ISSN 0265-2919

80p

THE HOME COMPUTER COURSE 20

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

Over The Rainbow Sinclair's Spectrum can be expanded into a powerful system 386

Memotech MTX 512 A home computer designed for Basic or machine code use 390

Software

Play Acting Simulation programs can turn a home computer into an interactive teacher 389

Basic Programming

Time And Motion We add the ability to sort and modify records and store files 396

Insights

Computer Literate Science fiction writers have often been surprisingly accurate in their predictions for the future 381

Common Sense A home computer can be used to control domestic appliances 394

Passwords To Computing

Original Author Software is now available that will generate programs almost automatically 384

Purpose Designed We examine the second generation chip that is dramatically reducing the cost of home computers 388

Pioneers In Computing

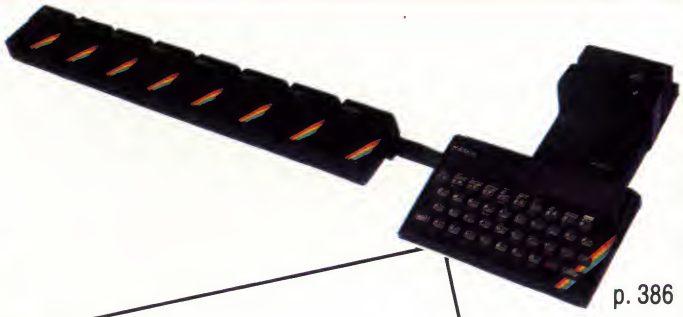
Vannevar Bush His analyser was able to solve complex differential equations by electromechanical means 400

Sound And Light

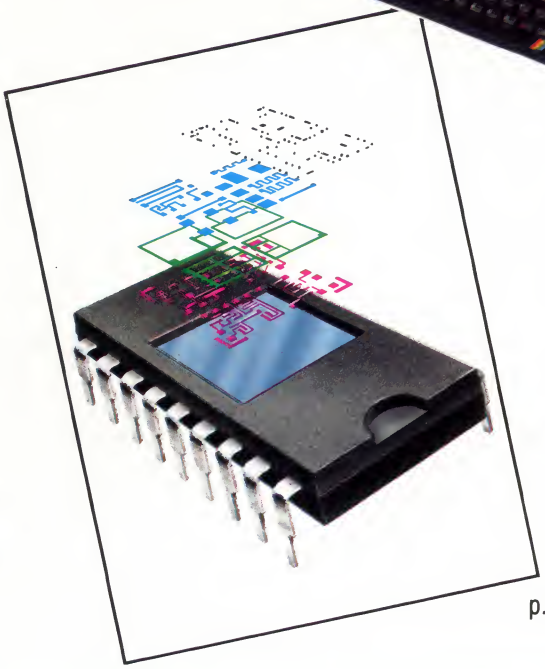
Sound Spectacular...Light Show We look at the Oric-1's sound capabilities and scrutinise the Spectrum's graphics 392

Next Week

- **The Osborne-1 was the first truly portable data processing system. We examine this remarkably compact machine**
- **Microprocessors are increasingly being used to produce exciting toys**
- **We look at the way in which optical discs can be used to provide fast access mass storage**



p. 386



p. 388

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 Assistant** Liz Dixon; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Lisa Kelly, Steven Colwill, Richard King, Geoffrey Nairns; **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** 85 Charlotte Street, London W1;

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.

Computer Literate

© COLUMBIA WARNER

Superman III
Computer fraud is the central theme of the third Superman film. Richard Pryor plays a villain who makes his fortune by stealing a half-cent from every transaction in a bank's computer. This part of the plot is based on several real cases of fraud. The film ends with the destruction of the largest computer in the world, which has been built entirely for criminal purposes

Computers have featured widely in science fiction. In many cases writers have accurately predicted technologies that we now take for granted

Many scientific and technical achievements have occurred to writers of fiction, or film makers, long before they were actually feasible. Arthur C Clarke, the author of *2001 — A Space Odyssey,* first put forward the idea of geostationary satellites in an article for the magazine *Wireless World* in the early 1950's — almost two decades before they were developed. Similarly, Robert Heinlein's short story, *Waldo,* described remotely-controlled manipulators well before robot hands came into use. In fact, many inventors and development scientists have been inspired by the creative ideas of science fiction writers and film makers.

Fictional computers, however, often bear little resemblance to reality. In the futuristic film *Rollerball,* for example, a computer that has speech-recognising input and voice output takes the form of a cube-shaped tank of liquid. Computers resembling the real thing, of course, tend to be undramatic and less interesting, although they have often been seen in films as a part of the furniture. There can be no doubt that in the 1960's and 1970's, films that featured computers closely modelled on the actual machines helped to educate the general public by showing what these new, near-mythical 'computers' actually looked like.

Creative imaginations began conjuring up ideas of computers not very long after Charles Babbage (see page 220) began his pioneering work on an Analytical Engine, in the middle years of the 19th century. In 1879, Edward Page Mitchell wrote a story called *The Ablest Man in the World,* which described how a calculating machine was implanted into the brain of an idiot, turning him into a genius. Mitchell's ideas preceded actual scientific advance on many counts. In the first instance, he grappled with the idea of miniaturisation — the computing machine is at once small enough to fit into the idiot's cranium and yet powerful enough to endow him with a superior intellect. Secondly, Mitchell prefigured the idea of interconnecting a computer with the human body. Today, almost a century after the story was written, the techniques to connect simple controllable electromechanical devices to the body's central nervous system are beginning to be perfected.

Generally, few writers have an extensive knowledge of computer architecture, though some are accomplished engineers and many use computers (in the form of word processors) in their work. Yet most can present a convincing picture of intergalactic travel, even though they are not highly qualified astro-physicists or rocketry experts. Similarly, there is no reason why writers cannot speculate on the possible attributes of future generations of computers without

comprehensive knowledge of the machines currently in use.

This speculation, however, has led to a standard science fiction computer that appears to have — by current computer standards — a range of facilities that are impossible to achieve. To begin with, this standard computer has stored in its memory absolutely every piece of information and idea ever thought of, and it can retrieve any of this information instantaneously, by processes analogous to the workings of the human mind. The computer HAL in *2001 — A Space Odyssey* is one such super-intelligent machine.

B.F.I. STILLS

Spaced Oddity
HAL — the Heuristically programmed ALgorithmic computer in Arthur C Clarke's '2001 — A Space Odyssey' is a good example of the omnipotent computing machine often seen in science fiction films. In this case HAL is entrusted with details of the mission's objectives while the human crew members are not, which leads the machine to believe the men to be dispensable. It is widely believed that Clarke chose the letters HAL for their proximity to IBM in the alphabet

The supercomputer generally postulated by science fiction writers is omnipresent as well, although it appears to each user that he alone has access to the machine. Spoken output (with no hint that it might be a synthetic product of strung together phonemes) and voice recognition (which infallibly makes allowance for the characteristics of individual speech patterns) are both essential requirements of this supermachine; while visual object recognition and the ability to synthesise food (perhaps from its basic elemental constituents) are other useful attributes.

The supercomputer that we have outlined here is usually enlivened with human attributes as well, and this characterisation makes it appear as some sort of superbeing. However, the character of the computer can sometimes turn decidedly malevolent or deranged. In the film *Dark Star,* for example, a computer-controlled bomb is given the unsettling characteristics of a psychopathic killer. When portrayed in this way, the supercomputer certainly does belong in the realms of fantasy. But, on the other hand, we can recognise in modern computing equipment the possible ancestors of some of the other attributes we have outlined.

High capacity memory with a very short access time is already possible. By the early 1980's, gigabyte memories (a thousand million bytes) had been created, and the fastest commercial machines were processing information at much more than ten million instructions per second. Again, in the field of spoken output, we are very close to achieving the perfection shown in films where computers talk to their operators. The quality of spoken output simply depends on available memory space, processing speed and programming time. Voice recognition, however, is more difficult to achieve because there is such a wide divergence between individual speech patterns. Two people may be speaking the same language, but to the computer they can seem to be speaking entirely different ones.

Visual object recognition is also in its infancy, but the technology is advancing rapidly. When we looked at industrial robots (see page 281), we noted that a great deal of progress was being made in object recognition by means of charge-coupled device television cameras, and that the robot could pick a specified item out of a mixed bag. Meaningful visual recognition depends on the size of the visual vocabulary, which again is a function of memory size and processor power. As for food synthesis, it may not be possible to make a meal look like meat and potatoes or fried fish, but it is certainly possible to make it taste and smell like them, even if computers cannot, as yet, create them from their elements.

Not all authors go to these lengths of attributing remarkable powers to their fictional machines. John Brunner, for example, in the science fiction novel *Stand On Zanzibar,* published in 1969, describes the world as it might be in 2010 when the problems of overcrowding and starvation have reached crisis level. The computer that he describes, which he calls Shalmaneser, obviously has considerable memory capacity and processor speed (for it is on-line to every television set on earth) but its interrogation language is very similar to something we might use today:

```
PROGRAMME REJECTED
Q reason for rejection
ANOMALIES IN GROUND DATA
Q define Q specify
DATA IN FOLLOWING CATEGORIES NOT
ACCEPTABLE. HISTORY COMMERCE SOCIAL
INTERACTION CULTURE
Q accept data as given
QUESTION MEANINGLESS AND INOPERABLE
```

Brunner has obviously gone to some lengths to use nearly-normal English within a context that a computer user would recognise as an operating system response. His other predictions are equally convincing, and it is not surprising that the novel

won most major science fiction awards.

In more recent films and novels, computers have become more than part of the furniture, or even characters, and are now often integral parts of the plot itself. A first-class example is Walt Disney's *Tron*. We talked about this exceptional film (its name is derived from an operating system mnemonic — TRace ON), as an example of computerised animation (see page 181). It takes place both in the real world and *inside* a computer. The outside world has characters such as software engineers, systems programmers and other computer people; but inside the machine the individual parts of the program and operating system become the characters, and the machine architecture is the scenario against which the action is played out.

There are also works of fiction that do not actually mention computers at all, but leave the reader in no doubt that without extremely powerful computing machinery, the situation portrayed could never exist. Foremost among these are George Orwell's *1984* and Aldous Huxley's *Brave New World*. Both of these books are set in their author's future, in a world totally overtaken by a small ruling clique who repress the rest of the population. It is perhaps to these two books that we should look to glimpse the possibilities of misuse of computing power.

It is not possible for us to be entirely exhaustive in this analysis of fictional representations of computers, but some novels display uniquely imaginative ideas. John Barth's *Giles Goat-Boy* is a good example. This is a considerable novel (812 pages in the paperback edition), which centres on the premise that it is the work of a supercomputer called WESCAC, and that it relates an incident that happened to its 'author'.

Finally, we should also remember that it is not only in the realm of fiction that one finds creative writing about computers. Of all the thousands of specialist, non-fiction books that have been written about computers and computing, one stands out from the rest simply for the quality of its narrative. Tracy Kidder's *Soul Of A New Machine* is the history of the development of Data General's Eagle, a 32-bit minicomputer. While the story is ostensibly that of the engineers and managers involved in the project, one is left in no doubt that the star is the computer itself.

