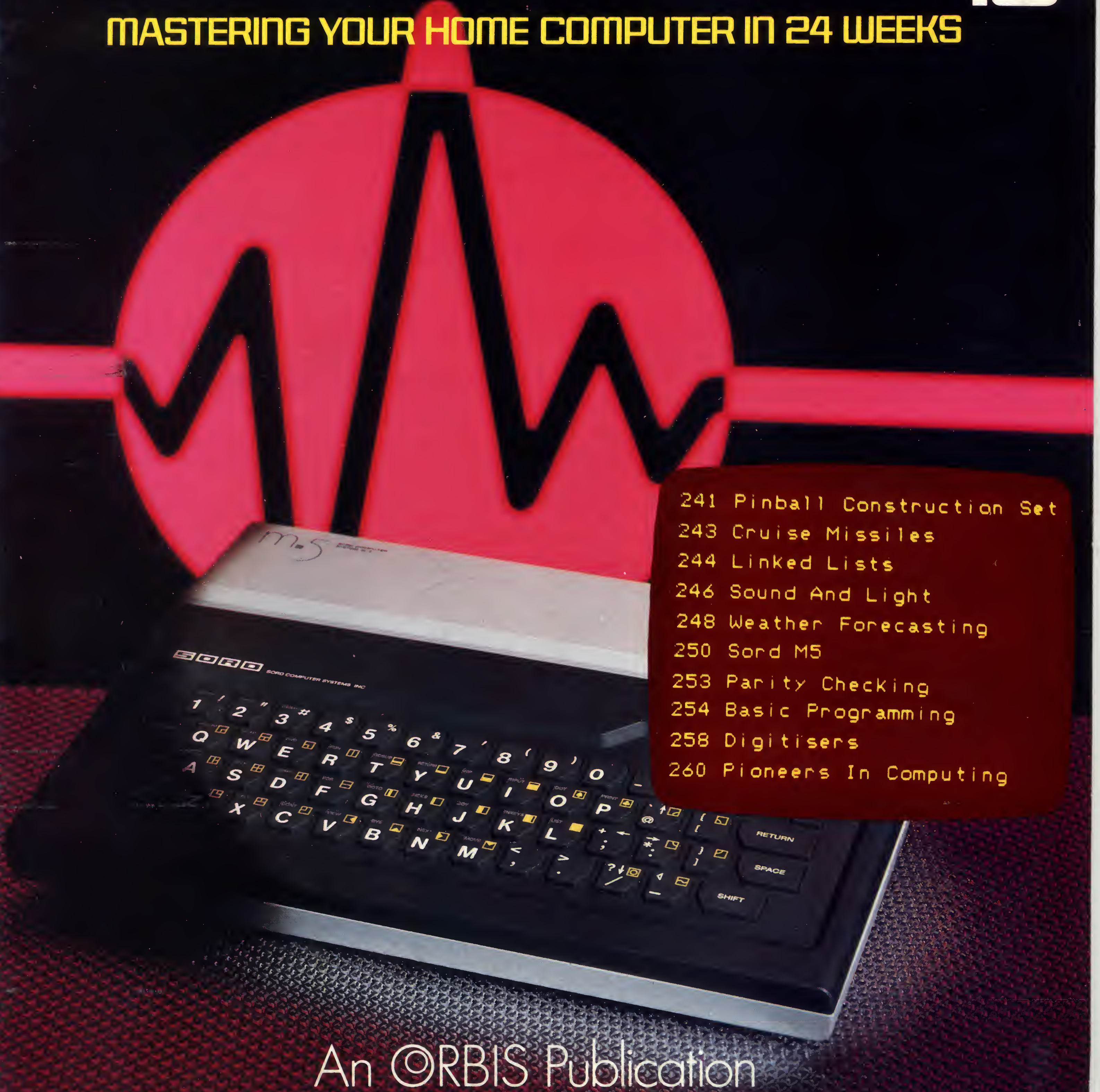
ISSN 0265-2919
R 80p

THE HOME COMPUTER COURSE

13

MASTERING YOUR HOME COMPUTER IN 24 WEEKS



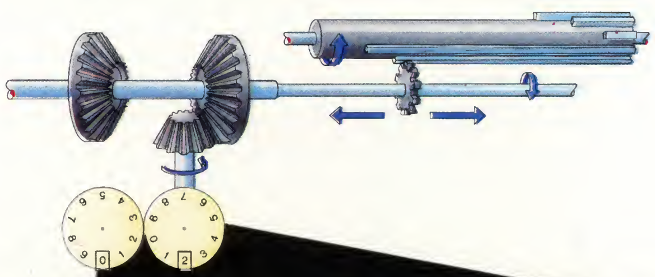
An ORBIS Publication

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

CONTENTS

Next Week

- We look at the Tandy Color Computer, a home computer that has been on the market for some time and is well supported by software and peripherals
- Apple's Lisa might just look like an expensive business microcomputer but its software will revolutionise computing, both in business and at home
- CRTs (Cathode Ray Tubes) form the heart of both televisions and monitors. However, there are other forms of display, and we'll be looking at how some of them work



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 Editors** Tracy Ebbetts, Robert Pickering; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Elizabeth Coley, Richard King; **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: 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.

Pinball Wizard

The Pinball Construction Set — a remarkable advance in software design — allows you to design and play your own pinball games on the screen of an Apple computer

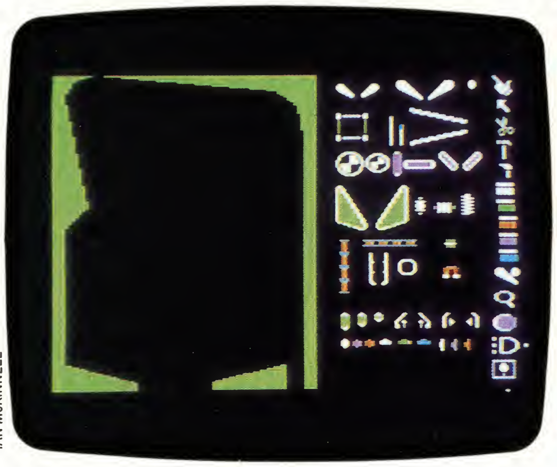
Even in the fast-developing microcomputer industry, where one can reasonably expect remarkable new developments to be quite commonplace, it is still a rare thing to come across a product that is radically different both in concept and quality. Such a piece of software is Budgeco's Pinball Construction Set (PCS). Running on a 48 Kbyte Apple II, with one disk drive and a joystick, this package performs an apparently simple function. It gives the user a picture of a bare pinball table, and a menu of 38 different types of 'furniture' that are used to equip it to the player's own design. There is, in addition, a functions menu from which to choose the tools you can use.

Having filled the table according to your plan — you are allowed to position upto 128 pieces on the table, but there is no limit to the number of times you may use any one type — all that remains is to play the game. You do this by selecting yet another function with the joystick. Up to four players may take turns, but each is allowed only one ball, instead of the three on most pinball machines, and there is no 'free ball' facility. At the end of the game, pressing ESCAPE gets you back to the menu. You are encouraged to go on developing the table after each game by the degree of feedback you get every time you play.

Both in its conception and execution, PCS points the way towards truly user-friendly software. As soon as the program is loaded (and this requires the user simply to insert the disk and press RETURN) virtually all the action is controlled from the joystick. The first tool to be used is a hand. It is moved so that it points to an object in the 'furniture' menu (such as a bumper or a flipper) and when you press the joystick button the hand 'picks up' the object indicated. The hand pulls it to its desired position on the table, and when you release the joystick button, the object is put firmly in place.

The interesting thing here is that you are moving not only the collection of data that defines the shape of the object, but also the set of rules that will govern the way it behaves when you come to play the game. A flipper, for example, always moves through 45 degrees, first up and then back down again. A bumper always repels the ball whilst accelerating it according to a definable 'kick' factor. The ball obeys the Newtonian laws of motion, and falls down the table according to the laws of gravity.

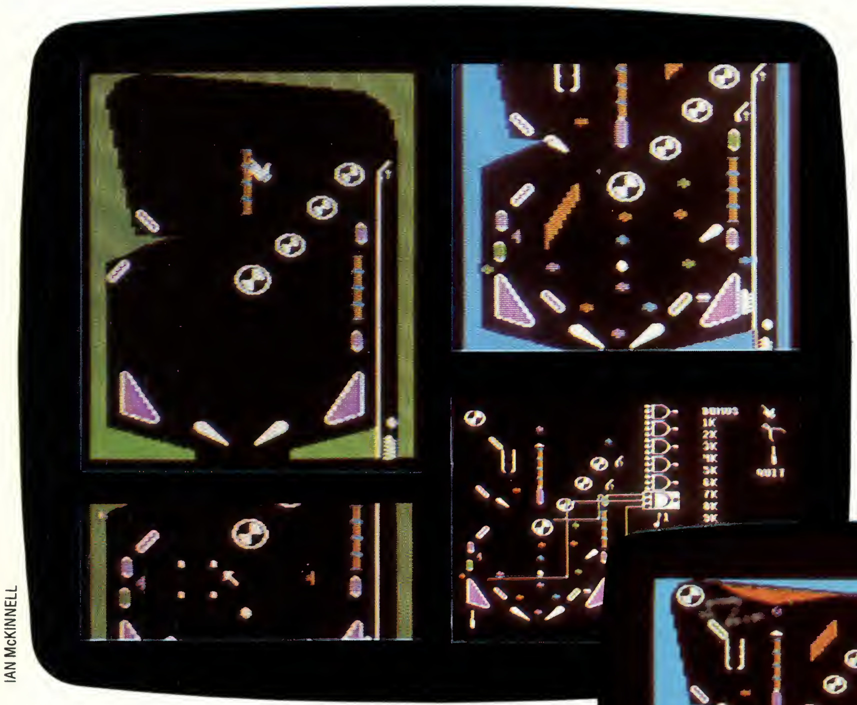
But having said all this, there is one tool (suitably given the symbol of a planet in partial sunlight) that allows you to alter the parameters of the real world — gravitational force, for example, or even time! This function is also controlled by the joystick. The position of each value on a scale is altered, just as one would move a slide-type


IAN McKINNELL

Do-It-Yourself Games
The Pinball Construction Set displays an empty table; a variety of types of 'furniture' — bumpers, targets, roll-overs, flippers and so on; and, in the column on the right, the tools for placing the objects on the table. This column also contains functions for adjusting the size, shape, colour and degree of interaction of the pieces, as well as for saving finished games on disk

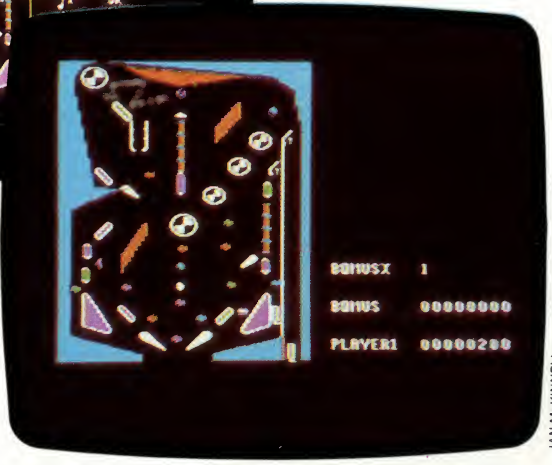
THE IMAGE MAKER

Kids' Stuff
PCS even gives you authentic sounds and the equivalent of flashing lights! But it's actually more fun to devise and build games than to play them. Now, if it had a TILT built into it . . .



IAN McKINNELL

Step By Step
These four pictures show various stages in the construction of a pinball game. First of all the basic pieces are installed, then a polygon is added to form a central island. The polygon is deformed and painted orange. Finally, some of the objects are tied together (by means of an AND gate) so that a bonus is scored when all three have been activated



IAN McKINNELL

Ready To Go
Once the game is composed on the table it can be saved on disk. Because all the operating functions 'travel' with the table, the original software package isn't needed to re-run the program

audio volume control by 'pushing' it up or 'pulling' it down.

All the other functions that one would expect in a well developed graphics package are also available. There are 'tools' for stretching and deforming lines by pulling them out between predetermined nodes (called 'rubber-banding'); for painting the blocks with one of the colours from the palette; and for magnifying small portions of the graphic image so that you can work in greater detail.

It is not so much the individual functions and capabilities of the Pinball Construction Set that are important, however, as its overall operating philosophy. Object oriented programming — where each operating element of the software package carries with it details of how it will work and how it interacts with any of the other objects or elements — lends itself to the production of programs that need very little computing experience or aptitude on the part of their users. This programming method will be used almost exclusively in the fifth generation of computers currently being developed. Object oriented programming is hailed as the most important breakthrough in the field of software science since high level languages were first introduced in the late fifties.