Time Out Of Mind
Part vehicle, part artificial intelligence, Dr Who's Tardis falls very firmly into the area of computer as a product of the author's imagination. Though the interrogation method is via a keyboard and visual display unit, the intelligence of the machine itself is assumed to be limitless, as is the contents of its memory

War Games
Unwittingly, a teenage computer user attempting to communicate with a friend over the public telephone network breaks into the main NATO defence computer. Believing what he sees to be a game, he starts to play, only to discover that he has started World War Three...

Original Author

It is possible to write computer programs that will themselves generate other programs, or correct errors in human coding

'If computers are so smart, how come they need humans to program them?' Experienced computer users will tend to shrug off a question like this from a sceptical newcomer, but it is not as silly as it seems. Much research is being undertaken in writing programs that can generate other programs, and operating systems that can correct bugs in code written by humans.

Consider the SYNTAX ERROR? message, which is frequently encountered by home computer users. This can be infuriating because the message gives you so little information. A compiler on a large mainframe computer will generally give far more information as to the nature of the error encountered. For example, the error message could read:

```
1090 LET A=(C*2+F$)*((FG—C)*TH+1))
                ↑                ↑
ERRORS: 1) MISMATCH — STRING VARIABLE F$
           NOT ALLOWED
        2) LAST CLOSE BRACKET NOT EXPECTED
```

There is no fundamental reason why such techniques cannot be used in an interpreter on a home computer — the cost of the extra ROM needed to store the routines would be minimal. But few home computers employ even cursory error monitoring procedures: most don't even check the syntax of the code as it is entered. However, it is often possible to buy additional ROM chips or plug-in software cartridges that will extend the range of BASIC commands available, particularly those related to the development and de-bugging of programs. These BASIC commands include:

HELP — prints out the program line and highlights the exact character position where program execution terminated. This will usually, but not always, indicate the source of the syntax error.

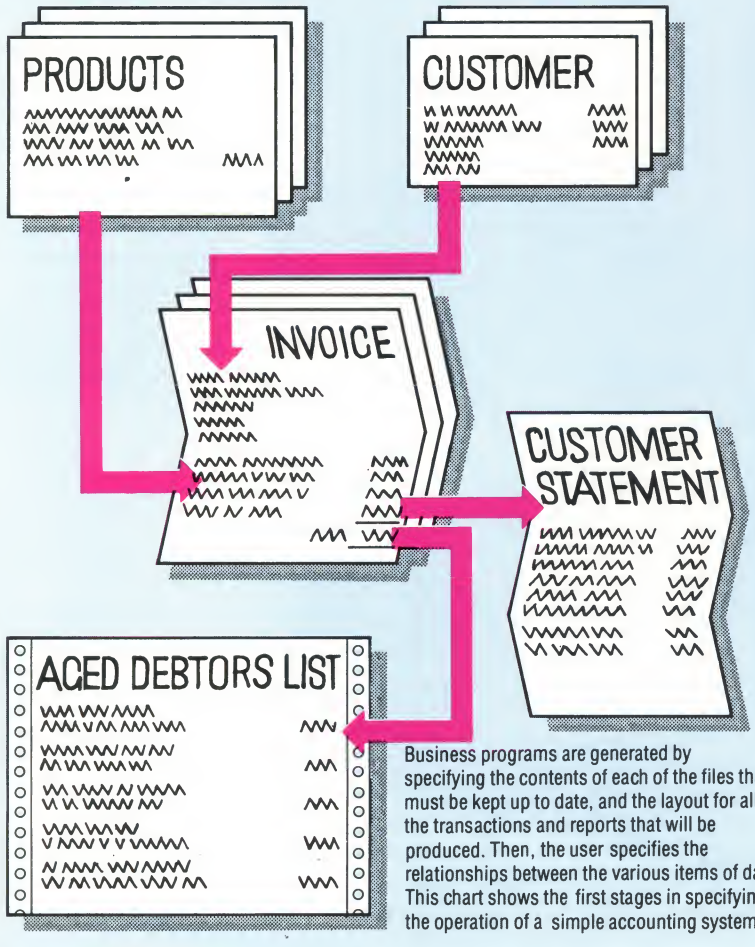
DUMP — prints a list of all variable names and their contents currently in use by the program. This is helpful in deducing how far the program had got in its task before the error occurred.

TRACE — displays (in a window in the corner of the screen) the line number (or numbers) currently being executed while the program is running. This helps the user to trace the flow of the program, and ensures, among other things that subroutines are being executed in the desired order.

Writing programs that allow a computer to correct human coding errors is, in general, not a simple task. But in the case of some errors it is fairly easy. For example, we know that all program lines have to start with a BASIC keyword (though some machines will allow you to drop the word LET). Therefore, if a line begins with PRUNT or PRONT, it would be easy to work out that it should say PRINT. In the Basic Programming course we have discussed the idea of fuzzy matching (algorithms for choosing the closest match to any phrase) and this could be applied to program keywords as well. Alternatively, the interpreter could simply include a list of common typing errors, and their correct equivalents. For safety, it would be desirable for the computer to check any alterations it makes with the operator.

But beyond these basic procedures, automatic correction becomes a great deal more difficult. In

Paper Money



Business programs are generated by specifying the contents of each of the files that must be kept up to date, and the layout for all the transactions and reports that will be produced. Then, the user specifies the relationships between the various items of data. This chart shows the first stages in specifying the operation of a simple accounting system

LIZ DIXON

the example we have given, is F$ a misprint for F or FS or F4? Or something entirely different? If you were to show the complete listing to another competent programmer, he should be able to identify the faults and make the corrections. He would use two criteria in making his decisions: the context in which the program line appeared, and his own experience.

Strangely enough, this technique has been more widely applied to correcting English text than to checking program code. A spelling checker package, for example, will work through a text and highlight any words that don't match the entries in its dictionary of perhaps 50,000 words, held on disk. Most of these packages have the facility to learn new words (such as the spelling of company or proper names) and add these to their dictionaries. The more sophisticated will even suggest the correct spelling if a close match is detected. Experimental word processors have also been developed that can apply the same processes to grammar and writing style — pointing out such things as incorrect punctuation, repetition of words within a paragraph, mixed metaphors, and inapplicable adjectives. Again, these work by examining the context of any phrase, and by reference to a library of previously used examples.

More effort, however, has been put into the development of systems that will *create* programs, rather than *correct* existing ones. In 1981, a software product was announced that set off one of the fiercest battles ever waged within the microcomputer industry. Cleverly named The Last One, it purported to be a program that could write any other program you might want, and hence was the last program you would ever need to buy. This proved to be an unjustified claim, but The Last One was a very useful aid in the development of certain types of program — mainly business applications. There are now several such products on the market for business microcomputers, and a few for home computers, and these are collectively called 'program generators'.

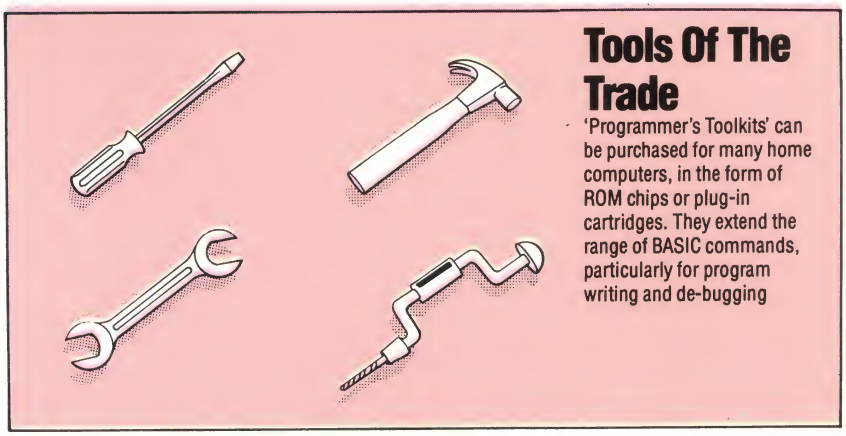
Let's now look at the basic concept behind a program that can write programs. Consider this trivial example:

```
10 PRINT "WHAT DO YOU WANT THE PROGRAM TO
   DISPLAY ON THE SCREEN?"
20 INPUT A$
30 PRINT "THE PROGRAM IS:"
40 PRINT "10 PRINT ";CHR$(34);A$;CHR$(34)
```

If you answer HELLO to the question, the program (which should run on most home computers) should print out the line:

```
THE PROGRAM IS
10 PRINT "HELLO"
```

If you apply the same technique to the input, calculation and output phases of the application you have in mind, then you could write yourself a very simple program generator. If all the questions that it asks are plainly worded, it should be possible for someone with no previous experience to develop a simple program using your generator.



Tools Of The Trade

'Programmer's Toolkits' can be purchased for many home computers, in the form of ROM chips or plug-in cartridges. They extend the range of BASIC commands, particularly for program writing and de-bugging

KEVIN JONES

Commercially produced program generators use the same techniques. Most business applications consist of a combination of five distinct processes: input of data, output to screen or printer, storage in a data file, retrieval, and calculation. The generator will have standard and very flexible subroutines for each of these. By asking you to specify the exact structure of the data you will be using, the calculations that go with that data, and the layouts you require on the screen and printer, the generator will change the values of certain variables in the subroutines, and string them together to create the program.

Although program generators are becoming more sophisticated, they are unlikely to replace human programmers in the immediate future because they suffer from the following limitations. First, the technique described is all very well for transaction-based business applications such as accounting or stock control, but generally these program generators can't be applied to writing word processor or games programs. Secondly, because the program generator has to make use of these standard flexible subroutines, the resulting listing won't be nearly as efficient (either in terms of speed or memory used) as it would have been if it had been purpose-written by a programmer. Thirdly, programs produced by generators generally aren't as user-friendly as the systems currently being produced by human programmers. For example, they seldom make good use of the graphics facilities offered by the latest machines.

Finally, the program generators that are now available can only really replace the final stage in programming — the writing of the code. The user still has to put the work into thinking out the exact form of the data, input and output that is needed. Generally, the earlier stages of programming are the most difficult, and require specific skills distinct from those of programming. Most large companies employ specialists called 'systems analysts' to specify the programs they need, and these specifications are then turned into code by programmers. Program generators have yet to acquire all the skills required to create a computer program.

Over The Rainbow

Some home computers can be expanded into sophisticated machines with a variety of add-ons. We take a Spectrum to its limits

When it was introduced in early 1982, the Sinclair Spectrum was hailed as a real price and performance breakthrough. In 1983, its first full year of production, Spectrum sales (600,000 units) accounted for more than half the number of home computers sold in the United Kingdom, which surprised even the manufacturers. The Spectrum was certainly a vast improvement on Sinclair's earlier ZX81 model, with 16 or 48 Kbytes of RAM as standard; eight colours for border, background and text, and a limited high-resolution graphics capability; an improved — but still ungainly — keyboard; and the ability to generate simple sounds. But all these different facilities did not prevent independent manufacturers from producing a wide range of accessories. Sinclair itself has also been active, adding mass storage, in the form of the Microdrive, and interfaces for plug-in ROM cartridges and joysticks.

TV Monitor
As the Spectrum has no facility to output to a monitor, it is difficult to obtain high quality graphics. Shown here is Sony's Profeel, a modular television system that has its receiver separate from the monitor

Serial Printer
In addition to the ZX printer, it is possible to attach a conventional dot matrix or daisy wheel printer to the Spectrum using an RS232 or Centronics interface card. Sinclair's own Interface 1 supports serial devices

Music Synthesis
Petron's Trichord is a good example of a type of music synthesiser available for the Spectrum. Trichord, as its name suggests, is a three voice unit that offers up to 6,134 three-note chords when used with a 48 Kbyte Spectrum

Cassette Player
Even with the Microdrive fitted, the Spectrum can still write to and read from a domestic cassette recorder, so commercial software issued on cassette can still be used

Speech Synthesis
The Cheetah Sweet Talker is a synthesiser that uses the simplified allophonic system to produce speech by building up words with their component syllables. Sixty-three allophones and four space intervals of varying lengths are provided

Interface 2
A number of independent companies have produced programmable joystick controller interfaces, but Sinclair's own model incorporates a ROM socket, so that games and alternative languages are easier to install and use

ZX Printer
In order to produce a low-cost printer, sufficient for the needs of the home user whose primary requirement is to list programs, Sinclair produced a unit that uses a form of spark erosion to make marks on aluminised paper

Keyboard
Though Spectrum's standard keyboard is a vast improvement over those fitted to the ZX80 and ZX81, it is still not suitable for touch-typing

Interface 1
Specifically designed to serve the Microdrive, Interface 1 also contains circuitry to allow several devices like printers, plotters or modems to be used, and in addition allows a number of Spectrums to be linked together into a local area network

Purpose Designed

Uncommitted Logic Arrays (ULAs) can handle all the functions of a home computer, apart from the CPU, ROM and RAM

Of the many advances in electronic design that have resulted from the microcomputer boom, one of the most significant has been the development of a type of chip called an Uncommitted Logic Array (ULA). Though largely unrecognised by the general public, this quiet revolution has been going on for some years now, to the point where it recently became possible to build very sophisticated computers and other devices with no more than four major components: a CPU, some RAM, some ROM and — to tie these all together — a ULA.

So what is a ULA? As the name suggests, it is a large number (an array) of logic gates, which are initially uncommitted but can be modified to carry out almost any operation that the designer needs. The ULA can be considered as a development of the ROM, since the contents of both of these components can only be specified by the chip manufacturer, not by the user.

Before a ROM or ULA is 'programmed' it consists of no more than a large number of simple electronic circuits or cells, which are not connected and therefore cannot perform any action. All chips are constructed by building up layers of semi-conductor materials (see page 122). The final layer is usually made of conducting material, and forms the connections between the various cells. It is the wide variety of possible interconnections that gives the ULA its flexibility; and though each cell is quite simple, consisting perhaps of a couple of transistors or a single resistor, they can be connected to each other by the final layer to build fairly complex circuits such as flip/flops (see page 305).

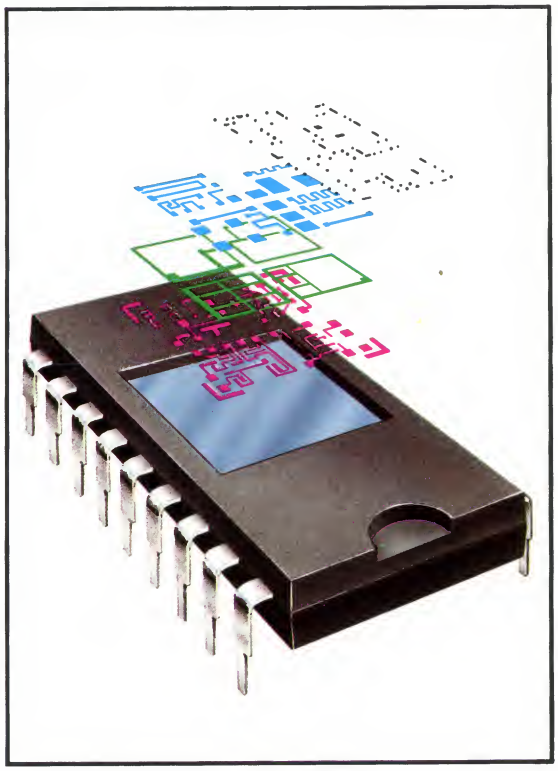
Such circuits, called 'modules', can usually be built from less than half a dozen cells, and since a large ULA may have several thousand cells, the modules themselves can be interconnected to build complex circuits, such as registers, counters, and timing circuits. The functions performed by these circuits are normally carried out in a home computer by a collection of general purpose logic chips.

A ULA can be programmed to perform an extremely diverse range of activities. Any given ULA could be made to synthesise sound, or control the exposure, focus and motor in a camera, or do most of the work in a digital thermometer. And besides the ULA, almost no external circuitry is needed — except for a battery, a switch and some sensors or control buttons.

As might be expected, computers are extensively used in the process of designing the layer that interconnects the cells of a ULA. A mini-computer, such as a DEC PCP11/23, running a Computer Aided Design system, first builds up an encoded diagram of the desired logic. The system then draws, and similarly encodes, a map of the planned layout. This is done on a graphics terminal, and a hard copy of the design can be produced on a plotter.

Once the design is complete it is transmitted to a larger computer, which checks that the plan is acceptable, compares it with the original logic design, and ensures that it doesn't contain any serious errors. It is then submitted to another program that simulates the circuit which would result, using a test program provided by the customer. When the design is finalised, the computer can produce the artwork for the optical mask used in making the final layer.

How far can ULAs go? The idea of putting a lot of simple circuits in silicon and allowing the user to decide how they should interact is so appealing that it might become the recognised method of implementing most circuitry. However, at the present level of technology, ULAs are economic only when at least a couple of thousand identical circuits are needed. The PROM (Programmable Read Only Memory), EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM), and EAROM (Electrically Alterable ROM) are all alternatives to the ROM that can be programmed by a user with suitable equipment. It may not be too long before user-programmable equivalents to the ULA appear, too.



STEVE CROSS

Higher Plane
All semiconductor chips are built up from layers of semiconductor deposits, which are individually etched to create the circuit elements. The final layer determines the connection between elements. A ULA consists of an array of logic elements that can be combined to form a complex logic circuit

Play Acting

Simulation is one of the best educational applications for home computers. We take a further look at some of the programs available

Winds

This simulation program, available from Longmans for the BBC Micro, puts you in the position of master of an old sailing ship. On the screen is a map of the world, with your ship represented by a small dot, and this is steered using compass bearings — N, E, SW, etc. You choose your starting and finishing ports and the date you first set sail. The ship's speed depends on the prevailing wind's direction and strength, and this is shown at the bottom of the screen. Also shown are: the ship's position expressed in longitude and latitude; the wind zone ('westerlies' for example); the date; the distance already covered; and the total length of the voyage.

Let us take the most direct route from London to Rio de Janeiro, and let's set sail on January 1. Heading south, we make good progress using the Atlantic westerlies until we reach the Equator. Here we are becalmed for three days in the 'doldrums'. Eventually, the south-west trade wind blows up, but this poses a problem — how do we sail south-west into a south-westerly wind? The solution is to zigzag (or 'tack') first south and then west, until we reach Rio, 15,480 km (9,620 miles) and 207 days later.

Other voyages are more fraught with danger: hurricanes, polar ice and being shipwrecked are just some of the hazards you could face. This simulation program can be used in a variety of ways. At its simplest it can be used as a game to teach young children the points of the compass. In a school geography lesson, students could investigate the various wind zones and the idea of route optimisation.

Flight Simulation

Flight Simulation is a home computer version of the arcade game, in which you are the pilot of a small aeroplane. The screen displays your view from the cockpit, with the instrument panel below. This contains several dials, gauges and lights, all of which must be closely watched in order to fly the plane successfully. The plane's control column is represented by the four arrow keys, and the other controls (power, landing gear, etc.) are operated by other keys.

Unlike the arcade game, however, you are given the chance to familiarise yourself with the controls by starting with the plane already in the air. To see where the plane is, you can display a map showing the plane's position, navigation beacons and runways. The best way to navigate is to 'lock on' to a certain navigation beacon, and then bank the plane round until it is directly in line with the beacon. This is shown by the flashing dot on the 'RDF Clock', which moves round until it is at the top of the dial. You then fly straight ahead until the beacon is reached. Using this method, it is easy to fly to an airport, where you can attempt to land.

Landing is the most difficult manoeuvre. You need to be directly in line with the runway and approaching at the right speed, height and angle. Even if all these factors are correct, you can still make the simple — but fatal — mistake of forgetting to lower the undercarriage!

Physiological Simulation

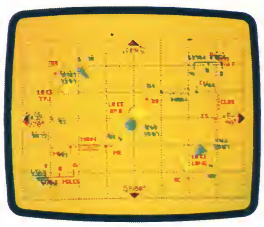
In this program you act as the human brain and your task is to keep the body alive for just 50 minutes! The program simulates the various physiological changes (body temperature, water loss, etc.) that occur when the body performs some activity. The first thing to do is to enter the person's age and sex, and the activity you wish to perform. Sleeping is the easiest activity to simulate, as the body uses little energy. Others include walking, rock-climbing and — most strenuous of all — running.

The parameters that you have to control are: the breath volume, the rate of breathing and the rate of sweating. You then choose some initial values — for example, a breath volume of 2.5 litres, a rate of breathing of 15 breaths per minute, and a sweat rate of three grams per minute — and the simulation begins.

On the screen you see five graphs of the various bodily functions, and a clock. As time progresses, the graphs display how well the body is performing the chosen activity, and you have to prevent any of the graphs from exceeding the danger level. If, for example, the body temperature is getting too high, you can briefly suspend the activity and increase the rate of sweating to try and lower it. If you are unsuccessful and a graph does cross a warning line, you may get the curt diagnosis: 'PERSON IS DEAD'. Heinemann Educational Software produce this program for the BBC Model B.

Winds

Flight Simulation



Physiological Simulation

IAN McKINNELL

Memotech MTX 512

A high standard of construction, and some interesting standard software for handling machine code, are the distinguishing features of this machine

The Memotech MTX 512 comes remarkably close to fulfilling the requirements specified for an 'MSX standard' computer (see page 252); and were it not for the use of Texas Instruments' 76489 sound chip (MSX specifies a General Instruments AY-3-8910), the MTX 512 could claim to be one of these 'standard' machines. It does conform to the MSX specifications, however, in having a Z80 CPU, a Texas Instruments TMS9918/9928 Video Display Processor, and a dialect of BASIC that is acceptably close to the Microsoft version.