Most home computers have quite sufficient memory capacity and processing power for their user's needs. Any increase in that capacity and power is likely to be used to increase user friendliness. The truly remarkable thing about PCS is that it manages to achieve a high degree of user friendliness in only 48 Kbytes.

While object oriented programming applies itself readily to games and other graphics programs, it takes a little more programming ingenuity to introduce it into the field of business software. Though they do not use graphics as their main means of communication, spreadsheet packages (like Visicalc and Supercalc) are object oriented to a certain degree, in that each field or cell can contain both a piece of data and the relationships that define it.

Another example is Apple's Lisa system, which uses a 'mouse' to manoeuvre a pointer around the screen to select the program (represented by a graphic symbol) that you wish to run. The word processor, for example, is represented by a sheet of typing paper; the graph plotting program by a sheet of squared paper.

Perhaps the most fascinating of all its functions is the method Lisa uses to transfer data from one program to another. One of its 'Icons' (the name given to pictorial representations of functions on the screen) is a clipboard. If we wanted to take a small section of a spreadsheet and reproduce it as a graph, it is necessary only to define the window on the spreadsheet, transfer that window to the clipboard (which is a temporary storage area) and carry it across to the graph plotter program.

When we talked about arcade games (see page 221), we noted that there were a number of generically different types. PCS could well form a new category. It is tempting to suppose that the next step the home computer games industry will take will be the production of Maze and Chase Construction Sets, Space Invaders Construction Sets, and so on; at which point many games program writers could find themselves redundant.

CHRIS STEVENS

Objective Outlook
As well as being an intriguing and educational game, the Pinball Construction Set is a fine example of object oriented programming. In normal programming, the structure of the data is defined, and then program routines are written to manipulate this. In object oriented programming, the calculations and procedures are inseparable from the data. In the pinball program, moving the symbol for a pinball machine's flipper onto the board not only sets up the data (in this case, the shape of the flipper), but arranges for the associated routines to be set up to activate the flipper.

Object oriented programming lends itself to visual applications. Spreadsheets are another example: the field that displays a result will also contain the formula to get that result.

The current trend for business workstations that simulate the layout of items on a desktop also derives from the same idea. Pointing to an image of a piece of typing paper on the screen activates the word processor, whilst pointing to a miniature drawing of a filing cabinet will file the results away

Cruise Control

Cruise missiles are a controversial subject, but they contain some interesting computer technology — such as bubble memory — which will soon be appearing in home computers

When Neil Armstrong took his one small step onto the surface of the moon, it was largely due to computerised guidance systems. Of course, interplanetary rocketry relies on very precise engineering, but without computer hardware and software it would never be possible to perform positional calculations either fast enough, or with sufficient accuracy, to allow one object to engage with another at a vast distance — even an object as big as the moon.

When one considers current military requirements that call for the placement of warheads to within 20 or 30 metres (70 to 100 feet) after a flight across a continent, then the scope of data processing power needed to perform the calculations becomes enormous.

Early military experience showed that the fundamental problem with missiles was that once fired, no correction was possible. The first major advance came with the development of simple guidance systems that were able to judge where the rocket was in relation to a point on the earth's surface (the launch site) by deducing how far it had travelled, and in what direction. But even a first-class modern system of this type will be prone to significant error.

Another, and more accurate, method uses satellites in geo-stationary orbit as reference points. The main drawback to these systems is that the flightpath of the missile — and probably its target — are deducible by the enemy very soon after launch, given the capability of modern over-the-horizon radar systems. To combat this vulnerability, the ideal military requirement was for a low-flying missile with a small radar cross-section that could actually decide for itself the course it would fly to its target. And so the Cruise missile was born.

The Cruise missile constantly updates its position by analysing the contours of the ground over which it is flying. This is done by matching a succession of height-above-ground readings, from an extremely accurate radar altimeter, with a contour map of the terrain stored in an on-board bubble memory.

This system, developed by McDonnell Douglas, is known as TERCOM (TERrain COntour Matching), or DPW-23. Each missile has stored in its bubble memory some 25 'route profiles' that it compares with the terrain it is passing over. However, there are drawbacks to this. For example, the system is not usable over water as that has no permanent features. It is also not reliably accurate over sand desert, where the terrain is in constant motion. Neither, one suspects, is it accurate in the depths of a North European winter, when the terrain will be significantly altered by the large seasonal snowfalls.

Cruise does not use this guidance system from the moment of launch. It remains inertial while the missile flies at altitude in friendly airspace. Once it is vulnerable to attack from the air or the ground, it dives to within 15m (50 ft) of the ground for its flight over enemy territory. Even though it may be up to a kilometre (1,100 yds) off course at this point, it is predicted that it will be sufficiently close to one of its 25 mapped routes to be able to relocate itself precisely.

When the missile nears its target it turns on a Terminal Correlator Unit which contains — once again in bubble memory — a detailed digital picture of the target area as it would be seen from an on-coming missile. Tests have shown that this system is likely to be accurate to within 18m (60 ft), after a flight of some 2,800km (1,750 miles).

COURTESY OF AEROSPACE PUBLISHING LTD

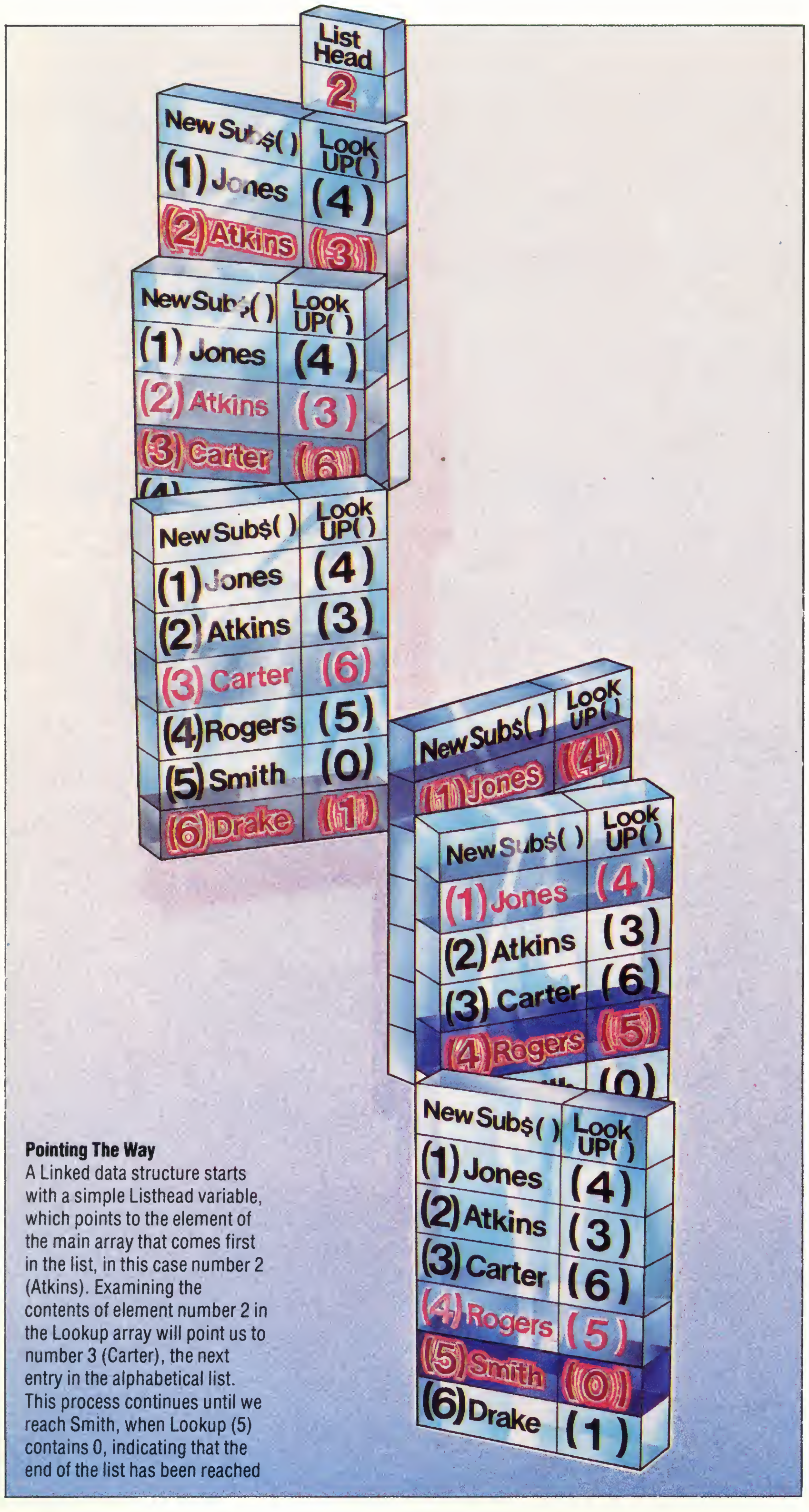
Self-Seeking Missile
The General Dynamics 'Tomahawk' Ground Launched Cruise Missile is 6.40m (21ft) long, and weighs less than one and a quarter tons (1,200kg). Fired from a tube mounted on a mobile launcher, it starts life as a conventional rocket, but soon deploys small wings and settles down to low-level flight powered by a remarkably small and compact turbo-fan jet engine

COURTESY OF NEW SCIENTIST

Bubble, Bubble
In bubble memories, 'bubbles' of magnetic force are created to form a '1', and not created to represent '0', on a tiny chip of garnet. The advantages are the packing density — currently one million bits, or 128 Kbytes per chip — and no loss of contents when the power is turned off. However, bubble memories react considerably more slowly than conventional RAM

Chain Mail

Indexing is one way of structuring large quantities of data, such as names and addresses. The Linked List or chain is an alternative with distinct advantages



Pointing The Way
A Linked data structure starts with a simple Listhead variable, which points to the element of the main array that comes first in the list, in this case number 2 (Atkins). Examining the contents of element number 2 in the Lookup array will point us to number 3 (Carter), the next entry in the alphabetical list. This process continues until we reach Smith, when Lookup (5) contains 0, indicating that the end of the list has been reached

TONY LODGE

In a computer's memory there is only data, byte after byte of it, stored in thousands of voltage patterns. Meaning is given to those bytes by the data structure that the central processor imposes. Those various data structures decide whether any particular byte is interpreted as part of an instruction, or as digits belonging to a larger number, or as a character code.

From the user's point of view some kinds of data structure are virtually wired into computers. Programming languages usually demand that data be structured in a limited number of ways. BASIC imposes the idea of numeric and string data types, and supplies variables and array structures for manipulating those types. Other languages usually support those and additional structures. The strength and variety of its data types are major components of a language's power.

The BASIC data structures — variables and arrays — will be all that we need to simulate some other ways of looking at data.

The indexed array is a useful data structure, and easily implemented in BASIC. It has its limitations, however, particularly when the data to which it refers is likely to change often and/or unpredictably.

Suppose British Telecom keeps a file of its new subscribers for eventual inclusion in the next issue of the telephone directory. Until that time, the names and addresses have to be kept in alphabetic order for easy reference, but the file is constantly growing, and the additions arrive unpredictably. On Monday the file NewSub$ ( ) might look like this when it's read into the array:

NewSub$ ( )	Index ( )
(1) Jones	(2)
(2) Atkins	(3)
(3) Carter	(6)
(4) Rogers	(1)
(5) Smith	(4)
(6) Drake	(5)

The array Index ( ) shows the order in which to read NewSub$ ( ) so that the entries are in alphabetic order. Thus, the first item alphabetically is NewSub$ (2), Atkins. The second item is NewSub$ (3), Carter. In this example only the names are shown, but in fact a directory entry comprises name, initials, and address — typically about 60 characters. Moving blocks of 60 characters around in memory is slow (as sorting requires many data moves) and wastes memory, so it is more efficient to leave NewSub$ ( ) unsorted, and create Index ( ) instead. Now a new name, Bull, has to be added to the file, so the arrays look like this:

NewSub$ ( )	Index ( )
(1) Jones	(2)
(2) Atkins	(7)
(3) Carter	(3)
(4) Rogers	(6)
(5) Smith	(1)
(6) Drake	(4)
(7) Bull	(5)

Notice that the contents of Index ( ) above the new insertion are unchanged, and its contents below the insertion are in the same order as previously, but have all been moved one place down in the array. Insertion to an index therefore requires: finding the position of the new element, moving every element between there and the end of the index down one, and writing in the new entry. This is preferable to doing the same thing with the actual data, NewSub$, but is still relatively slow, if the index is large.