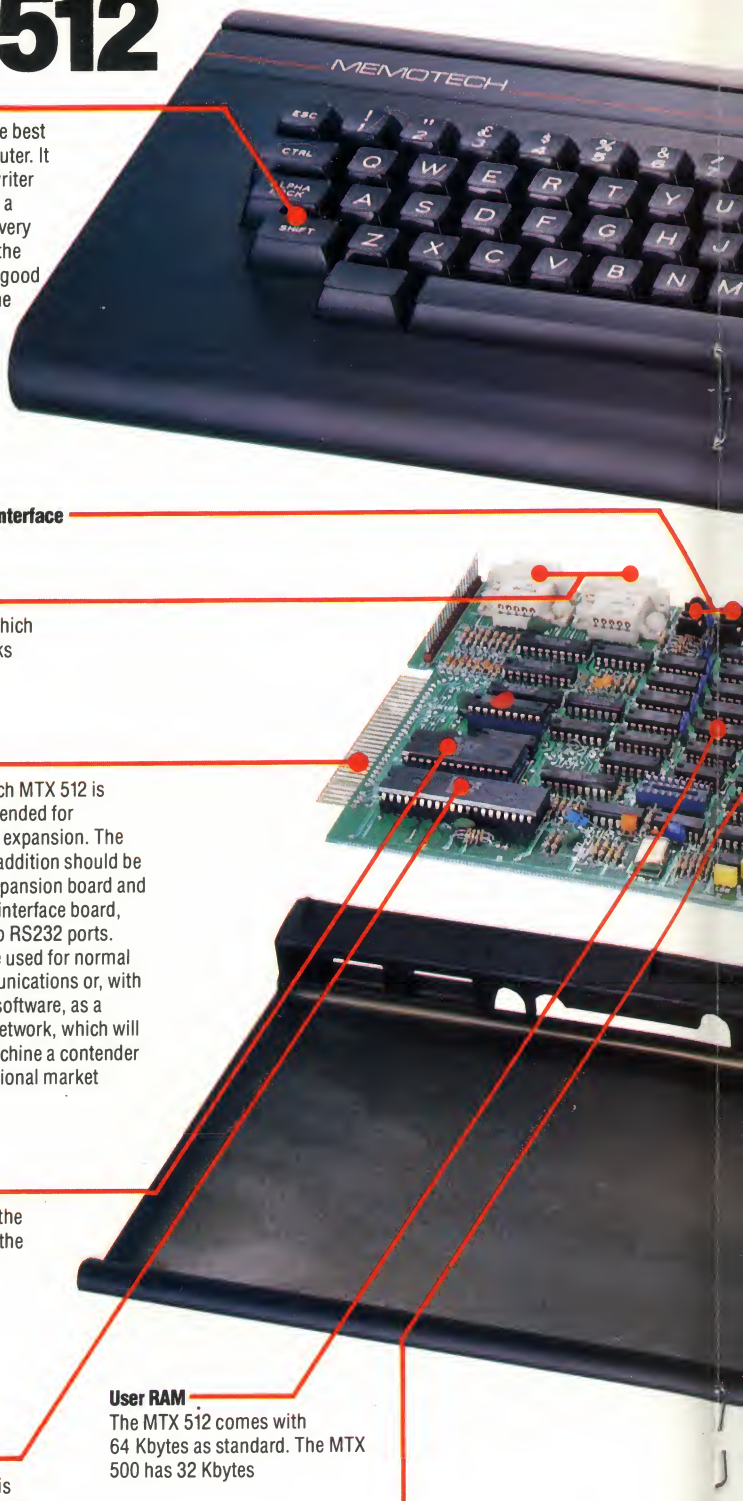
The Memotech MTX 512 is such a comprehensive and elegantly designed machine that it is sure to win many admirers. Its external appearance is a great improvement on many other computers, which often cram sophisticated electronics into a cheap and flimsy casing. The MTX 512, on the other hand, is housed in a black, well-sized and smartly designed casing, constructed from aluminium in a wedge-shaped slab.

The machine is designed to allow easy access to the inside (simply by unscrewing two Allen-key bolts and swinging the bottom casing away) to reveal the circuit board. Compared to other machines, the MTX 512 has a relatively large number of chips. The machine's designers evidently preferred, or found it more economical, to avoid using a few big ULAs. By using a more traditional layout, consisting of many tightly packed chips, the machine facilitates quicker and easier diagnosis of faults. In ULAs, however, faults are very difficult to locate and impossible to repair.

The user manual is not as good as those of other companies and, apart from the covers, it has neither colour nor tints, which would highlight headings and make reference easier. Another drawback is its lack of an index, which makes it difficult to use. However, it is a relatively comprehensive manual. Memotech have decided to make their machine 'open', meaning that they aren't holding any secrets from the purchaser. Information about the machine is presented in great detail: full memory maps, tables of useful locations, input/output addressing, the circuit diagram, and a good introduction to the BASIC language are featured. And specialised chapters on NODDY (see panel), the assembler/disassembler and graphics are also included.

The Memotech MTX 512 is particularly unusual in having an assembler/disassembler that can give, along with the 'Front Panel' software package provided, a machine code programming facility. The assembler package, however, cannot handle symbolic addresses and labels; but provided careful notes are kept while programming, it is quite adequate for moderately sized programs. We will be looking at assembler packages and machine code in more detail later in the course.

The Front Panel is a novel addition to a machine at this level, and is capable of most impressive machine code de-bugging. Unfortunately, this is



Keyboard
The keyboard is among the best ever put on a home computer. It has 79 professional typewriter keys, which are backed by a steel sheet. This makes it very rigid, and combined with the aluminium casing gives a good solid weight to the machine

Cassette Interface

Joystick Connectors
Two ports are provided, which will work with any joysticks using the Atari standard

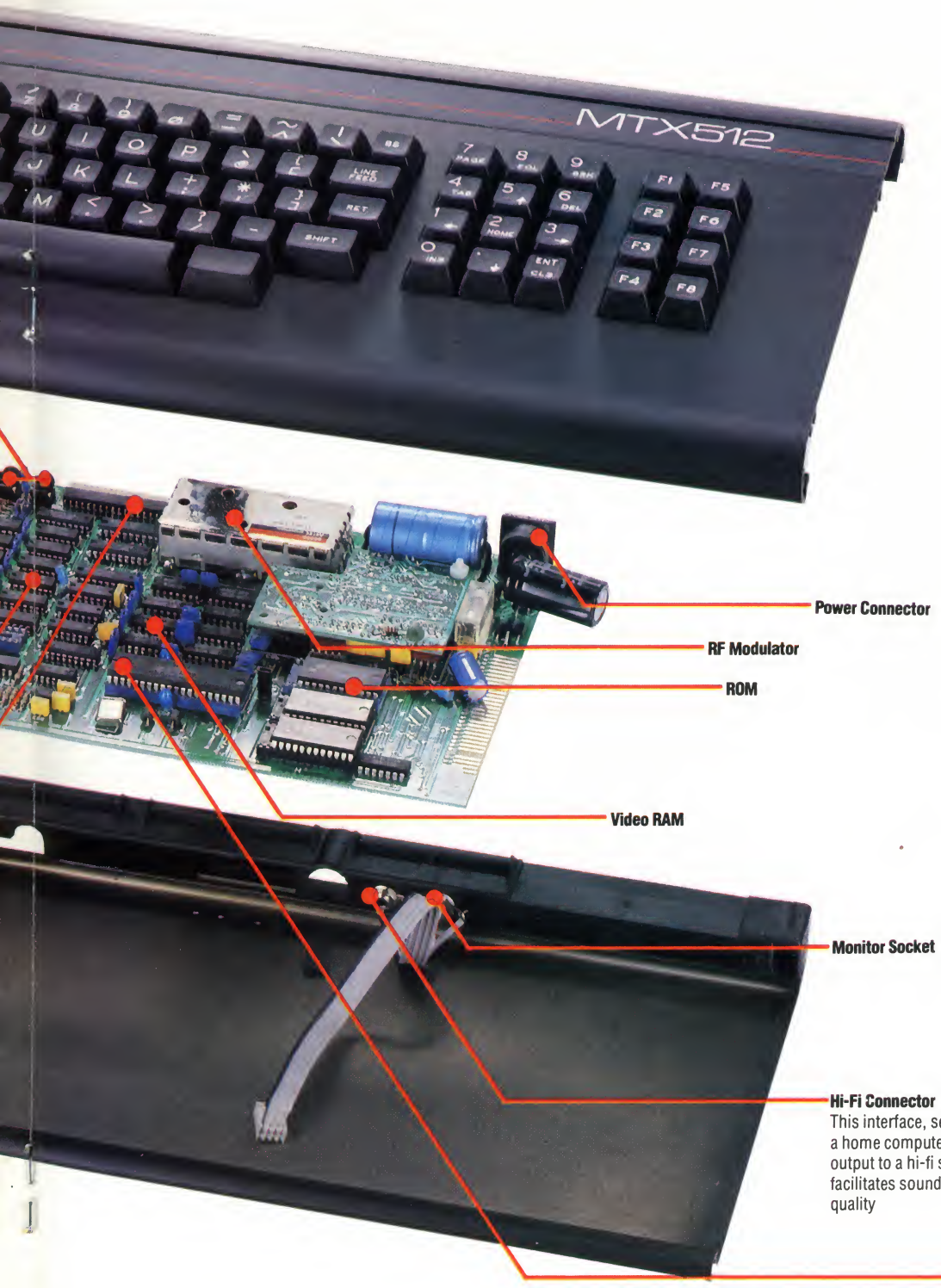
Expansion
The Memotech MTX 512 is obviously intended for considerable expansion. The first serious addition should be a memory expansion board and a dual serial interface board, providing two RS232 ports. These can be used for normal serial communications or, with appropriate software, as a distributed network, which will make the machine a contender in the educational market

Clock Timer Chip
The Z80 CTC provides all the timing functions used by the microprocessor

User RAM
The MTX 512 comes with 64 Kbytes as standard. The MTX 500 has 32 Kbytes

CPU
The Z80 microprocessor is used, at a clock speed of 4MHz

Parallel Interface
This port corresponds to the Centronics standard for parallel interfaces and, together with the RS232 interfaces, allows the MTX 512 to address virtually any printer



MEMOTECH MTX 512

PRICE
£310.00

SIZE
488×202×56mm

CPU
Z80

CLOCK SPEED
4MHz

MEMORY
ROM: 24K
RAM: 64K user RAM, plus 16K video RAM
Expandable to 512K

VIDEO DISPLAY
24 lines of 40 characters, 16 colours with background and foreground independently settable. 127 pre-defined characters and 127 user-definable characters

INTERFACES
Cassette, TV, composite video monitor

LANGUAGE SUPPLIED
BASIC, NODDY, Assembler

OTHER LANGUAGES AVAILABLE
To be announced

COMES WITH
Installation and BASIC manuals, TV lead

KEYBOARD
79 high-quality keys

DOCUMENTATION
Thorough and reasonably complete, but not very interesting to look at. It holds enough information about the internal working of the machine to enable most competent programmers to achieve full control

Hi-Fi Connector
This interface, seldom found on a home computer, enables output to a hi-fi speaker, which facilitates sound of a higher quality

Graphics Chip
This is a Texas Instruments TMS 9928, which controls all aspects of video generation and gives the MTX similar graphics features to the TI99/4A and Sord M5 computers. However, the operating system of the MTX has some useful graphics facilities as well, such as the ability to divide the screen up into several windows

one aspect of the machine that is thinly documented, and though the various commands are listed, little information is given about their functions, and few examples of their use.

The Memotech MTX 512 can be considerably expanded, and with the various extensions that are planned it should become a very capable machine. It will no doubt win many satisfied users and stimulate the development of plenty of supporting software.

NODDY

A subset of the NODDY language is included in the system software and adds a unique dimension to the machine. Being designed as a first-time language for untutored users, NODDY appears to be a very simple language, but on closer inspection it is clear that some of the commands are very sophisticated. It is limited by having only 11 commands, as well as no ability to handle arithmetic. This is because the language is designed principally to handle textual information. Beginners often find it easier to use text rather than numbers as basic data

Sound Spectacular

The Oric-1 permits sophisticated sound control on a budget

MUSIC And PLAY
The following program uses the MUSIC and PLAY commands to play the chord of C major (C,G, & E) using each envelope in turn:

```
10 REM ********
20 REM *CHORD*
30 REM ********
40 MUSIC 1,4,1,0:REM
   *C*
50 MUSIC 2,3,8,0:REM
   *G*
60 MUSIC 3,3,5,0:REM
   *E*
70 FOR E=1TO7:REM
   *SELECT ENV*
80 PLAY 7,0,E,750:REM
   *PLAY CHORD*
90 PLAY 0,0,0,0:REM
   *STOP CHORD*
100 WAIT 50:REM
    *PAUSE*
110 NEXT E:REM *NEXT
    ENV*
```

The Oric-1 is supplied with an extensive range of facilities, and among the more impressive of these are its sound capabilities. It has a range of seven octaves and standard features include three oscillators, a noise generator and seven preset envelopes (see page 276) that can be selected to shape the sounds produced. Sound out is via the built-in speaker.

The Oric-1's BASIC defines a set of sound commands — ZAP, PING, SHOOT and EXPLODE — that describe themselves very well. The following program shows how they are used, as well as demonstrating the useful WAIT command, which causes the computer to pause for the time stated in hundredths of a second (in this case two seconds):

```
10 ZAP:WAIT 200
20 EXPLODE
30 GOTO 10
```

The SOUND command is best used for your own special effects, and is constructed like this:

```
SOUND C,P,V
```

where C=Channel or oscillator number (1–6); P=Pitch (10–5000); and V=Volume (0–15). Channel, set at 1, 2 or 3, selects any one of the three oscillators (4, 5 or 6 are equivalent but select noise as well). Pitch is a little inaccurate, but 10 is the highest note (at approximately 10KHz), and 5000 the lowest (at 100Hz). Volume is highest at 15, but 6 is a comfortable level. If V is set at 0, control is taken by a volume envelope selected by the PLAY command.

The biggest drawback to the SOUND command is that there is no way to set a note duration. This also means that SOUND cannot turn itself off! The only way to stop a note sounding is by using the PLAY command and then stopping PLAY by specifying all zeros.

The MUSIC command is ideal for specifying notes accurately. Its simple construction makes it easy to understand a quite complex music program. The format is as follows:

```
MUSIC C,O,N,V
```

where C= Channel (1, 2 or 3); O= Octave (0-6); N= Note (1-12); and V= Volume (0-15). This command works in a similar way to SOUND. Channel selects oscillators 1, 2 or 3 (although noise cannot be set in MUSIC) and Volume ranges from 0 (where control is taken by the PLAY command) to 15. Octave allows the selection of a specific octave in which the note (N) will be part of the commands, played. Octave set at 0 gives the lowest notes starting at 32.7Hz. Octave 6 extends to 3951.07KHz. For the note (N) part of the command, the numbers 1 to 12 correspond to the standard musical notes in this way:

1	2	3	4	5	6
C	C#	D	D#	E	F
7	**8**	**9**	**10**	**11**	**12**
F#	G	G#	A	A#	B

Light Show

The Spectrum's graphics, though limited, are easy to use

The Spectrum makes an excellent starting point for those who are interested in high resolution graphics and colour. The simplicity of use makes graphics design easily accessible, even for those with limited programming experience.

The normal upper and lower case character sets are available, together with several of Sinclair's own characters. These may be PRINTed in any one of eight colours. Colours for the character, screen and border are set by the appropriately named INK, PAPER and BORDER commands. In addition to the standard character set, up to 21 graphics characters can be defined by the user.

The screen display consists of 24 rows of 32 character spaces. The bottom two rows are, however, reserved for messages from the computer or for keyboard entries. This means that the useable screen is 22 × 32 characters. In high resolution, this converts to 176 × 256 pixels. One extremely useful feature of the Spectrum is that it has the ability to mix high resolution displays with text on the screen, allowing the creation of labelled diagrams, bar charts and so on. Once a screen display has been designed it is then possible to SAVE the display onto tape to be reloaded when required. The SCREEN$ command is responsible for this and can also be used to transfer the contents of the screen to a printer.

Low resolution output may be positioned on the screen using the PRINT AT command, which

To play note A at 440Hz on channel 1 at a volume of 6, the command would be:

```
MUSIC 1,3,10,6
```

However, to achieve the full range of the Oric-1's capabilities, it is best to use MUSIC in conjunction with PLAY. The PLAY command is made up like this:

```
PLAY C,N,E,P
```

where C=Channel (0-7); N=Noise (0-7); E=Envelope (1-7); and P=Envelope Period (0-32767). Channel and Noise select more complex options than the previous commands, in the following ways:

Number	Channel	Noise
0	All Oscs. off	off
1	Osc. 1	+ Osc. 1
2	Osc. 2	+ Osc. 2
3	Oscs. 1 & 2	+ Oscs. 1 or 2
4	Osc. 3	+ Osc. 3
5	Oscs. 1 & 3	+ Oscs. 1 or 3
6	Oscs. 2 & 3	+ Oscs. 2 or 3
7	Oscs. 1, 2 & 3	+ Oscs. 1, 2 or 3

Previously defined MUSIC (or SOUND) commands with volume set at 0 can be PLAYed together, according to the channel number selected, to produce chords of up to three notes. The Noise part of the command selects which oscillators, if any, are to have noise mixed with them. Envelope selects one of seven preset volume envelopes for the specified note or notes. These options are given in the Oric User Manual.

The only variable control over the Envelope is given by the Envelope Period part of the command. This allows the programmer to specify the full duration of an envelope (from 0 to 32767). This varies with each envelope, but as a guide an envelope of 5000 lasts approximately 2 seconds.

The Oric-1's sound commands are easy to use, and show that much thought has gone into providing sensible built-in facilities. The only other home computer that offers helpful BASIC and control over envelope is the BBC Micro, which goes a lot further in the ways it can create a sound. Even so, the low cost of the Oric-1 makes it marvellous value for anyone who wants to make computer music on a small budget.

SOUND
This little program uses SOUND to make a noise like a landing space ship:

```
10 REM **********
20 REM *LANDING*
30 REM **********
40 FOR P=10 TO 3000
   STEP 10
50 SOUND 2,P,6
60 PLAY 2,0,1,1
70 NEXT P
80 WAIT 75
90 PLAY 0,0,0,0
100 END
```

allows vertical as well as horizontal character positioning. There are a number of special effects available also. As well as the usual inverse display, characters can be FLASHing or BRIGHT. A further useful low resolution facility is the OVER command, which allows a second character to be merged with the original in any one character position. This is particularly effective when merging text and a high resolution display, as it is possible to write over diagrams without rubbing them out. This effect must, however, be used with some caution because whenever the INK colour is reset in a particular square the original display also changes to the new colour.

The screen display is governed by two areas of memory: one that displays the characters, and another that holds information about the attributes of any particular character position. The list of attributes includes such information as: the INK and PAPER colours, whether the character is FLASHing, and so on. These attributes are represented by a single byte and the state of any screen position can be interrogated from a BASIC program using the ATTR command.

High resolution displays are easily achieved on the Spectrum using BASIC commands. This is largely due to the fact that there is no separate high resolution screen, making it simple to mix graphics and text into a single display.

BASIC commands include:

```
PLOTx,y
```

This sets the pixel with co-ordinates (x,y) to the current INK colour.

```
DRAWx,y,p
```

As the name suggests, DRAW creates a line between the current cursor position and a point x units to the right and y units up. If a third number is added then the line changes to a circular arc. This third number is normally a fraction of PI (3.14159...). Making the third number PI would cause a semicircle to be drawn; PI/2 would cause a shallower arc to be drawn. Arcs can be made to bend to either side of the straight line between the points by making the third number either positive or negative.

```
CIRCLEx,y,r
```

The CIRCLE command causes a circle, with centre (x,y) and radius r, to be drawn. With most CIRCLE commands in BASIC it is possible to squash the circular shape to form ellipses, but unfortunately the Spectrum does not provide this facility.

There is one main drawback to using colour in high resolution displays. As a consequence of being able to mix text and graphics, only one INK colour may be specified within any one square of eight by eight pixels. Thus, if two lines of different colours cross then, inside the character square where they meet, all set pixels will take on the last INK colour.

Smile Please
This program demonstrates the use of the high resolution commands PLOT, CIRCLE, and DRAW to create a 'smiley' face.

```
10 REM * SMILEY FACE *
20 CLS
30 BORDER 6
40 PAPER 6
50 INK 2
60 CIRCLE 122,88,50
70 CIRCLE 97,108,5
80 CIRCLE 147,108,5
90 PLOT 92,68
100 DRAW 60,0,PI/3
110 STOP
```

Common Sense

Sensors for light, temperature and other effects can all be interfaced to a home computer. The information can be used to control a heating system or burglar alarm

Microprocessors are increasingly used in a variety of domestic appliances, such as washing machines, toasters, video recorders, and central heating units. It is not surprising, therefore, that we can interlink these controlling chips so that devices around the home can share information with one another, or report to a central control system. It is perfectly possible to design and build centralised controlling systems that can regulate domestic appliances. Such systems break down into three categories: dedicated systems, interrupt systems and networked controllers.

Dedicated systems are commercially available, although it is possible for you to design your own devices. These couple to a conventional home computer and use specific interfaces to link directly with, and regulate, electrical or electro-mechanical units such as lights or a thermostat. However, in order to assemble such a system, you will need to have extensive knowledge of computer and electrical hardware and be able to write your own control programs. Dedicated systems also have a fundamental limitation in that the program must run continuously. Any interruption to the power supply will leave the device, at best, locked in a steady state and unable to carry out the adjustments and procedures of the control program.

The second category of control systems use 'interrupts' — electronic signals generated by the regulating devices attached to your central heating, burglar alarm, or fire and smoke detectors. When one of these devices has something out of the ordinary to report to the computer, it sends a priority signal that causes the program in use to be interrupted. Although an interrupt system must run continuously, it is more capable of coping with a breakdown in the power supply because the devices it controls are partially self-regulating anyway.

The computer holds several programs in its memory: one for each of the devices connected to it, as well as the program you are using. Let's say that you are playing an adventure game when the smoke detector triggers an interrupt message. In response to this signal, the computer stops playing the game (preserving all the necessary information about the state of play), and starts running the smoke detector program. A message might appear on the screen saying that a potential fire had been detected; or, if the computer was not being used, an alarm might be sounded. Once the source of the smoke had been detected and dealt with, you could return to the precise point in your game at which you were interrupted. If the signal had come, however, from the central heating timer, the computer would check the time and internal and external temperature sensors, and fire up the boiler accordingly — accomplishing all this so quickly that there would appear to be no interruption to your game!