Suppose, now, that we structure the data in a different way. Leave NewSub$ ( ) unsorted because manipulating it is slow and expensive, and establish a parallel array called LookUp ( ), whose contents are simply numbers referring to positions in NewSub$ ( ).

ListHead (2)

NewSub$ ( )	LookUp ( )	Index ( )
(1) Jones	(4)	(2)
(2) Atkins	(3)	(3)
(3) Carter	(6)	(6)
(4) Rogers	(5)	(1)
(5) Smith	(0)	(4)
(6) Drake	(1)	(5)

The first difference is that a simple variable called ListHead is needed: it points to NewSub$ (2) which is alphabetically the first element of NewSub$ ( ).The next difference is that the number (0) has been used in LookUp (5): this indicates that NewSub$ (5) is alphabetically the last element of the array.

The next difference is the contents of Index ( ) and LookUp ( ). Index ( ) has to be read: 'the first element is in NewSub$ (2), the second is in NewSub$ (3), the third is in NewSub$ (6)'...etc. while ListHead ( ) is read: 'the first element is in NewSub$ (2); Then LookUp (2) says that the next element is in NewSub$ (3); LookUp (3) says that the next element is in NewSub$ (6); and so on. LookUp (5) says that NewSub$ (5) is the last element.

Index ( ) gives an absolute position for elements of the file, while LookUp ( ) gives only relative positions — any item in LookUp ( ) tells you only where to find the next element, and says nothing about absolute position. The number in Index (4) points to the fourth item in the alphabetically ordered file, whereas the number in LookUp (4) points only to the item that comes after NewSub$ (4) in the ordered file. LookUp ( ) implements the data structure called a 'Linked List'. Reading a Linked List is like following a treasure hunt: at the start you're told your first destination; when you get there you find a clue which points you to your next destination, and so on. Reading an Indexed Array is like being on a car rally: at the start you're told all your destinations and the order in which to visit them.

The great advantage of the List structure is its flexibility. Consider the List after insertion of the new element, Bull:

ListHead (2)

NewSub$ ( )	LookUp ( )
(1) Jones	(4)
(2) Atkins	(7)
(3) Carter	(6)
(4) Rogers	(5)
(5) Smith	(0)
(6) Drake	(1)
(7) Bull	(3)

The array LookUp ( ) has changed in only two places:

i) LookUp (2), which formerly pointed to NewSub$ (3) as containing the next alphabetic element after NewSub$ (2), now points to NewSub$ (7) since it is now the next alphabetic element after NewSub$ (2)

ii) LookUp (7), which was unused, now points to NewSub$(3) as the next item after NewSub$ (7) in the alphabetic ordering.

This illustrates the general process of insertion to a Linked List: find the element of the list which should come just before the new element, and make that element point to the new element; then make the new element point to the element that it has displaced. These simple operations will be all that is required for insertion to a Linked List, and only the first of these is affected by the size of the List. Inserting an element to a List is like inserting a new link into a chain — decide where to put the link, break the chain, join the preceding link to the new one, and the new link to the succeeding link. Linked Lists are sometimes called Chained Lists. The numbers in LookUp ( ) — the links — are sometimes called Pointers.

A striking feature of Lists is their strong seriality; it is impossible to find an element in a List except by starting at the beginning and inspecting every element until the target is found. The List is implemented here by using arrays, which are designed to be Direct Access structures, but the List has effectively turned them into Sequential Files. In other languages, such as LISP and PASCAL, the List facility is built-in.

Lists are useful structures for handling dynamic data (data that regularly changes), and can be powerful tools when dealing with either natural language (as in speech recognition) or artificial language (when compiling programs), where the data itself naturally forms a list of elements.

Introducing Sound

Sound And Light is a new series that will teach you how to get the most from the sound and graphics facilities on your computer

As home computers have developed over the last few years the features provided have become more comprehensive. Games facilities have been of vital importance to the popularity of each new computer and much time and effort has gone into developing sophisticated colour graphics capabilities. Though not so obvious in importance, sound and music-making features have been developed to a similar degree. If you asked successful games writers how important sound routines were in their programs they would probably place them a close third behind the game concept and graphics. Intelligent use of sound effects and music add considerably to the excitement and entertainment value of all arcade-type games.

In addition to games applications it is possible to further your knowledge of music by using the sound capabilities provided by your home computer. In many cases special music commands are provided in BASIC to enable you to write short programs to play quite complex tunes that even include chords. Some computers also provide ways to change the nature of the sound to make it more pleasing to the ear or approximate the sounds of conventional musical instruments. In all cases the computer keyboard can be configured, by means of a suitable program, to act in a similar manner to a piano keyboard, enabling you to play music in 'real time'.

Even if you have little knowledge of programming it is possible to write short and simple programs that make reasonably sophisticated musical sounds. If you wish to use the sound facilities to their best advantage, most software houses produce comprehensive music programs that enable you to write and play tunes immediately. Whichever approach you take, it is useful to understand how your computer generates, shapes and controls its sound output.

... And Light

Low and High Resolution

Graphics on microcomputers can be divided into two categories: low resolution and high resolution. The difference between low and high resolution is best described by considering how a character (a letter, number or shape) is made up.

If you take a close look at a standard character printed on a television screen you can see that its shape is made up of a group of small squares. These squares are called picture elements, or 'pixels', and every character or shape that appears on the screen is an arrangement of these in a pattern. On most home computers the characters are formed from a square of 64 pixels, grouped into eight rows of eight. The letter 'A' can be made up of a pixel pattern like this:

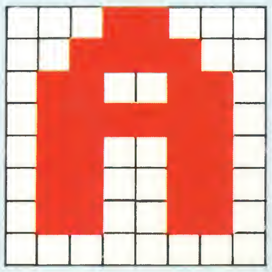
Each illuminated pixel on the grid can be represented in the computer's memory by a '1' and each dark pixel by a '0'. Eight bits make a byte, so each row of the character grid may be stored in one single location of the computer's memory. Thus it takes eight memory locations to hold a single character.

Graphic displays are sometimes made up of blocks the size of whole, half, or quarter character grids. Graphics designed using these large, simple building blocks are said to be of low resolution. On many home computers it is now possible to design graphic displays that are built up from single pixels. These are high resolution displays. A good way to demonstrate the difference between the two types is to look at a plot of a sine curve, as illustrated, using both kinds of resolution.