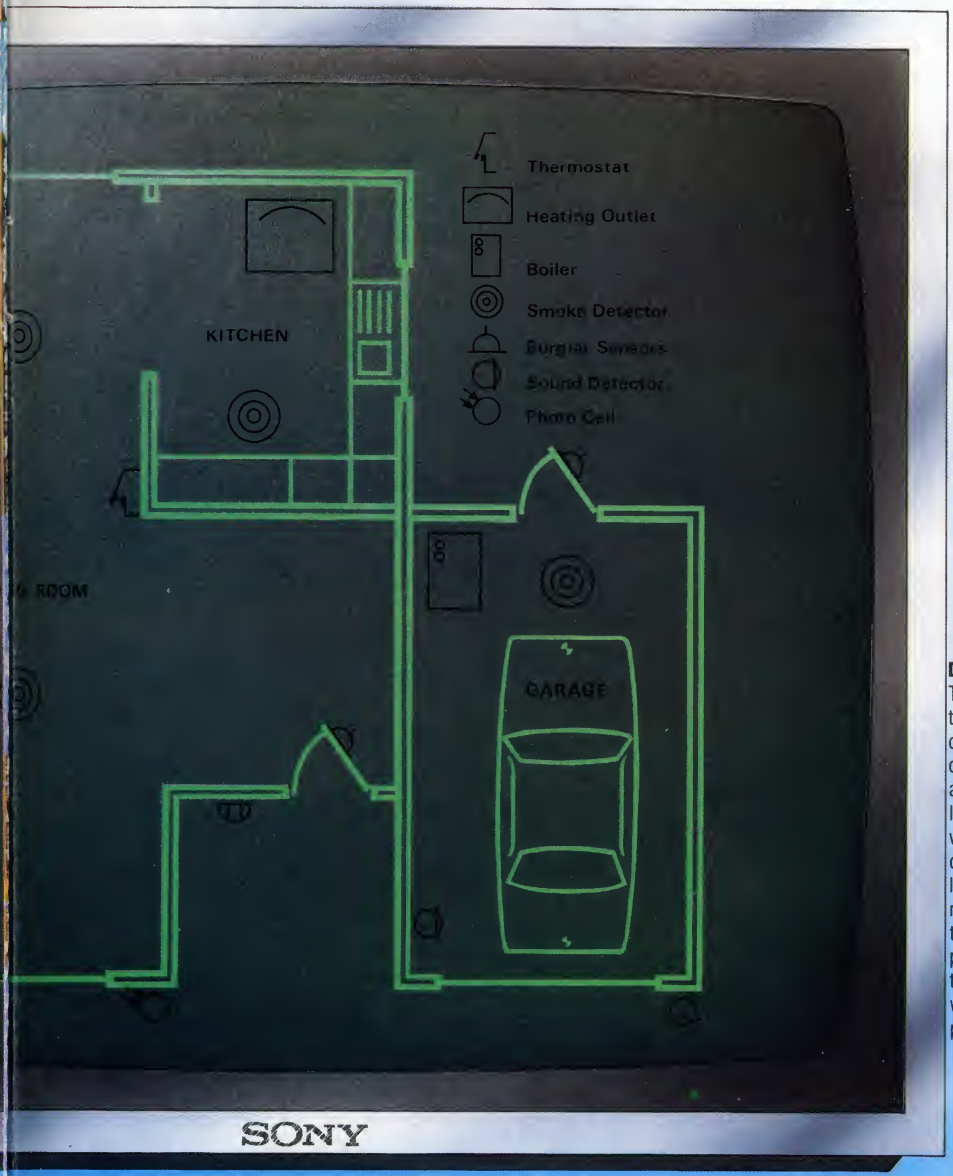
An example of the third category (networked controllers) is a system called the BSR Home Controller. This ingenious device uses the mains wiring in a house to control electrical units plugged into any socket on a particular circuit. Each controller is given a code number (effectively an address), which enables it to be switched on or off by a high frequency signal sent through the mains wiring. However, interfaces of this type are extremely dangerous to install. Only a qualified electrician should attempt to connect outputs from a home computer to the mains.

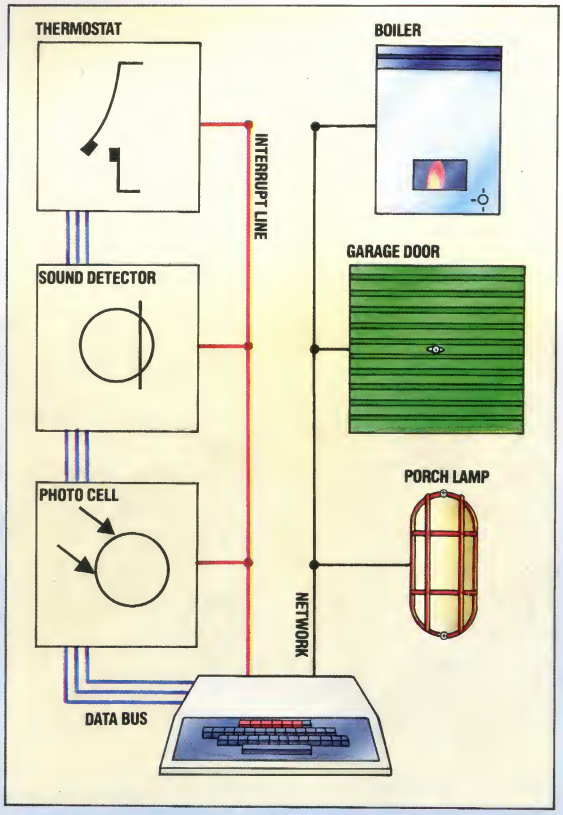
Once a control system has been installed, the next step is to provide some form of remote operation — so that getting to the office and then remembering that the heating is on or the alarm system is disabled is no longer a problem. Any standard communications device, such as a modem, can be fitted to all of these systems, enabling control to be exercised at a remote terminal. Some form of password to gain access becomes a necessity if this facility is provided.

All the systems we have described are commercially available. Sensors can be anything from simple microswitches of the sort used in burglar alarms to the most complex digital thermometer chips. There are several home computers that are capable of supporting this level of expansion, such as the Apple II, Commodore 64, and the BBC Model B; but other machines would require considerable modification to achieve similar results.

The main cost of setting up computer control of your home is incurred in buying the hardware to connect the computer to the mains power — numerous isolators, relays and solid state switches are required to do this safely and effectively. But it is probable that the most demanding task for the home computer user in setting up such a system would be writing the software. Because these systems rely on speed of response (it's no use raising the fire alarm after the house has burnt down), the control programs must be written in machine code. Off-the-shelf programs are not yet on the market, but may be available in the future.

KEVIN JONES





Domestic Address
This diagram illustrates two of the techniques by which a home computer can address a number of domestic appliances. When any of the three sensors on the left has something to report, it will send an electronic pulse down the common interrupt line. This leads directly to the microprocessor, which will temporarily suspend any program it is running and jump to a special routine that will read whatever data the sensor now places on the data bus.

The devices on the right are linked into a network so the computer can activate any of them simply by sending a package of data consisting of, say, the device number of the garage door and the instruction for it to open

Control Centre

A schematic representation of your house displayed on the screen of your computer is not beyond the bounds of possibility. Industrial plant control and computerised security systems use such methods. Of course, if the computer operates under the 'interrupt' method, then there would not necessarily be any need for a screen display, since the software would perform all aspects of control in the background, usually with no noticeable delay to your game of space invaders, or whatever. It may not be many years before houses are designed with built-in internal networks, as commonly as the electrical ring main

Time And Motion

Sorting an array in Basic can be a time-consuming operation, but will ultimately speed up our searches for specific records

So far we have developed most of the code needed to create entries in the address book 'database', but have not yet tackled the necessary programming for saving the entries on tape or disk. The only major omission has been a suitable routine for the creation of the MODFLD$ field, as specified earlier in the course.

The complete program to do this is given in this instalment of the course. First, all characters are converted to upper case (capital letters) in lines 10250 to 10330. Lines 10350 to 10370 then count through the characters in the string and check each one to see whether it is a space. The last space encountered leaves the variable S set to the value corresponding to its position in the string.

Lines 10400 to 10420 transfer characters, one at a time, from the string of upper case characters to CNAM$. Characters are transferred, until we get to the last space, if they have an ASCII value of more than 64. Any characters that fail this test are ignored, so this process eliminates full stops (ASCII 46), apostrophes (ASCII 39), spaces (ASCII 32) and all other punctuation marks. Lines 10450 to 10470 do the same for the characters after the final space, transferring them to SNAM$.

If N$ contains only a single word, TREVANIAN, for example, variable S will be 0 and all the characters will be transferred to SNAM$. The variable used for the forename has been called CNAM$ rather than FNAM$. CNAM$ is used to remind us of 'Christian name', as variables starting with the letters 'FN' will confuse many BASICS into thinking that a call to a user-defined function has been made.

Lines 10490 and 10500 are needed to set the string variables used in this routine to nulls before they are used again. This is a point to watch out for whenever structures of the type LET X$=X$+Y$ are used. Failure to 'clear' the variables will result in more and more unwanted characters accumulating in them each time they are used. Notice that CHOI is set to 0 in the ADDREC routine, since we only want to make sure that the user adds a record if there are none in the file (that is, the first time the program is used).

Now that we have a way of adding as many new records to the file as we want, we need a way of saving the file on tape or disk. The simplest way would be to write all the records to the data file (ADBK.DAT in this version of the program) in the order they happen to be in. The chief disadvantage of this approach is seen when we need to search the file for a particular record. If we cannot be sure that all the records in the file are sorted in some way, the only way to search for a record would be to start at the beginning and examine each record in turn to see if it matches the 'key' of the search. If the record you were searching for happened to be the last one entered, every record in the database would need to be examined before the one you wanted was located. If the last record entered was for a William Brown (i.e. MODFLD$(SIZE-1)="BROWN WILLIAM"), a search routine should anticipate the record to be somewhere near the beginning of the file — if the records had been sorted. Unfortunately, both sorting and searching are very time-consuming activities; so it is a question of determining your priority. We have adopted the principle that an address book is consulted far more often than it is added to (or modified in some other way). This being so, it is better to assume that searches will be far more frequent than sorts, so we will always ensure that the records are sorted before they are stored in the data file after the program has been used.

With this in mind, a variable called RMOD is created to use as a flag. It can have one of two values: 0 or 1. It is initially set to 0 to indicate that no record has been modified during the current execution of the program. Any operation that does modify the file in any way — such as adding a new record — sets RMOD to 1. Operations that 'need to know' if the file has been modified will check the value of RMOD before proceeding. For example, EXPROG, the routine that saves the file and exits from the program, checks RMOD in line 11050. If RMOD=0, no sorting and saving is needed as the data file on tape or disk is assumed to be in a fully sorted and unmodified form. Other routines, such as those that search through the file for a particular record, will also need to check RMOD. If RMOD is 0, the search (or other operation) can proceed. If RMOD is 1, the routine will first have to call the sort routine. After the whole file has been sorted, the sort routine will then reset RMOD to 0.

Our sorting routine, called *SRTREC* in the program listing, resets RMOD to 0 in line 11320 after all the records have been sorted. Before going on to look at *EXPROG* (the routine that saves the file on tape or disk and then ENDs), a few words about *SRTREC* are called for. *SRTREC* is a form of a simple sorting technique called a 'bubble sort' (see page 286). There are many ways of sorting data and the bubble sort is one of the simplest and slowest. A good case could be made for a more

efficient sorting routine, but more sophisticated sorts are much harder to understand than the one we have used. Whether or not you should consider a better sort routine depends on the number of items to be sorted. The 'time complexity' of a bubble sort such as ours is n^2. In other words, the time taken for the data to be sorted increases as the square of the number of items being sorted. If two items took four milliseconds to sort, four items would take 16 milliseconds, 50 items would take two and a half seconds and 1,000 items would take more than 16 minutes. A wait of two or three seconds might be perfectly acceptable during the use of a program like ours, but a wait of a quarter of an hour certainly wouldn't be.

The way this program has been written allows a maximum of only 50 records, so unacceptable delays during sorts should not be a problem. Later in the course, however, we shall outline some of the techniques that can be used to create dynamic files that can grow to almost any size. If you do attempt such a modification to the program, a more advanced sort routine would be one of the first problems to be tackled.