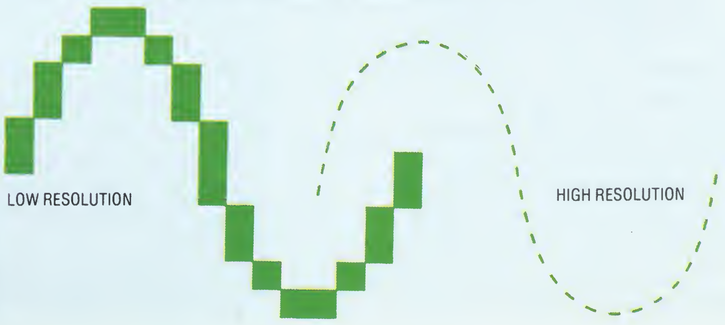
BIT PATTERN

```
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```

PIXEL PATTERN



LIZ DIXON



Oscillators

Oscillators are electronic circuits that produce repetitive signals. When these signals are amplified and fed to a speaker they make sounds of a given pitch. The number of oscillators provided by home computers varies between one and four — the more oscillators you have the more notes you can play at once.

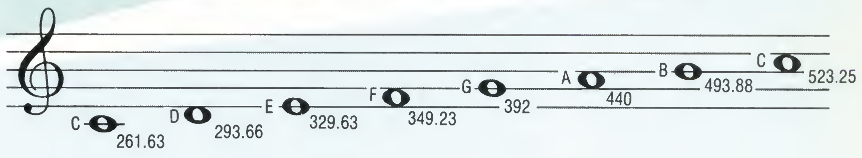
Three characteristics describe the sound created: frequency, envelope (which includes volume) and waveform. Frequency will be introduced in this instalment and envelope generators and waveform dealt with in the second.

Frequency

This is the most important characteristic that we need to control, as it determines the pitch of the sound. Frequency is the number of times a signal repeats itself every second and is measured in hertz (Hz, cycles per second). Sounds that can be heard by the human ear have frequencies greater than 20Hz but less than about 20,000Hz. Although we cannot hear frequencies below 20Hz they can be used to modify the characteristics of an audible sound. This technique is called modulation and at present can be applied only on the Commodore 64 among home computers.

However, it is not necessary to delve deeply into frequencies. What you really need to know is how to play musical notes. The ease with which you can do this varies enormously from one machine to another. Some have BASIC commands that work out the frequencies for you so that you need only specify a pitch number or even a musical letter symbol — A, A#, B, and so on. Others make it much more difficult by providing only a table in the user manual where you look up the frequency corresponding to the required note and POKE the frequency value into a memory location. The table shows accurate conversions for the scale of middle C. It will also be useful for those wishing to program music in machine code, where BASIC is unable to help you calculate the frequencies.

Music Notes To Frequencies
You can work out the frequency of each note in the scale by multiplying the frequency of the note one semitone below it by 1.0594631. This may appear a little baffling but if the multiplication is carried out 12 times the original frequency is doubled. There are 12 semitones in an octave (the difference between two notes with the same letter) so doubling the frequency moves the sound up one octave. This table provides accurate conversions from music note symbols (for the scale of middle C) to frequencies



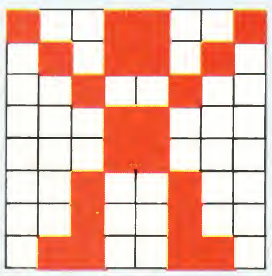
User-Defined Characters

To create unusual and attractive screen displays it is often useful to have characters available that are different from the normal alphanumeric character set. The Vic-20 and Commodore 64 have a special set of graphic characters that can be used directly from the keyboard, but even these do not cover every eventuality. On most home computers it is possible to create new characters. This is usually achieved by redefining the binary patterns of the eight locations of memory in which a character is stored. In the process the old set of binary patterns is often lost, or 'overwritten', and the 'user-defined' character takes on some of the properties of the one it has replaced in memory. Thus the new character can be used in PRINT statements by simply pressing the key of the character it has replaced. Here is an example of a user-defined character, together with its associated binary codes:

BIT PATTERN

128	64	32	16	8	4	2	1
1	0	0	1	1	0	0	1
0	1	0	1	1	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	1	1	0	0	1	1	0

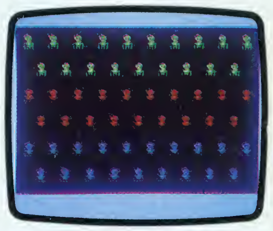
PIXEL PATTERN



The ease with which user-defined characters can be set up varies greatly according to the computer being used. For example, with the Sinclair Spectrum's USR command, all that is required is to enter the appropriate binary patterns; whereas on the Commodore 64 the user first has to move the complete character set from ROM to RAM before POKEing in to memory the eight decimal equivalents of the bit patterns that make up the shape. However, several character-designing utility programs, available from independent suppliers, make life easier for the Commodore 64 owner.

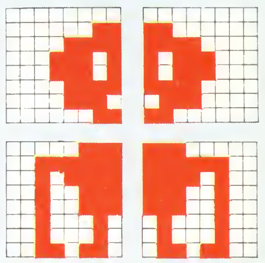
To create larger figures it is possible to group two or more user-defined characters together. The alien figures shown (right) were constructed from four user-defined characters. The program, which runs on the Commodore 64, PRINTs the character groups on the screen in three different colours. The characters were created by using a short routine to move the normal character set from ROM to RAM and replace the graphics characters ▞ , ◨ , ◪ , and ' ' by reading in decimal numbers from DATA statements and using POKE commands to place them in the appropriate locations. Full details of how you can do this will be given in a forthcoming instalment.

Even when sprites (see page 152) are available there is often a limit to the number that can be displayed at any one time on the screen, so user-defined graphics come in useful where many similar shapes need to be displayed at the same time.



IAN McKINNELL

Extra Terrestrial
These alien creatures were created from four characters, each defined by the programmer. This method can be used on many machines that don't have sprites



Brighter Outlook

Use of high-speed computers, both to process satellite images and analyse patterns of data, has made weather forecasts a great deal more accurate than they used to be

The results of many of the most complex data processing tasks are present in our everyday lives, often without us knowing about them. One of the most advanced computer applications, requiring greater data processing capacity than almost any other in the country, gives us daily information about our weather conditions and what we can expect from them. Given the complexity of weather forecasting, it is perhaps surprising that our forecasters come up with the right answers as often as they do. Computer aided prediction is an immense asset to them in dealing with the vast array of possibilities.

The climatographic factors that affect the weather patterns over the British Isles, and to a lesser extent the Atlantic seaboard of the European landmass, are extremely complex. Primarily, they are conditioned by our proximity to both the North Pole and the Atlantic Ocean. Being situated on the eastern side of the Atlantic, we are more prone to the climatic effects created within its 2,500 mile width, because of the 'Coriolis effect'. This phenomenon is due to the earth's west-to-east spin. It is best understood if we remember that at the equator an object on the earth's surface is travelling at more than 1,600 kilometres per hour (1,000 mph); and this powerful spinning motion, combined with the normal pole-to-equator wind patterns, creates the prevailing westerlies (winds that originate in the west) in the Northern Hemisphere. It is this constant onslaught of wet air — rising and falling according to local variations in temperature — that causes the predominant weather conditions in Britain.

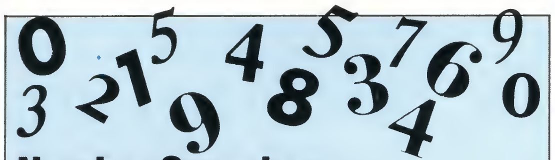
Weather forecasters in the United Kingdom rely primarily on observations from data collection stations spaced at strategic locations in the Atlantic — weather ships, buoys, balloons and patrolling aircraft — to provide them with information about approaching conditions. They

Pictures From Space
The Meteosat 2 weather satellite, launched in June 1981, is in a geostationary orbit (that is, it does not move in relation to the earth) some 35,880 Km (22,300 miles) above the equator, on the zero meridian. It gathers information from a large number of earth stations

then predict what will happen as these climatic phenomena approach the land mass, according to the known behaviour of similar phenomena in the past.

Before March 1979, when the Meteosat 1 weather satellite was launched, the only method of prediction available to forecasters was to plot reports from the weather stations onto a map to build up an isobaric chart. Isobars are imaginary lines that join points of equal barometric pressure, rather as contour lines on a map join points of equal height. From these it is possible to decide on the speed and direction of warm and cold fronts — and their associated cyclones and anticyclones — and thus make what are best described as educated guesses about the expected weather conditions.

While isobaric charts are by far the most common, they are by no means the only maps that the Meteorological Office produces. From the vast weather database held in its computer system it can produce charts that show average temperature, rainfalls, hours of sunshine per day, and so on.



Number Crunchers

One of the chief uses of large computers in scientific research is to process purely numerical information in the form of very large and complex equations. Pure science applications such as nuclear physics, and applied science applications such as meteorology have similar requirements. While one could perform calculations of this complexity on a home micro, the length of time taken would be prohibitive — as a result not only of the number of terms in the equation, but also of the sheer magnitude of the numbers involved, which can go to 30 or more decimal places. In order to perform this function in a reasonable time, one needs very fast computers with very large amounts of memory

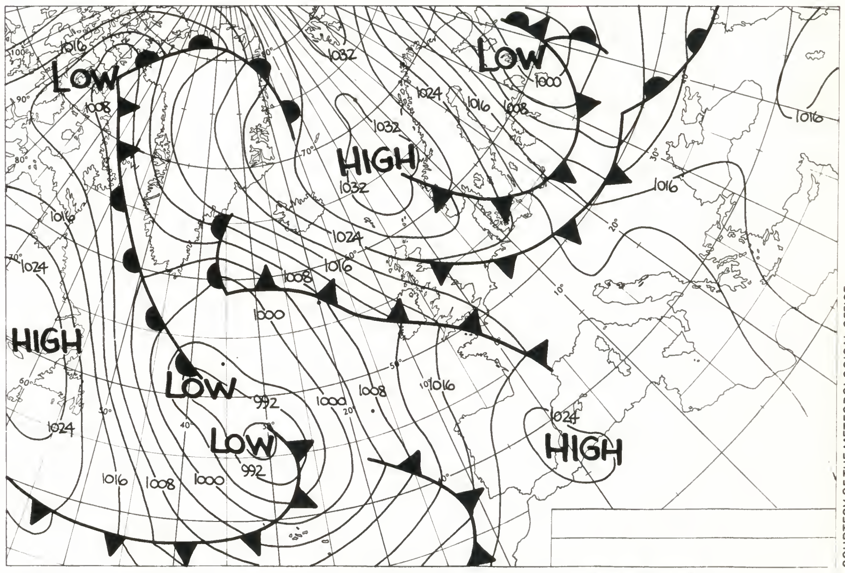
The Meteorological Office still follows this procedure for its accurate charts of current conditions, but now also uses the images received from Meteosat. These are analogue signals which are digitised for processing and display by the computer in the form of artificially coloured maps. The images create a live picture of the weather pattern as it occurs. They are regenerated approximately every four minutes, so the forecaster is able to observe the creation of weather systems in real time.

Meteosat 2, which replaced the earlier satellite in June 1981, sits in a geostationary orbit some 35,880 km (22,300 miles) above the Equator. It gathers data from a large number of earth stations spread out across the surface of the globe, and relays that information to anyone who wishes to subscribe to the system.

It would be theoretically possible to analyse and interpret this information (though not in real time) on a home computer by writing the received data to disk as it arrives from the satellite. However, the signal is an analogue one, so the conversion might be difficult. You would also need to install your own dish aerial precisely aligned with the satellite. The processing of these satellite images is only one very small function of the Meteorological Office's computer system. Along with other similar organisations in other parts of the world, it maintains a global weather system model and extracts from this model a vast amount of statistical data. This forms the database of historical information from which trends in global and local climate are plotted. It includes not only barometric data, but also details of wind speed and direction, rainfall, and temperature — not just at sea or ground level but also at specific altitudes.

Collection of this data is important for historical analysis. It is vital to agriculture, to many industries, and to the economy and ecology of whole continents, for it is only by this means that changes in climate can be recognised. Examples of this include the results of the progressive destruction of the Amazon rain forest and the increase in size of the polar ice-caps that could indicate the approach of another ice age.

Earth Stations
Satellite receiving aerials (known as dish aerials, after their shape) can vary immensely in size and complexity. The one shown here is capable of both receiving and transmitting, and is not confined to signals from geostationary satellites. It has sophisticated computer control that allows it to track an orbiting satellite precisely



COURTESY OF THE METEOROLOGICAL OFFICE

Isobaric Charts
The 'weather maps' that we see on television or in our newspapers are actually charts of barometric pressure. The concentric lines join points of equal air pressure. Winds flow anti-clockwise around a 'low', clockwise around a 'high' (the reverse in the southern hemisphere), and wind speed is directly related to the distance between the isobars

Sord M5

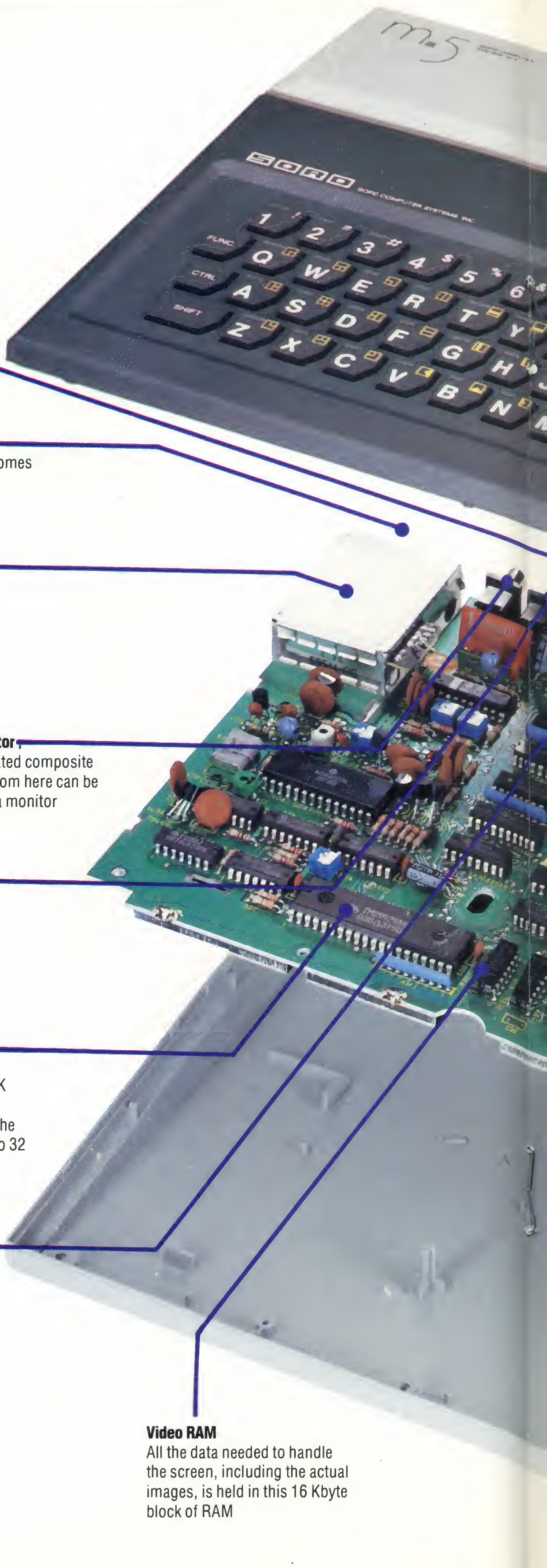
Though this machine features only four Kbytes of user memory as standard, its superb graphics facilities mean that the user can still write worthwhile programs

Most of the early home computers were designed in California, USA. More recently, British-designed machines have started to capture a large share of the worldwide market. However, it can only be a matter of time before the Japanese dominate the scene, as they have done in every other consumer electronic market. The Sord M5 is certainly not the first Japanese microcomputer, but it is the first to have made a significant impact on the home, as distinct from the business market.

It is a solid and compact machine similar in size to the Sinclair Spectrum, but is considerably heavier and feels much more robust. In many other respects it has similar capacities, with a Z80A CPU, single-key entry for BASIC, and program/data storage on cassette. Internally, however, it's much more sophisticated, as witnessed by the built-in Centronics printer port. But the two major differences are the size of the RAM memory — which at four Kbytes (expandable to 36 Kbytes) is much smaller in the unexpanded machine — and the inclusion of dedicated graphics and sound chips.

The graphics are handled by a TI 9918, 9928 or 9929 (depending on the country in which the computer is sold), which gives a resolution of 192 × 256 dots in up to 16 different colours. There are four main graphic modes, three of which may have up to 32 independently moving sprites, which can

The ROM Cartridge
One of the best features of the M5 is that the language can be changed because it is kept in a ROM cartridge. Three versions of BASIC are available for the M5: BASIC-I (simple, for beginners); BASIC-G (very strong on graphics); and BASIC-F (scientific and mathematical). There is also a special user-oriented, general-purpose program called FALC, which has a combination of spreadsheet, filing and graphics functions, and can be used to develop sophisticated applications for home or business use



Printer Connector
A Centronics compatible parallel printer interface is available at this socket, allowing many widely available printers to be directly connected to the M5

RF Connector
TV compatible output comes out of here

Modulator
The output from the VDP is converted into a standard TV signal

Video Connector
The unmodulated composite video signal from here can be used to drive a monitor

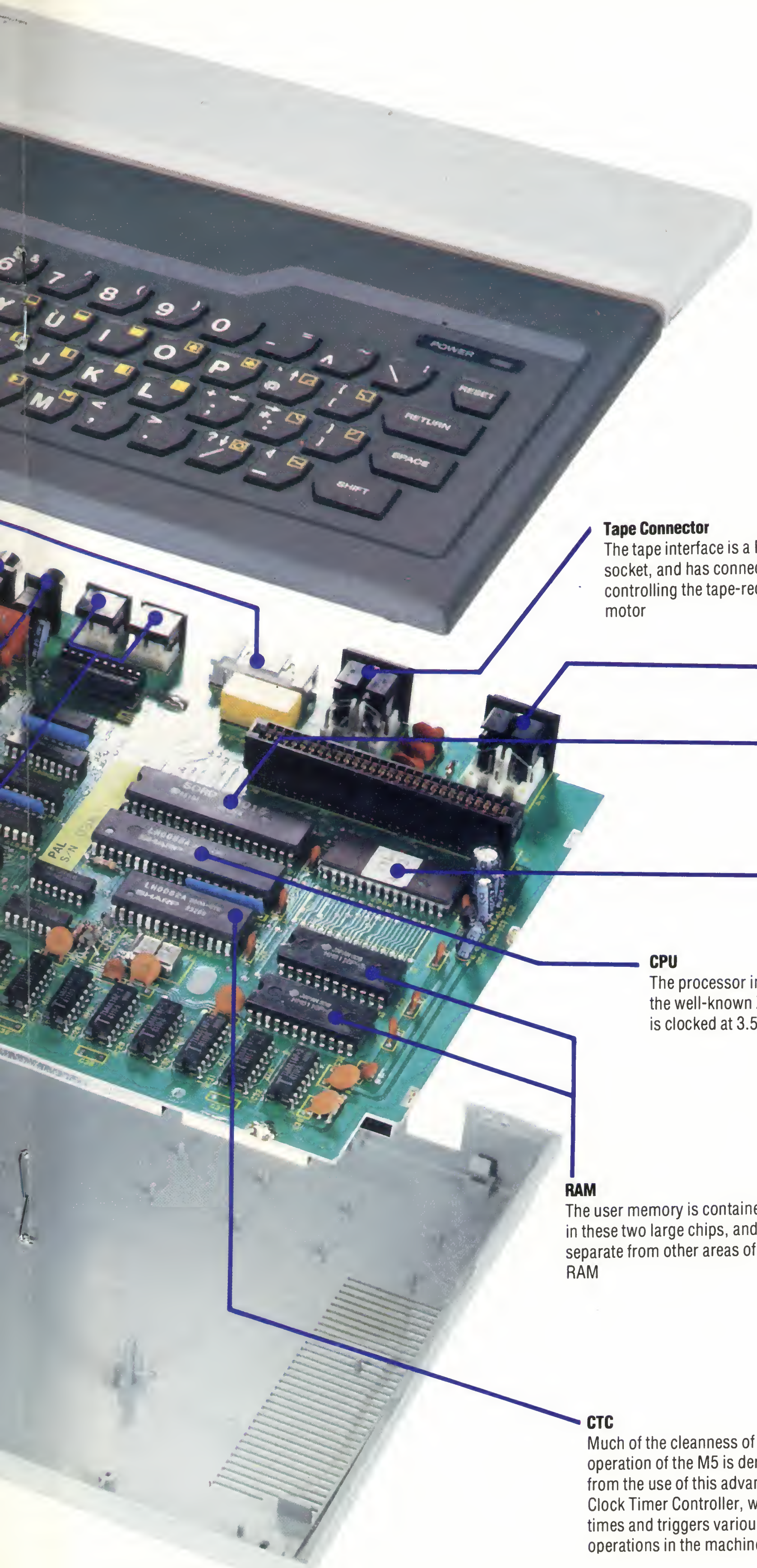
Audio Connector
The audio output can be fed into an amplifier from this socket

VDP
The Texas TMS 9929 Video Display Processor (in the UK version of the M5) is responsible for controlling the screen, and can handle up to 32 separate sprites

Joypad Connectors
The two Joypads plug in here, for games playing

Video RAM
All the data needed to handle the screen, including the actual images, is held in this 16 Kbyte block of RAM

The Joypads
The joypads are the Sord equivalent of joysticks. They work by sending a signal for each of four diagonal directions. Since these signals actually interrupt the CPU, no matter what task it is executing, the response time is very fast indeed



Tape Connector
The tape interface is a DIN-type socket, and has connections for controlling the tape-recorder motor

Power Connector
Power is supplied here from a small transformer

Custom Chip
The M5 uses a piece of sophisticated custom logic to achieve its advanced functions at a reasonable price

ROM
The only built-in programs in the machine are a set of low-level control programs, which are called up by the user program. These take care of the details of handling the screen, keyboard and cassette

CPU
The processor in the Sord M5 is the well-known Z80A. This one is clocked at 3.58MHz

RAM
The user memory is contained in these two large chips, and is separate from other areas of RAM

CTC
Much of the cleanness of operation of the M5 is derived from the use of this advanced Clock Timer Controller, which times and triggers various operations in the machine

be standard-sized or enhanced. The machine can display upper and lower case letters, punctuation and numbers. It has line and block drawing symbols, as well as a very large range of accented lower case letters for use with foreign languages — and since any character can be redefined, the possibilities are very wide indeed.

Other machines use the same graphics chips — in particular the TI99/4A (see page 189) — and it is the use of such dedicated chips that makes the Sord M5 so effective despite its lack of RAM. Since the screen memory is totally separate from the program memory, the only contents of the main RAM will be the actual program, plus, of course, the data needed by the variables.

Something that is currently being hotly argued over in the home computer industry is the

SORD M5

PRICE	£145
SIZE	185 × 70 × 55mm
WEIGHT	1kg
CPU	Z80A
CLOCK SPEED	3.58MHz
MEMORY	8 Kbytes ROM 20 Kbytes RAM, of which 16 Kbytes are used for graphic display. With the addition of cartridges the ROM can be expanded to 16 Kbytes and the RAM by 32 Kbytes
VIDEO DISPLAY	Up to 16 colours, which can be used on different 'planes'. There are sprite graphics and four different screen modes: two graphic, one text and a 'multi-colour' mode
INTERFACES	Cassette, printer (Centronics), joypads, ROM cartridge, audio
LANGUAGE SUPPLIED	Language cartridge is integer Basic, BASIC-I
OTHER LANGUAGES AVAILABLE	BASIC-G (graphics), BASIC-F (floating-point BASIC), FALC (a spreadsheet and database language)
COMES WITH	Power supply adaptor, cassette leads, television lead, two joysticks with leads, BASIC-I cartridge and a cassette with two games
KEYBOARD	55 keys: eight shifts giving all alphanumeric characters, 28 BASIC statements, and 64 graphic patterns
DOCUMENTATION	There is an 18-page User Guide that describes how to connect up the computer, how to load and play the two games, with a page dedicated to simple fault-finding. There is no description of the BASIC language or of using the cassette or other interfaces for any other purposes than for playing the games supplied

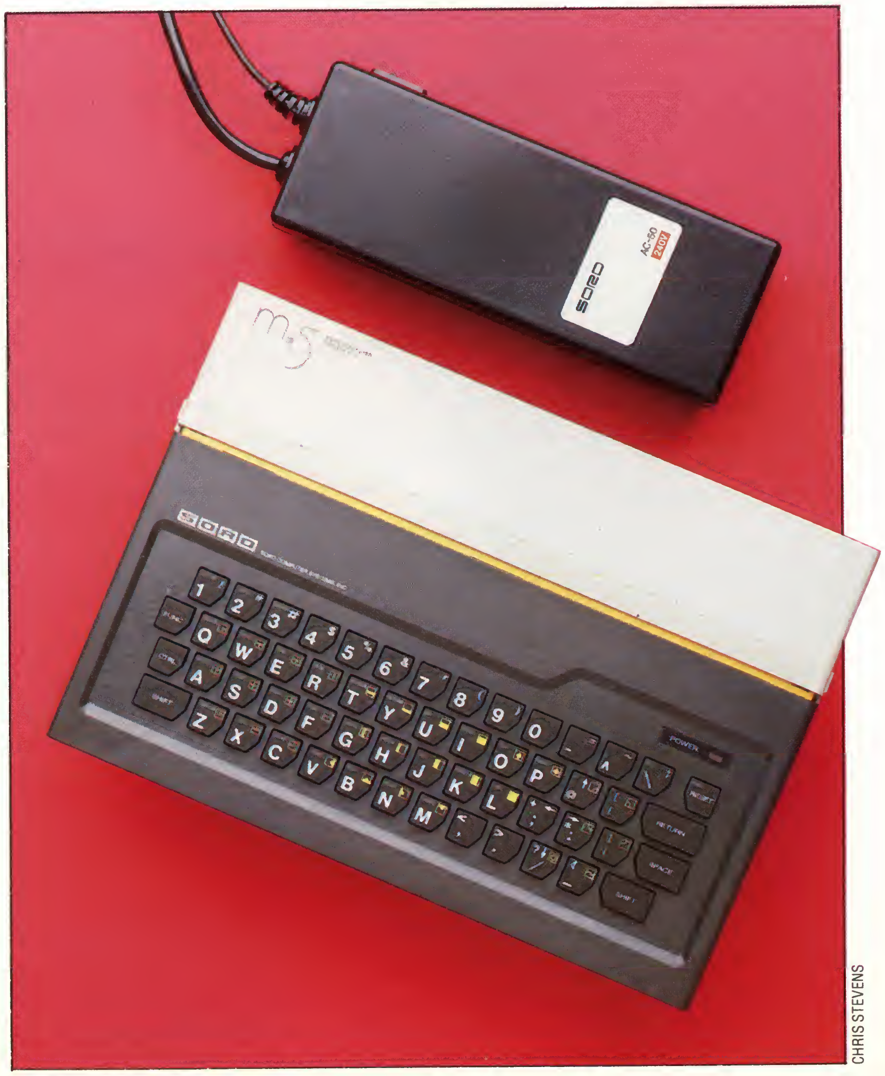
proposed 'MSX standard', developed by a group of major Japanese manufacturers, including Sord. The idea is that if manufacturers stick to these proposed standards for the design of home computers (covering both hardware and the dialect of BASIC to be used), it will be possible to write software that will run on all such machines, without modification. In terms of the graphics chips, the Sord M5 fulfils that standard.

However, MSX also specifies that the sound chip must be the AY-3-8910 from General Instruments. To make sounds, the Sord M5 (like the BBC Micro) uses a TI 76489 chip, which has better control over the range of sounds produced than the GI chip, though it is similar in having three tone channels and one noise channel. This means that the M5 is not a true MSX machine. However, it is sufficiently close to give an idea of what such machines will be like in use.

Three different versions of BASIC, several utilities, some games and other applications can be supplied in ROM cartridge form, and since these may be up to 16 Kbytes in capacity, some useful programs may well appear for this machine.

The M5 may be a little more expensive than other computers of similar physical appearance, but the quality is definitely worth the extra cost.

The Sord M5 Keyboard
The rubber keyboard is slightly larger than the Sinclair Spectrum's, and a lighter touch makes it more suitable for typing. A total of 55 keys can be used in a number of ways, to obtain alphanumeric characters, graphic symbols, or whole BASIC keywords, by means of the FUNC key. All keys will repeat automatically if held down — which is very useful for screen editing



CHRIS STEVENS

Against All Odds

'Even parity' ensures that the number of 1 bits in a byte is always even. This makes transmission errors easier to detect

One of the main advantages of digital computers over analogue devices is that the errors and inaccuracies that occur in all electrical circuits do not accumulate as a signal is passed through many circuits (see page 239). However, when data is transmitted over any distance — whether by means of a serial interface and a pair of wires, or over a telephone line — the background electrical 'noise' in the line can sometimes be enough to flip a single bit from 0 to 1, or vice versa. Normally, the receiving computer would have no way of knowing that this had happened, and would accept the erroneous data as being correct.

Look at what happens if one bit in the ASCII code for the letter Q becomes corrupted:

[ ] 1 0 1 0 0 0 1 (Transmitted ASCII code for Q)
[ ] 1 0 0 0 0 0 1 (Received ASCII code for A)

An error such as this in the transmission of data would, at the least, be a nuisance and could be potentially catastrophic. However, you will remember that ASCII codes are assigned only to values up to 127, which requires only seven bits (numbered 0 to 6). The Most Significant Bit (bit seven) is therefore often used as a 'parity' bit, to detect when an error has occurred.

There are two conventions for using parity bits: 'even parity' and 'odd parity'. We shall consider the former. 'Even parity' means that the parity bit (bit seven in an ASCII code) is set so that the total number of 1 bits in the byte is always an even number. Here's how the letters A and Q would look with even parity:

[0] 1 0 0 0 0 0 1
(the ASCII code for A with even parity)
[1] 1 0 1 0 0 0 1
(the ASCII code for Q with even parity)

There are two 1 bits in the ASCII code for A, so the parity bit is made 0 so that the total of all eight bits is even. In the ASCII code for Q, there are three 1 bits, so the parity bit is made a 1. This brings the total number of 1 bits to four, which is an even number.

Now let's see what would happen if bit four in our ASCII letter Q became corrupted as in the example above.

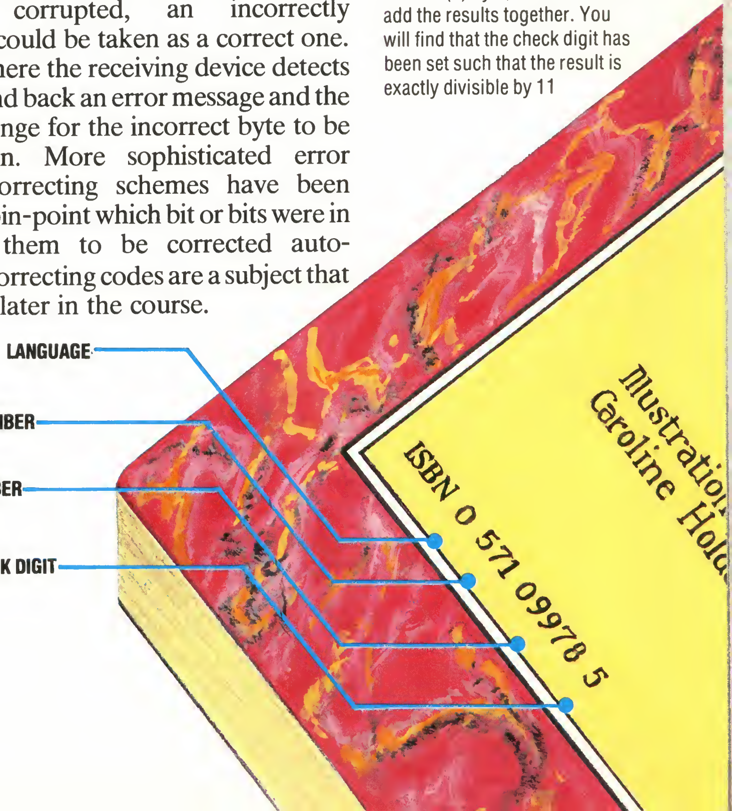
[1] 1 0 0 0 0 0 1 (corrupted ASCII Q)

When the parity of the byte is checked (either by software or by special hardware) it is seen that the correct Q has an even number of 1s in it (including the parity bit). The corrupted Q, by contrast, accidentally had bit four changed from a 1 to a 0, but the original parity bit — bit seven — is still a 1. When the parity of this corrupted byte is checked, it will be found to have an odd number of 1 bits, and so this byte is known to be corrupted and can be rejected. If you think about it, you will see that even if the parity bit itself were to become corrupted in transmission, the fact that an error had occurred would still be picked up by the parity checking process, and the byte would be rejected.

If you look at the ASCII codes used in your own computer, you will probably find that bit seven (the Most Significant Bit, or MSB) is in fact used, but not as a parity bit. This is done to enable the computer to have an additional character set (usually a set of graphics characters), and because errors in data transmission *inside* a computer are very rare. Parity is normally used only when transmitting data over long distances, or when recording data onto a magnetic recording surface (such as tape or disk) which is equally susceptible to 'bit errors'.

Parity checking is fine for indicating that a given byte has been transmitted incorrectly, but it does not indicate which bit in the byte was wrongly transmitted, so the error cannot be corrected by the receiving computer. Worse still, if two bits in a byte become corrupted, an incorrectly transmitted byte could be taken as a correct one.

But in cases where the receiving device detects an error, it can send back an error message and the software can arrange for the incorrect byte to be transmitted again. More sophisticated error detecting and correcting schemes have been devised that can pin-point which bit or bits were in error, enabling them to be corrected automatically. Error correcting codes are a subject that will be discussed later in the course.



Just Checking
The last digit in an International Standard Book Number (ISBN) is a check digit — equivalent to parity in a computer. Multiply the first digit (0 here) by 10, the second (5) by 9, and so on, then add the results together. You will find that the check digit has been set such that the result is exactly divisible by 11

Rank And File

Continuing our programming project to develop a computerised address book, we now look at how our file of data will need to be split up into records and fields

We ended the previous instalment of the Basic Programming course by setting the task of refining the elements of the programming exercise through one or more layers of 'pseudo-language', up to the point where the examples could be coded into BASIC. We will start by revising this exercise and giving some possible solutions. The first 'Statement of Objectives' for the exercise was:

```
INPUT
A name (in any format)
OUTPUT
1. A forename
2. A surname
```

In our first level refinement we found that this could be broken down into six steps (later we found that the last step could be dispensed with). These were:

```
1. Read the name (* READ *)
2. Convert all the letters to upper case (* CONVERT *)
3. Find the last space (* SPACE *)
4. Read the surname (* READSURNAME *)
5. Read the forename (* READFORENAME *)
6. Discard the non-alphabetics from the forename
```

We are treating all of these activities as subroutines and the name we have assigned to each subroutine is given in brackets. Unfortunately, most versions of BASIC are unable to call subroutines by name and it will be necessary when writing the final program to insert line numbers after the respective GOSUBs. During the development phase, however, it is much easier to refer to subroutines by name. These names can then later be incorporated in REM statements. We are indicating this use of named subroutines by putting the names within asterisks. In languages that can call subroutines by name (such as PASCAL), subroutines like these are usually referred to as 'procedures'.

Even though your BASIC may not be able to handle procedures, it is recommended that you pretend it can while programming at the pseudo-language stage. Similarly, your version of BASIC may not be able to handle long variable names such as COUNT or STREETNAME$, but at the pseudo-language level it is easier and clearer to assume that it can. Try to make them descriptive. It is much clearer to call a temporary variable for a string TEMPSTRING$ than to call it XV$. Fortunately, many versions of BASIC now allow longer variable names.

We have already developed the second of the steps (Convert all the letters to upper case) through a second and third level of refinement and created a short program in BASIC to do this task. We will now attempt this for the other steps:

```
2ND REFINEMENT
3. (Find last space)
BEGIN
LOOP while unscanned characters remain in NAME$
  IF Character = "  "
    THEN note position in a variable
    ELSE do nothing
  ENDIF
ENDLOOP
END
```

```
3RD REFINEMENT
3. (Find last space)
BEGIN
READ FULLNAME$
LOOP (while unscanned characters remain)
  FOR L = 1 to length of FULLNAME$
  READ character from FULLNAME$
  IF character = "  "
    THEN LET COUNT = position of character
    ELSE do nothing
  ENDIF
ENDLOOP
END
```

We are now in a position to code from pseudo-language into programming language:

```
10 INPUT "INPUT FULL NAME "; FULLNAME$
20 FOR L = 1 TO LEN (FULLNAME$)
30 LET CHAR$ = MID$ (FULLNAME$,L,1)
40 IF CHAR$ = "  " THEN LET COUNT = L
50 NEXT L
60 PRINT "LAST SPACE IS IN POSITION ";COUNT
70 END
```

Note that line 10 is a dummy input for testing the routine; line 60 is a dummy output, also for testing; and line 70 will have to be changed to RETURN when the routine is used as a subroutine.

Now let's try the same process for step four:

```
2ND REFINEMENT
4. (Read surname)
BEGIN
Assign characters to right of last space to SURNAME$
END
```

```
3RD REFINEMENT
4. (Read surname)
BEGIN
  READ FULLNAME$
```

```
    Locate last space (call ★ SPACE ★ subroutine)
    LOOP while characters remain in string after space
       READ characters and add to SURNAME$
    ENDLOOP
 END
```

Before going on to code this into BASIC, you should note some potential pitfalls. In locating the last space in the final refinement above, the pseudo-language calls for the use of the ★SPACE★ subroutine, but it would not be possible to write this out in BASIC and test it if the ★SPACE★ subroutine had not already been written. As a general rule, it is not worth coding each module into BASIC (or any other high level language) until the whole program has been developed in pseudo-language. However, if you do wish to test a module, you may need to write some dummy variable values as well as dummy inputs and outputs. In the example above, COUNT is the variable that holds the value of the position of the last space in FULLNAME$. In testing, we can cheat a little by assuming that the routine to do this works properly:

```
10 LET FULLNAME$ = "TOM BROWN"
20 LET COUNT = 4
30 FOR L = COUNT + 1 TO LEN (FULLNAME$)
40 LET SURNAME$ = SURNAME$ + MID$
   (FULLNAME$,L,1)
50 NEXT L
60 PRINT "SURNAME IS "; SURNAME$
70 END
```

Here is the process for finding the forename (step five). Remember, we decided that a forename is a concatenation of all the alphabetic characters up to the last space in the name. Full stops, apostrophes, spaces and so on were to be discarded.

2ND REFINEMENT

```
5. (Read forename)
BEGIN
LOOP while characters remain in FULLNAME$ up to
   last space
   Scan characters
   IF character is not a letter
      THEN do nothing
      ELSE add character to FORENAME$
   ENDIF
ENDLOOP
END
```

3RD REFINEMENT

```
5. (Read forename)
BEGIN
LOOP while characters remain up to COUNT
   LET TEMPCHAR$ = Lth character in string
   IF TEMPCHAR$ is not a letter
      THEN do nothing
      ELSE LET FORENAME$ = FORENAME$ +
         TEMPCHAR$
   ENDIF
ENDLOOP
```

Now we are ready to code into BASIC, but as an intermediate stage, we are going to use unnumbered BASIC statements in a structured format so that you can compare the structure with the stage above:

TONY LODGE

CODING

```
5. (Read forename)
REM BEGIN
REM LOOP
   FOR L = 1 TO COUNT - 1
      LET TEMPCHAR$ = MID$ (FULLNAME$,L,1)
      LET CHAR = ASC(TEMPCHAR$)
      IF CHAR>64 THEN FORENAME$ =
         FORENAME$ + CHR$(CHAR)
      REM ENDIF
   NEXT L: REM ENDLOOP
REM END
```

In ordinary BASIC this would be:

```
10 FOR L = 1 TO COUNT - 1
20 LET TEMPCHAR$ = MID$(FULLNAME$,L,1)
30 LET CHAR = ASC(TEMPCHAR$)
40 IF CHAR > 64 THEN FORENAME$ = FORENAME$
   + CHR$(CHAR)
50 NEXT L
60 END
```

As it stands, however, this program would not work. There are three problems with it: COUNT needs to be assigned a value; there is no provision for inputing a name (assigning a string to FULLNAME$); and there is no 'output' in the form of a print statement for us to check if it has worked properly.

If this routine were part of a subroutine, the parameters passed to it (the input) and the parameters passed from it (the output) would have to be handled elsewhere in the program. This is a very important consideration: the flow of information within a program should always be carefully thought through before we begin to code into BASIC. This is particularly important when we are using variables (COUNT, for example) and the same variable name is used in different parts of the program. There is no point in calling a subroutine that uses a variable such as COUNT if the subroutine has no way of knowing what its value is supposed to be. If a subroutine initialises the value of COUNT, that value will remain the same unless a new value is assigned later — perhaps in another subroutine. This is one reason why it is not good programming practice to jump out from the middle of a loop, since the value of the loop variable will be unknown. Consider the consequences of having these two program fragments as parts of different subroutines in a program:

Part of subroutine X

```
FOR L = 1 TO LEN(WORD$)
LET CHAR$ = MID$(WORD$,L,1)
IF CHAR$ = " . " THEN GOTO 1550
NEXT L
```

Part of subroutine Y

```
FOR Q = 1 TO LIMIT
LET A(L) = P(Q)
NEXT Q
```

This part of subroutine Y is reading values into a subscripted array, where the subscript is denoted by the variable L. If subroutine Y is called after subroutine X, and if the test condition in subroutine X has been met (that one of the characters is a " . "), the value of L would be completely unpredictable and so we would not know which element of the array values were being assigned to in subroutine Y. Apart from the error of branching out of a loop, this subroutine also uses a GOTO, and this practice should also be avoided. GOTOs lead to confusion and they should be avoided wherever possible.

To avoid confusion when using variables, it is good practice to make a list of them at the pseudo-language stages of program development, together with notes saying what they are being used for. Some languages (but not BASIC) allow variables to be declared as 'local' or 'global' — that is, they have values that apply either in only part of a program (local) or throughout the whole program (global). Many variables, such as those used in loops (for example, the L in LET L = 1 TO 10), are almost always local, so it is often wise to initialise the value of the variable before it is used (for example, LET L = 0). Some languages, such as PASCAL, insist on this; and although BASIC always assumes the initial value of a variable is 0 (unless otherwise stated), initialising is still recommended.

So far we have formulated a reasonable definition of a name for the purposes of our computerised address book, and developed some routines that can handle names in various ways that we shall use in our complete program. Now let's once again distance ourselves from the details of program coding and consider the structure of the 'records' in our address book 'file'

The terms 'record', 'file' and 'field' have fairly specific meanings in the computer world. A *file* is a whole set of related information. In a computer system it would be an identifiable item stored on a floppy disk or on a cassette tape and it would have its own name, usually referred to as a filename. We can consider our entire address book as a file, and we shall call it ADBOOK.

Within a file we have *records*. These are also sets of related information. If we think of our address book as a card index box, the file would be the whole box full of cards and the records would be the individual cards — each one with its own name, address and telephone number.

Within each record we have *fields*. The fields can be considered as one or more rows of related information within the record. Each of the records in our ADBOOK file will have the following fields: NAME, ADDRESS and PHONENUMBER. A typical record would look like this:

```
Peter Edvadsen
16A Holford Drive
Worsley
Manchester
061-540 2588
```

In this record there are three fields: the name field, which comprises alphabetic letters (and, possibly, the apostrophe in names such as Peter O'Toole); the address field, which comprises a few numbers and many letters; and the telephone number field, which comprises only numbers (ignoring the problem of whether or not to allow hyphens in numbers like 01-258 1191). Before we can begin to write a program to handle complex information such as this with flexibility, we must decide how to represent the data within the computer. One way might be to consider all the information within a record to be just one long character string. The problem with this approach is that extracting specific information is extremely difficult. Let's assume that the following entry is just one long character string:

```
PERCIVAL R. BURTON
1056 AVENUE OF THE AMERICAS
RIO DEL MONTENEGRO
CALIFORNIA
U.S.A.
(415) 884 5100
```

If we were searching the records to find the telephone number of PERCIVAL R. BURTON, would it be safe to assume that the last 14 characters in the record represented the number? What if we had included the international dialling code, like this: 0101 (415) 884 5100? Then the number would have had a total of 19 characters. To overcome this difficulty, the telephone number is assigned a separate field, and the program will give us all the characters (or numbers) in that field when requested.

The difficulty with this approach is that there has to be some way of relating the various separate fields, so that referring to one field (the name field, for example) can give us the other fields on the record, as well. One way this could be tackled is to have a further field associated with the record just for indexing purposes. If a record was, for example, the 15th record in the file, its index field would contain the number 15. This could then be used to point to the elements in a number of arrays. To illustrate this, let us suppose one record looked like this:

Jamie Appleton	NAME field
15 Pantbach Road	STREET field
Llandogo	TOWN field
Gwent	COUNTY field
0594 552303	PHONE NUMBER field
015	INDEX field

If we knew the name of this person and wanted his telephone number, all we would have to do would be to search through the elements of the array holding the names until a match was found. We would then find which element of the array the name was in — in this case, number 15. Then all we would need to do would be to find the 15th element in the PHONE NUMBER array to get the right telephone number.

If we had a number of friends in the Forest of

Dean area, we might want the program to search for every occurrence of 'Cinderford' in the TOWN field. The program could search through the TOWN fields and note the location of each occurrence of Cinderford. All that would then be necessary, to print the names and addresses of all these friends, would be to retrieve all the elements having the same number from all the arrays for each 'Cinderford' record. Using this approach, there would be no need to inspect the INDEX field, and the technique has the merit of being a relatively simple operation.

In the next instalment we will look at some of the problems involved in searching through lists to find specific items.

Exercise

■ Assume that records with the following fields will be adequate for our computerised address book:

NAME field
STREET field
TOWN field
COUNTY field
PHONE NUMBER field

Suppose that one of the options offered by a menu in the computerised address book is:

5. CREATE A NEW ENTRY

You type 5 and the program branches to the part where new records are created (you may assume that there are no entries in the address book yet). Since the program is to be fully menu-driven, you will always be prompted for the entries expected — with prompts such as ENTER THE NAME, ENTER THE STREET and so on. Here is a list of the expected results:

1. An element in an array for the name
2. An element in an array for the street
3. An element in an array for the town
4. An element in an array for the county
5. An element in an array for the phone number

Your task is to develop this, through a process of top-down programming using a pseudo-language, to a point where direct conversion into BASIC becomes possible. The pseudo-language can follow your own rules; we only suggest that you use capital letters for keywords such as IF, LOOP and so on, and small letters for descriptions in ordinary English of the operations to take place.

Basic Flavours

SPECTRUM

Step 3

```
  10 INPUT "INPUT FULL NAME";F$
  15 LET COUNT=0
  20 FOR L=1 TO LEN F$
  30 LET C$=F$(L)
  40 IF C$=" " THEN LET COUNT = L
  50 NEXT L
  60 PRINT"LAST SPACE IS IN
     POSITION";COUNT
  70 STOP
9990 DEF FN M$(X$,P,N)=X$(P TO
     P+N-1)
9991 DEF FN L$=X$(    TO N)
9992 DEF FN R$=X$(LEN X$-N+1 TO    )
```

In this programming project, the string functions MID$, LEFT$, RIGHT$ will be much used. Their equivalents in Sinclair BASIC are:

LEFT$(F$,N)	replace by F$(TO N)
RIGHT$(F$,N)	replace by F$(LEN(F$)-N+1 TO)
MID$(F$,P,N)	replace by F$(P TO P+N-1)
MID$(F$,P,1)	replace by F$(P)

Note that string variable names on the Spectrum cannot be more than one letter long (plus the "$").

Step 4

```
 5 LET S$=""
10 LET F$="TOM BROWN"
20 LET COUNT=4
30 FOR L=COUNT+1 TO LEN F$
40 LET S$=S$+F$(L)
50 NEXT L
60 PRINT "SURNAME IS ";S$
70 STOP
```

Step 5

```
 5 LET C$=""
10 FOR L=1 TO COUNT -1
20 LET T$=F$(L)
30 LET CHAR=CODE T$
40 IF CHAR>64 THEN LET C$=C$+CHR$
   CHAR
50 NEXT L
60 STOP
```

In this fragment, the problem of single letter string variable names has arisen: F$ is the Spectrum equivalent of the variable FULLNAME$, so C$ has to stand in for the variable FORENAME$.

Part of subroutine X

```
FOR L=1 TO LEN W$
LET C$=W$(L)
IF C$="." THEN GOTO 1550
NEXT L
```

Part of subroutine Y