The data items being sorted are the character strings in MODFLD$(L) and MODFLD$(L+1). Records are swapped only if MODFLD$(L) is greater than MODFLD$(L+1), and the index field (which is not being used at present) is updated in lines 11490 and 11570. Every time two records have been swapped, the variable S (to indicate that a swap has taken place) is set to 1. When the sorting routine reaches line 11290 it checks the value of S and branches back to compare all the records again. When all the records are in order, the value of S will be left at 0 and the routine will be terminated after the value of RMOD has been reset to 0.

The EXPROG routine (referred to as *EXPROG* in the program listing) begins at line 11000. It starts by checking to see if any record has been modified during the current execution of the program (line 11050: IF RMOD=0 THEN RETURN). If there has been no modification of the file, there will be no need to save again, so the routine RETURNs to the main program. This will take us back to line 100, which checks the value of CHOI. If CHOI has a value of 9 (as it would if *EXPROG* is being executed) the main program simply goes on to the END statement in line 110.

If the program finds that RMOD is 1 in line 11050 it means that one or more records have been modified in some way and that there is a chance that they are no longer in order. This being so, the *EXPROG* routine calls the sort routine (line 11070) and then, after all the records have been sorted, saves them onto tape or disk.

The save routine (*SAVREC*) is called in line 11090 and the routine starts at line 12000. *SAVREC*, in the main listing, is written in Microsoft BASIC, so it is important to bear in mind that the details of file-handling vary from one version of BASIC to another (see 'Basic Flavours'). Line 12030 opens the ADBK.DAT data file and assigns the channel number #1 for the operation. Line 12050 sets the limits for the loop that counts through all the records in the file. The upper limit is SIZE–1, not SIZE, because the SIZE variable always has a value one greater than the number of valid records in the file (so that if a new record is added, it will not be written over an existing record).

The format of lines 12060 and 12070 is particularly noteworthy. Each field is separated by a ",", which is also sent to the file. This comma is required by most versions of BASIC because INPUT# and PRINT# work in the same way as the ordinary INPUT and PRINT statements. Consider the statement INPUT X,Y,Z. This would expect an input from the keyboard such as 10,12,15<CR>, which would assign 10, 12 and 15 to X, Y and Z respectively. Without the commas, the INPUT statement would not be able to tell where each data item ended and would assign all the data to the first variable. Similarly, the INPUT# statement (in most BASICS) would not be able to tell where each data file record ended and would try to fill each string variable with as much data as could be fitted in. Since in most BASICS string variables can hold up to 255 characters, the data in the data file would soon all be assigned long before the FOR L = 1 TO SIZE–1 loop had terminated. This would result in an INPUT PAST END error message (which indicates that an INPUT statement was issued after all the data has been exhausted) and the string variables (such as NAMFLD$(x)) containing far more data than they should.

Once all the records have been stored in the data file, from L=1 TO SIZE – 1, *SAVREC* RETURNs to line 90 in the main program. Line 100 checks the value of CHOI to see if the last operation was *EXPROG* or not. If it was 9 (save and exit), the program goes on to the END statement in line 110. If CHOI has any other value, the program jumps back to *CHOOSE* and allows the user to select another option again.

As a final footnote, we should mention the *FLSIZE* routine that starts at line 12500. This is offered as a possible alternative to the statement in line 1510. As presented, the program depends on the presence of an end-of-file function: IF EOF(1) = –1 THEN LET L = 50. All BASICS have some way of indicating that the end of a file has been reached, either with a special function such as EOF(x) or a PEEK to a special memory location. The *FLSIZE* routine at line 12500 is offered as a suggestion if an EOF function is not available, in which case line 1510 would need to be replaced by GOSUB 12500.

Basic Flavours

SPECTRUM

Before running the address book program you must create on tape the name-field file. The following program will achieve this.

```
10 REM PROGRAM TO CREATE NFLD FILE ON
   TAPE
20 DIM Z$(1,30)
30 LET Z$(1)="@FIRST"
40 SAVE "NFLD" DATA Z$( )
50 STOP
```

SPECTRUM

When the program stops, rewind the tape and type VERIFY "NFLD" DATA Z$() to check the SAVE.

The following are the Spectrum versions of lines in the main listing. Note that line numbers above 9999, though retained here to relate the changes to the main listing, are not accepted by the Spectrum. Renumber all program lines below this, remembering to modify GOSUBs accordingly.

```
1100 REM *CREARR*
1110 DIM N$(50,30)
1120 DIM M$(50,30)
1130 DIM S$(50,30)
1140 DIM T$(50,15)
1150 DIM C$(50,15)
1160 DIM R$(50,15)
1170 DIM X$(50,30)
1180 DIM B$(30)
1190 DIM Z$(30)
1250 LET Z$="@FIRST"
.
1400 REM *RDINFL* SR
1410 LOAD "NFLD" DATA N$ ( )
1420 IF N$(1)=Z$ THEN LET Q$=Z$:RETURN
1430 LOAD "MFLD" DATA M$( )
1440 LOAD "SFLD" DATA S$( )
1450 LOAD "TFLD" DATA T$( )
1460 LOAD "CFLD" DATA C$( )
1470 LOAD "TELFLD" DATA R$( )
1480 LOAD "NDXFLD" DATA X$( )
1490 REM*FLSIZE*
1500 GOSUB 12500
.
1540 RETURN
.
1640 IF Q$=Z$ THEN LET SIZE=1
.
3520 IF Q$=Z$THEN GOSUB 3860:RETURN
.
3810 LET CHOI=CODE A$-48
.
10090 LET Q$=" "
.
10200 REM*MODNAM* SR
.
10250 LET D$=N$(SIZE):LET P$=" "
10260 FOR L=1 TO LEN (D$)
10270 LET A$=D$(L)
10280 LET T=CODE A$
10290 IF T>=97 THEN LET T=T-32
10300 LET A$=CHR$ T
10310 LET P$=P$+A$
10320 NEXT L
10330 LET D$=P$:LET P$="":LET A$="":LET
      T=LEN(D$)
10340 REM LOCATE LAST SPACE
10350 FOR L= 1 TO T
10360 IF D$(L)=" " THEN LET S=L:LET L=T
10370 NEXT L
10380 REM REMOVE RUBBISH
10390 REM STORE FORENAME IN P$
10400 FOR L=1 TO S-1
10410 IF CODE (D$(L))>64 THEN LET
      P$=P$+D$(L)
10420 NEXT L
10430 REM REMOVE RUBBISH
10440 REM STORE SURNAME IN A$
10450 FOR L=S+1 TO LEN (D$)
10460 IF CODE(D$(L))>64 THEN LET
      A$=A$+D$(L)
10470 NEXT L
10480 LET M$(SIZE)=A$+" "+P$
10490 LET P$="":LET A$=""
10510 RETURN
```

N.B. Because of the way that the Spectrum handles strings, the routine above splits the name at the first, not the last, space.

```
12000 REM *SAVREC* SR
12030 SAVE "NFLD" DATA N$( )
12040 SAVE "MFLD" DATA M$( )
12050 SAVE "SFLD" DATA S$( )
12060 SAVE "TFLD" DATA T$( )
12070 SAVE "CFLD" DATA C$( )
12080 SAVE "TELFLD" DATA R$( )
12090 SAVE "NDXFLD" DATA X$( )
12150 RETURN
12500 REM *FLSIZE* SR
12510 LET SIZE=50
12520 FOR L=1 TO 50
12530 IF N$(L)=B$ THEN LET SIZE=L:LET
      L=50
12540 NEXT L
12560 RETURN
```

ZX 81

Incorporate the Spectrum changes with the following differences (again, line numbers must be reduced below 9999). Add 1255 LET N$(1) = Z$. Delete lines 1410-1540 and 12030-12140 and insert 1410 RETURN and

```
12010 PRINT "INSERT TAPE, PRESS PLAY
      AND RECORD, AND HIT NEWLINE"
12020 INPUT W$
12030 SAVE "ADDBK"
```

Once the program has been saved, execute it thereafter by typing GOTO 40, never by RUN.

EOF

On the Commodore 64 and Vic-20 replace line 1520 by:

```
1520 IF ST AND 64 THEN LET L=50
```

On the Dragon 32 delete line 1520 and replace it by:

```
1485 IF EOF(-1) THEN GOTO 1510
```