```
FOR Q=1 TO LIMIT
LET A(L)=P(Q)
NEXT Q
```

VARIABLES

Of the most popular home computers, only the BBC Micro supports long variable names such as FULLNAME$. The Spectrum allows long numeric variable names, but only single letter string variable names. The Dragon 32, Vic-20, and Commodore 64 support long variable names, but only the first two characters are significant, so that FULLNAME$ is valid, but refers to the same memory location as FUJIYAMA$: both have the same first two characters.

On the Oric-1 variable names cannot be more than two characters (first a letter then a number or a letter), while the Lynx allows only single letter variable names, though both lower- and upper-case letters are valid and distinct.

Tracing Paper

Images drawn on paper can be transferred into your computer by means of a digitiser or graphics tablet

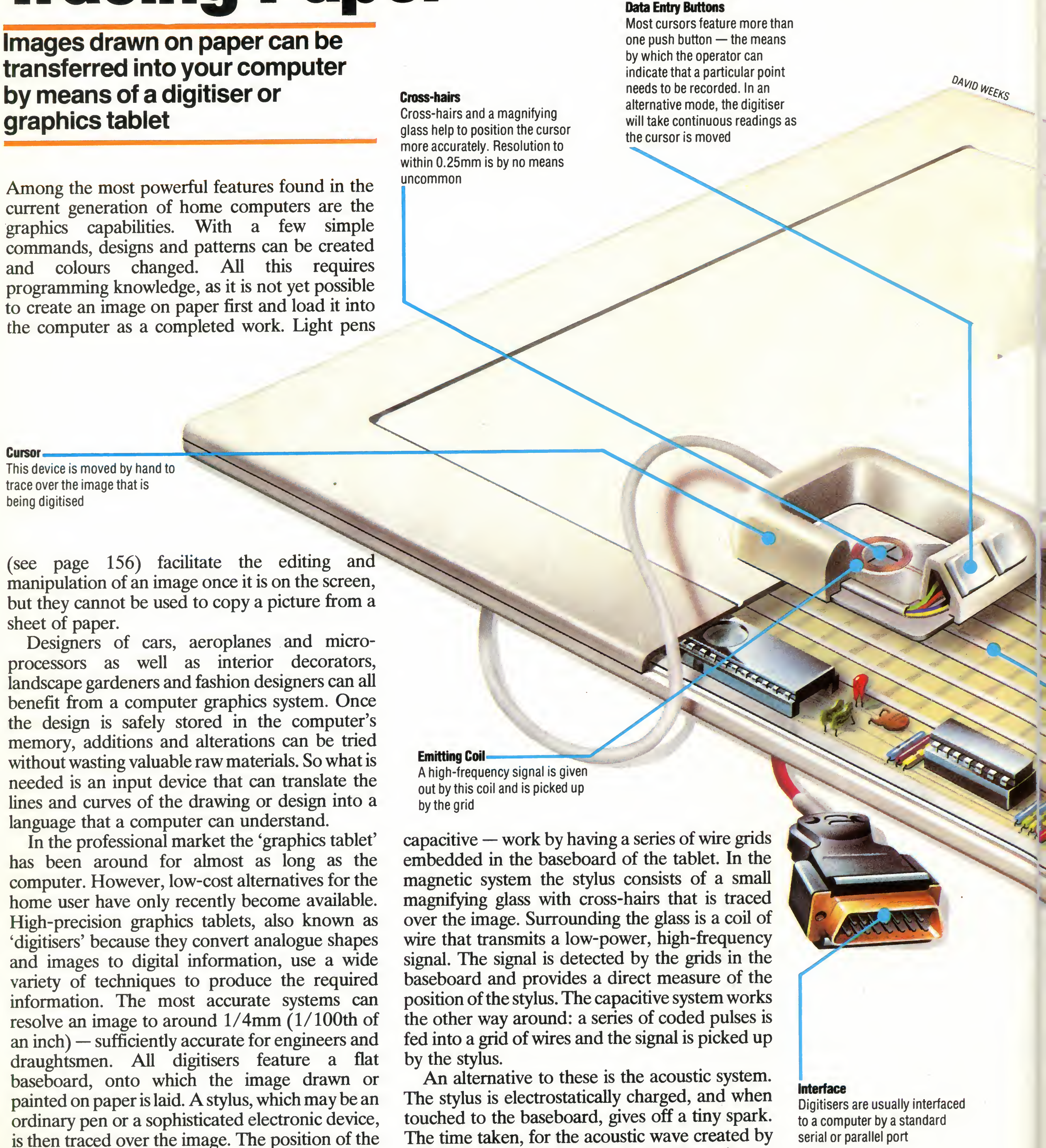
Among the most powerful features found in the current generation of home computers are the graphics capabilities. With a few simple commands, designs and patterns can be created and colours changed. All this requires programming knowledge, as it is not yet possible to create an image on paper first and load it into the computer as a completed work. Light pens (see page 156) facilitate the editing and manipulation of an image once it is on the screen, but they cannot be used to copy a picture from a sheet of paper.

Designers of cars, aeroplanes and microprocessors as well as interior decorators, landscape gardeners and fashion designers can all benefit from a computer graphics system. Once the design is safely stored in the computer's memory, additions and alterations can be tried without wasting valuable raw materials. So what is needed is an input device that can translate the lines and curves of the drawing or design into a language that a computer can understand.

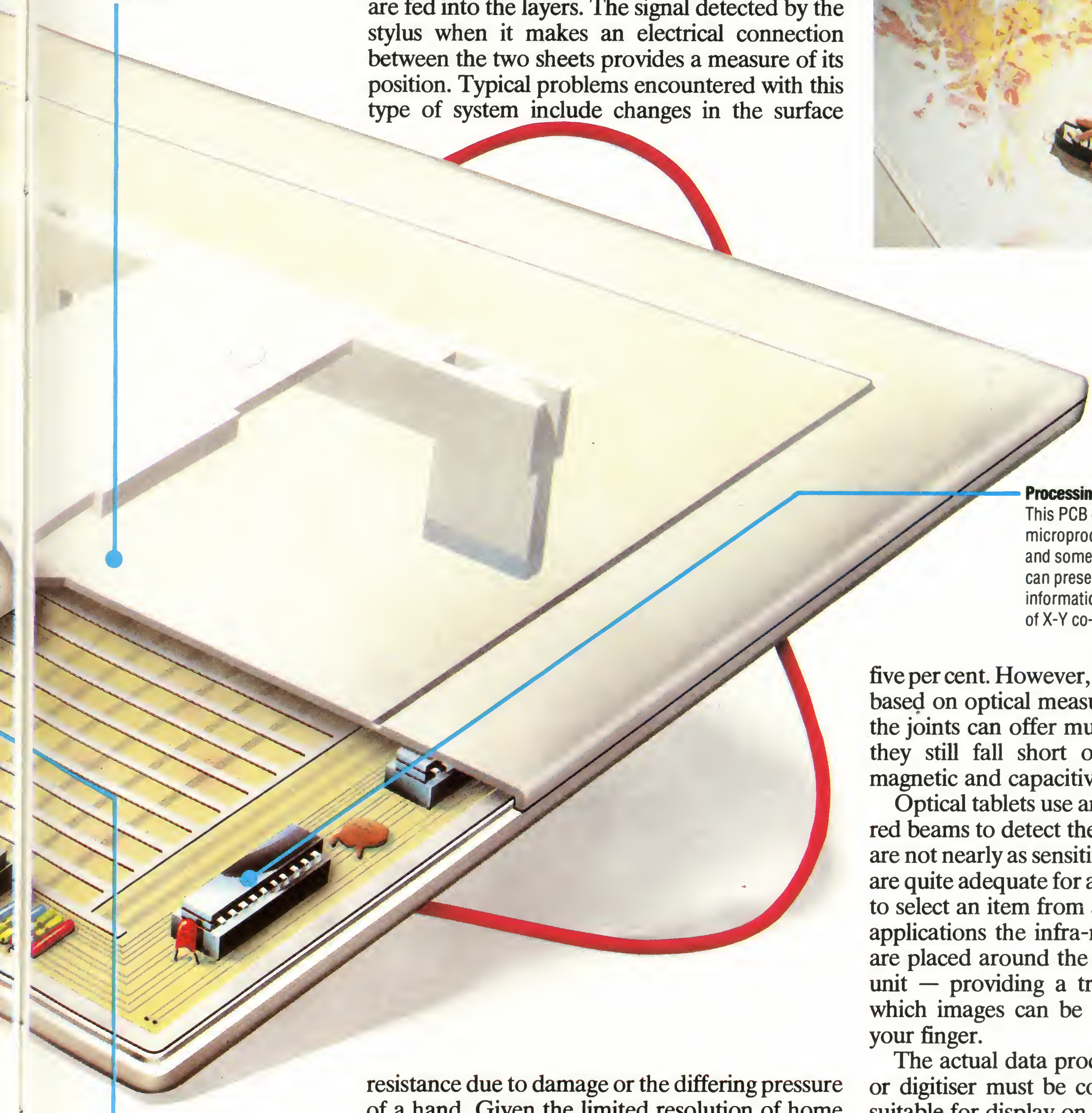
In the professional market the 'graphics tablet' has been around for almost as long as the computer. However, low-cost alternatives for the home user have only recently become available. High-precision graphics tablets, also known as 'digitisers' because they convert analogue shapes and images to digital information, use a wide variety of techniques to produce the required information. The most accurate systems can resolve an image to around 1/4mm (1/100th of an inch) — sufficiently accurate for engineers and draughtsmen. All digitisers feature a flat baseboard, onto which the image drawn or painted on paper is laid. A stylus, which may be an ordinary pen or a sophisticated electronic device, is then traced over the image. The position of the stylus is detected by the digitiser and transmitted as a changing pair of co-ordinates to the computer.

The two most accurate systems — magnetic and capacitive — work by having a series of wire grids embedded in the baseboard of the tablet. In the magnetic system the stylus consists of a small magnifying glass with cross-hairs that is traced over the image. Surrounding the glass is a coil of wire that transmits a low-power, high-frequency signal. The signal is detected by the grids in the baseboard and provides a direct measure of the position of the stylus. The capacitive system works the other way around: a series of coded pulses is fed into a grid of wires and the signal is picked up by the stylus.

An alternative to these is the acoustic system. The stylus is electrostatically charged, and when touched to the baseboard, gives off a tiny spark. The time taken, for the acoustic wave created by the spark to reach two microphones, gives a measure of the stylus position. Amongst other things, this offers the possibility of digitising the third dimensions, by means of a signal passing



Data Entry Buttons
Most cursors feature more than one push button — the means by which the operator can indicate that a particular point needs to be recorded. In an alternative mode, the digitiser will take continuous readings as the cursor is moved
Cross-hairs
Cross-hairs and a magnifying glass help to position the cursor more accurately. Resolution to within 0.25mm is by no means uncommon
DAVID WEEKS
Cursor
This device is moved by hand to trace over the image that is being digitised
Emitting Coil
A high-frequency signal is given out by this coil and is picked up by the grid
Interface
Digitisers are usually interfaced to a computer by a standard serial or parallel port



Baseboard
The image to be digitised is placed flat on this board. On some systems, an electrostatic charge is applied to the board to 'glue' the paper temporarily flat. It is very important that the image doesn't move relative to the board

Processing Board
This PCB contains a microprocessor, some ROM and some RAM. This is so that it can present the computer with information in the form of pairs of X-Y co-ordinates

Receiving Grid
Embedded in the baseboard is a grid of wires that can pick up the signal given out by the coil. The spacing of the grid is considerably coarser than the finest resolution of the digitiser, because the processing circuitry can interpolate from the relative strength of the signal picked up by adjacent wires

through the object.

At the lower end of the scale is the pressure-sensitive tablet: the image is placed on it and then traced with a stylus. This requires more pressure than the other systems. Two electrically conductive sheets are separated by a cellular insulator and two different high-frequency signals are fed into the layers. The signal detected by the stylus when it makes an electrical connection between the two sheets provides a measure of its position. Typical problems encountered with this type of system include changes in the surface resistance due to damage or the differing pressure of a hand. Given the limited resolution of home computer graphics, the accuracy of this method is more than adequate for today's home computers.

The cheapest and simplest digitisers are the pantographs — based on the principle of the old-fashioned drawing aid, constructed from linked arms. They use co-ordinate geometry to provide a direct measure of the position of the stylus. Variable resistances mounted at the two joints provide voltages proportional to the angles in the 'shoulder' and 'elbow' of the jointed arm. The resolution of the pantograph is limited by the accuracy of both the variable resistances and the mechanical linkages; typically it is only around five per cent. However, sophisticated pantographs based on optical measurement of the rotation of the joints can offer much better results although they still fall short of the capabilities of the magnetic and capacitive systems.

Optical tablets use an intersecting grid of infra-red beams to detect the position of a stylus. They are not nearly as sensitive as the other systems but are quite adequate for allowing a finger to be used to select an item from a program menu. In some applications the infra-red sources and detectors are placed around the edge of the visual display unit — providing a truly interactive screen on which images can be drawn simply by moving your finger.

The actual data produced by a graphics tablet or digitiser must be converted into information suitable for display on the screen and to this end most of the commercial products come with all the necessary software. However, just entering the data isn't the end of the usefulness of graphics tablets. Once the information is stored in the computer the tablet can be used as an editing tool, allowing colour to be added or changed and shapes to be modified. The surface of the tablet can be programmed to act as a menu that selects standard options from the program so that the keyboard need only be used for selecting the main functions. Computer animation systems (see page 181) all have a high-quality graphics tablet as their main form of input.

Mapping It Out
One of the most widespread professional uses for digitisers is collecting data from maps and surveys. Here, the computer is being used to predict the location of new oilfields from digitised geological data

Gottfried Leibniz

1646
Born on July 1 in Leipzig

1661
Enrols at University of Leipzig and awarded degree at 17

1660's
Works as lawyer and diplomat. Publishes paper on 'The Art of Combination'

1672
In Paris, he develops the principle of Sufficient Reason

1673
Calculating machine presented to Royal Society in England

1675
Invents calculus independently of Newton

1676
Considers dynamics through the concept of kinetic energy

1678
Appointed librarian and adviser to the Duke of Hanover

1679
Develops binary mathematics

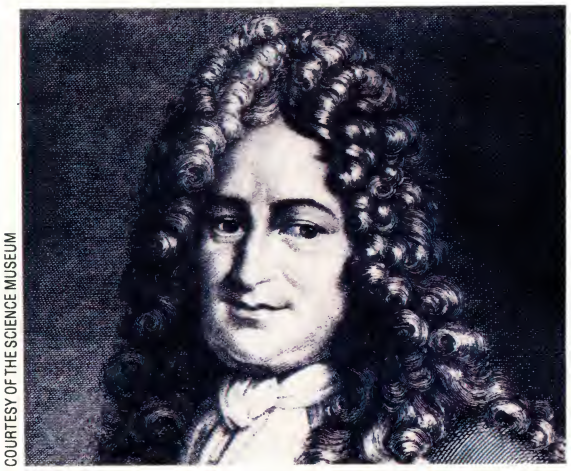
1683
Publishes pamphlet, 'The Most Christian War God', an attack on Louis XIV

1690's
His genealogy of the House of Hanover expands into a History of the World. Develops an interest in linguistics and the origin of languages

1700
Organises Berlin Academy of Sciences

1714
Responsible for establishing the right of succession of George I to the vacant English throne after the death of Queen Anne

1716
Dies in Hanover November 14


COURTESY OF THE SCIENCE MUSEUM

Scientists involved in the fifth generation computers are taking an interest in the work of this 17th century thinker

Gottfried Wilhelm Leibniz was the leading scientific light of his time — the period known as The Age of Reason. He was born in the central European city of Leipzig in 1646 and died in Hanover in 1716. During his life of three score years and ten (the sort of exact figure you might expect from a mathematician), he invented calculus, worked on dynamics, and made contributions to geology, theology, history, linguistics and philosophy. Most important of all, he developed ideas that would be fundamental to the creation of the computer.

He began his travels at the age of 20, after the University at Leipzig refused to confer a doctorate of law on him because of his youth. Throughout his life, without any private means to support him, Leibniz was forced to take up work that hampered his scientific research. In his early twenties he worked as a lawyer and diplomat; later in life he was a librarian and adviser to royalty.

His interests were wide-ranging, and his cosmopolitan nature led to extensive travel in Europe talking with all the great thinkers of his time. Leibniz was a prolific letter writer, as well — engaging in correspondence with over 600 people.

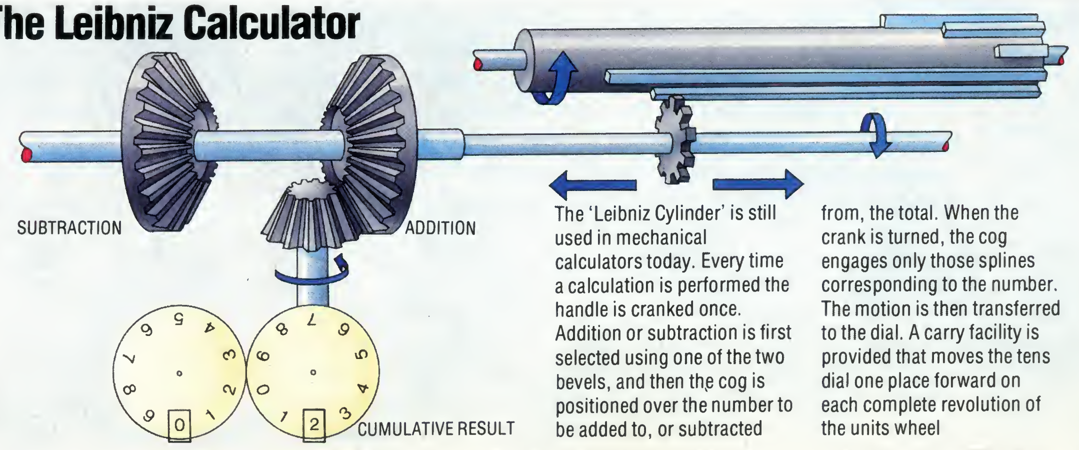
His first important contribution to philosophy came in 1672 when he formulated the principle of Sufficient Reason. This held, simply, that there must be a reason for everything, and 'everything is for the best in the best of all possible worlds'.

Turning his attention to mathematics, he then set to work to perfect the Pascaline adding machine invented by Blaise Pascal in 1642 (see page 86). Leibniz sought to upgrade it so that it would be capable of both multiplication and division. He did so by designing a mechanical device called the Leibniz Cylinder (see below). Leibniz's device was a major breakthrough for its time. Previously, because of the complexity of manipulating Roman numerals, multiplication had been taught only in the higher institutes of learning. A machine that could multiply mechanically made arithmetic more accessible. Once Leibniz had perfected this device, he moved on from base ten arithmetic to consider and formalise binary mathematics.

Leibniz's greatest ambition was to devise a universal language that could use the clarity and precision of mathematics to solve any problem that mankind faced. His language was to use abstract symbols to represent the fundamental 'atoms' of understanding, with a set of rules to manipulate these symbols. His attempt failed; but his ideas were taken up in a more modest way in the early 20th century by Bertrand Russell, who tried to explain mathematics in terms of a formal logical 'language'.

In the last few years, interest has been rekindled in the work of Leibniz by the scientists involved in the long-term project to create the fifth generation of computers. These machines, it is hoped, will be able to solve any problems of human endeavour with the same speed and certainty that computers of today execute mathematical calculations. To do this they will require a new sort of language altogether.

The Leibniz Calculator



The 'Leibniz Cylinder' is still used in mechanical calculators today. Every time a calculation is performed the handle is cranked once. Addition or subtraction is first selected using one of the two bevels, and then the cog is positioned over the number to be added to, or subtracted from, the total. When the crank is turned, the cog engages only those splines corresponding to the number. The motion is then transferred to the dial. A carry facility is provided that moves the tens dial one place forward on each complete revolution of the units wheel

KEVIN JONES

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

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