On the BBC Micro replace it by:

```
1520 IF EOF# X THEN LET L=50
```

where X is the numerical variable used in the OPENOUT statement (see page 319).

```
10 REM  'MAINPG'
20 REM  *INITIL*
30 GOSUB 1000
40 REM  *GREETS*
50 GOSUB 3000
60 REM  *CHOOSE*
70 GOSUB 3500
80 REM  *EXECUT*
90 GOSUB 4000
100 IF CHOI <> 9 THEN 60
110 END
1000 REM  *INITIL* SUBROUTINE
1010 GOSUB 1100: REM  *CREARR* (CREATE ARRAYS)
     SUBROUTINE
1020 GOSUB 1400: REM  *RDINFL* (READ IN FILE) SUBROUTINE
1030 GOSUB 1600: REM  *SETFLG* (SET FLAGS) SUBROUTINE
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM *CREARR* (CREATE ARRAYS) SUBROUTINE
1110 DIM NAMFLD$(50)
1120 DIM MODFLD$(50)
1130 DIM STRFLD$(50)
1140 DIM TWNFLD$(50)
1150 DIM CNTFLD$(50)
1160 DIM TELFLD$(50)
1170 DIM NDXFLD$(50)
1180 REM
1190 REM
1200 REM
1210 LET SIZE = 0
1220 LET RMOD = 0
1230 LET SVED = 0
1240 LET CURR = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN
1400 REM *RDINFL* SUBROUTINE
1410 OPEN "I",#1,"ADBK.DAT"
1420 INPUT #1,TEST$
1430 IF TEST$ = "@FIRST" THEN GOTO 1540: REM  CLOSE AND
     RETURN
1440 LET NAMFLD$(1) = TEST$
1450 INPUT #1,MODFLD$(1),STRFLD$(1),TWNFLD$(1),CNTFLD$
     (1),TELFLD$(1)
1460 INPUT #1,NDXFLD$(1)
1470 LET SIZE = 2
```

```
1480 FOR L = 2 TO 50
1490 INPUT #1,NAMFLD$(L),MODFLD$(L),STRFLD$(L),TWNFLD$
     (L),CNTFLD$(L)
1500 INPUT #1,TELFLD$(L),NDXFLD$(L)
1510 LET SIZE = SIZE + 1
1520 IF EOF(1) = -1 THEN LET L = 50
1530 NEXT L
1540 CLOSE #1
1550 RETURN
1600 REM  *SETFLG* SUBROUTINE
1610 REM  SETS FLAGS AFTER *RDINFL*
1620 REM
1630 REM
1640 IF TEST$ = "@FIRST" THEN LET SIZE = 1
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
3000 REM  *GREETS* SUBROUTINE
3010 PRINT CHR$(12):REM  CLEAR SCREEN
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT
3060 PRINT TAB(12);"*WELCOME TO THE*"
3070 PRINT TAB(9);"*HOME COMPUTER COURSE*"
3080 PRINT TAB(6);"*COMPUTERISED ADDRESS BOOK*"
3090 PRINT
3100 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
3110 FOR L = 1 TO 1
3120 IF INKEY$ <> " " THEN L = 0
3130 NEXT L
3140 PRINT CHR$(12)
3150 RETURN
3500 REM  *CHOOSE* SUBROUTINE
3510 REM
3520 IF TEST$ = "@FIRST" THEN GOSUB 3860: REM  *FIRSTM*
     SUBROUTINE
3530 IF TEST$ = "@FIRST" THEN RETURN
3540 REM  'CHMENU'
3550 PRINT CHR$(12)
3560 PRINT "SELECT ONE OF THE FOLLOWING"
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. FIND RECORD (FROM NAME)"
3610 PRINT "2. FIND NAMES (FROM INCOMPLETE NAME)"
3620 PRINT "3. FIND RECORDS (FROM TOWN)"
3630 PRINT "4. FIND RECORD (FROM INITIAL)"
3640 PRINT "5. LIST ALL RECORDS"
3650 PRINT "6. ADD NEW RECORD"
3660 PRINT "7. CHANGE RECORD"
3670 PRINT "8. DELETE RECORD"
3680 PRINT "9. EXIT & SAVE"
3690 PRINT
3700 PRINT
3710 REM  'INCHOI'
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 1 TO 1
3760 PRINT "ENTER CHOICE (1 - 9)"
3770 FOR I = 1 TO 1
3780 LET A$ = INKEY$
3790 IF A$ = "" THEN I = 0
3800 NEXT I
3810 LET CHOI = VAL(A$)
3820 IF CHOI <1 THEN L = 0
3830 IF CHOI >9 THEN L = 0
3840 NEXT L
3850 RETURN
3860 REM *FIRSTM* SUBROUTINE (DISPLAY MESSAGE)
3870 LET CHOI = 6
3880 PRINT CHR$(12): REM  CLEAR SCREEN
3890 PRINT
3900 PRINT TAB(8);"THERE ARE NO RECORDS IN"
3910 PRINT TAB(8);"THE FILE. YOU WILL HAVE"
3920 PRINT TAB(6);"TO START BY ADDING A RECORD"
3930 PRINT
3940 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
3950 FOR B = 1 TO 1
3960 IF INKEY$ <> " " THEN B = 0
3970 NEXT B
3980 PRINT CHR$(12): REM CLEAR SCREEN
3990 RETURN
4000 REM  *EXECUT* SUBROUTINE
4010 REM
4020 IF CHOI = 6 THEN GOSUB 10000
4030 REM
4040 REM  1 IS *FNDREC*
4050 REM  2 IS *FNDNMS*
4060 REM  3 IS *FNDTWN*
4070 REM  4 IS *FNDINT*
4080 REM  5 IS *LSTREC*
4090 IF CHOI = 6 THEN GOSUB 10000
4100 REM  7 IS *MODREC*
4110 REM  8 IS *DELREC*
4120 IF CHOI = 9 THEN GOSUB 11000
4130 REM
4140 RETURN
10000 REM *ADDREC* SUBROUTINE
10010 PRINT CHR$(12): REM  CLEAR SCREEN
10020 INPUT "ENTER NAME";NAMFLD$(SIZE)
10030 INPUT "ENTER STREET";STRFLD$(SIZE)
10040 INPUT "ENTER TOWN";TWNFLD$(SIZE)
10050 INPUT "ENTER COUNTY";CNTFLD$(SIZE)
10060 INPUT "ENTER TELEPHONE NUMBER";TELFLD$(SIZE)
10070 LET RMOD = 1: REM  'RECORD MODIFIED' FLAG SET
10080 LET NDXFLD$(SIZE) = STR$(SIZE)
10090 LET TEST$ = ""
10100 GOSUB 10200: REM  *MODNAM*
10110 LET CHOI = 0
10120 LET SIZE = SIZE + 1
10130 REM
10140 REM
10150 RETURN
10200 REM  *MODNAM* ROUTINE
10210 REM  CONVERTS CONTENTS OF NAMFLD$ TO UPPER CASE,
10220 REM  REMOVES RUBBISH, AND STORES IN THE ORDER:
10230 REM  SURNAME+SPACE+FORENAME IN MODFLD$
10240 REM
10250 LET N$ = NAMFLD$(SIZE)
10260 FOR L = 1 TO LEN(N$)
10270 LET TEMP$ = MID$(N$,L,1)
10280 LET T = ASC(TEMP$)
10290 IF T >= 97 THEN T = T - 32
10300 LET TEMP$ = CHR$(T)
10310 LET P$ = P$ + TEMP$
10320 NEXT L
10330 LET N$ = P$
10340 REM  LOCATE LAST SPACE
10350 FOR L = 1 TO LEN(N$)
10360 IF MID$(N$,L,1) = " " THEN S = L
10370 NEXT L
10380 REM  REMOVE RUBBISH AND STORE FORENAME
10390 REM  IN CNAM$
10400 FOR L = 1 TO S - 1
10410 IF ASC(MID$(N$,L,1)) > 64 THEN CNAM$ = CNAM$ +
      MID$(N$,L,1)
10420 NEXT L
10430 REM  REMOVE RUBBISH AND STORE SURNAME
10440 REM  IN SNAM$
10450 FOR L = S + 1 TO LEN(N$)
10460 IF ASC(MID$(N$,L,1)) > 64 THEN SNAM$ = SNAM$ +
      MID$(N$,L,1)
10470 NEXT L
10480 LET MODFLD$(SIZE) = SNAM$ + " " + CNAM$
10490 LET P$ = "": LET N$ = "": LET SNAM$ = "": LET
      CNAM$ = ""
10500 LET P$ = "": LET N$ = "": LET SNAM$ = "": LET
      CNAM$ = ""
10510 RETURN
11000 REM  *EXPROG* SUBROUTINE
11010 REM  SORTS AND SAVES FILE
11020 REM  IF ANY RECORD HAS BEEN
11030 REM  MODIFIED (RMOD = 1)
11040 REM
11050 IF RMOD = 0 THEN RETURN
11060 REM
11070 GOSUB 11200: REM  *SRTREC*
11080 REM
11090 GOSUB 12000: REM  *SAVREC*
11100 RETURN
11200 REM  *SRTREC* SUBROUTINE
11210 REM  SORTS ALL RECORDS BY MODFLD$ INTO
11220 REM  ALPHABETICAL ORDER AND UPDATES NDXFLD
11230 REM
11240 REM
11250 LET S = 0
11260 FOR L = 1 TO SIZE - 2
11270 IF MODFLD$(L) > MODFLD$(L + 1) THEN GOSUB 11350
11280 NEXT L
11290 IF S = 1 THEN 11250
11300 REM
11310 REM
11320 LET RMOD = 0: REM  CLEARS 'RECORD MODIFIED' FLAG
11330 REM
11340 RETURN
11350 REM  *SWPREC* SUBROUTINE
11360 LET TNAMFD$ = NAMFLD$(L)
11370 LET TMODFD$ = MODFLD$(L)
11380 LET TSTRFD$ = STRFLD$(L)
11390 LET TTWNFD$ = TWNFLD$(L)
11400 LET TCNTFD$ = CNTFLD$(L)
11410 LET TTELFD$ = TELFLD$(L)
11420 REM
11430 LET NAMFLD$(L) = NAMFLD$(L + 1)
11440 LET MODFLD$(L) = MODFLD$(L + 1)
11450 LET STRFLD$(L) = STRFLD$(L + 1)
11460 LET TWNFLD$(L) = TWNFLD$(L + 1)
11470 LET CNTFLD$(L) = CNTFLD$(L + 1)
11480 LET TELFLD$(L) = TELFLD$(L + 1)
11490 LET NDXFLD$(L) = STR$(L)
11500 REM
11510 LET NAMFLD$(L + 1) = TNAMFD$
11520 LET MODFLD$(L + 1) = TMODFD$
11530 LET STRFLD$(L + 1) = TSTRFD$
11540 LET TWNFLD$(L + 1) = TTWNFD$
11550 LET CNTFLD$(L + 1) = TCNTFD$
11560 LET TELFLD$(L + 1) = TTELFD$
11570 LET NDXFLD$(L + 1) = STR$(L + 1)
11580 LET S = 1
11590 REM
11600 RETURN
12000 REM  *SAVREC* SUBROUTINE
12010 REM
12020 REM
12030 OPEN "O",#1,"ADBK.DAT"
12040 REM
12050 FOR L = 1 TO SIZE - 1
12060 PRINT #1,NAMFLD$(L);",";MODFLD$(L);",";STRFLD$
      (L);","TWNFLD$(L)
12070 PRINT #1,CNTFLD$(L);",";TELFLD$(L);","NDXFLD$(L)
12080 NEXT L
12090 REM
12100 REM
12110 REM
12120 REM
12130 CLOSE #1
12140 REM
12150 RETURN
12500 REM  *FLSIZE* SUBROUTINE
12510 IF NAMFLD$(L) = "" THEN LET L = 50
12520 IF NAMFLD$(L) = "" THEN RETURN
12530 LET SIZE = SIZE + 1
12540 REM
12550 REM
12560 RETURN
```

Vannevar Bush

The Differential Analyser
This machine was designed to solve an important class of mathematical functions that occur in many areas of science and engineering, known as second order differential equations. The method had first been suggested by Lord Kelvin and involved feeding the output of one 'integrator' (a device that effectively calculates the area under a curve) into the input of another. However, the strength of the output was generally too weak to act as an input and it was not until amplifiers were invented that the method could be applied.

Bush's 1931 machine was entirely mechanical and was a complex structure of gears, axles and electric motors. Input and output were in the form of shaft rotations and the feedback problem was solved by using a 'torque' amplifer.

In the 1940's, a more advanced differential analyser was built using electrical components, but the machine weighed over a hundred tons. The output from the five registers was in a digital printout form and was accurate to one part in 10,000. The initial conditions and control parameters were supplied on punched paper tape. The machine was used throughout the Second World War for cryptography and ballistic work

The differential analyser, designed by Vannevar Bush, was an electromechanical calculator that solved differential equations

Many people argue that Vannevar Bush is the father of the computer. His most important contribution to the development of computer science came in 1931 when he created a mechanical differential analyser, which stimulated research that eventually led to the development of the digital computer

Bush was born near Boston, Massachusetts, on 11 March 1890 and, following in his father's footsteps, studied engineering at college. After graduating in 1913, he worked briefly for the General Electric Company before taking up a junior lectureship at his old college. This was followed by postgraduate studies at Harvard University and the Massachusetts Institute of Technology (MIT). During the First World War, Bush was involved in the development of submarine detectors for the US military.

Bush developed his first invention, a device for surveying land, while he was still a student. The mechanism, which was suspended between two bicycle wheels, calculated the height of the ground over which it travelled and displayed the output as a profile of the land in graph form. It also incorporated a device known as an integrator, since the determination of the height at any position required a knowledge of all the previous values on its journey.

At MIT, Bush became professor of electrical power transmission, and set out to investigate one of the major problems involved in the supply of electricity — how to avoid blackouts that occur as a result of sudden unpredicted surges in demand. The mathematical equations that govern such a situation had been discovered at the end of the 19th century by the Scottish scientist James Clerk Maxwell (1831 – 1879). But there were so many simultaneous equations involved that the problem could not be solved by hand, and so Bush set about inventing a machine for this purpose. Bush was also inspired by the work of Lord Kelvin (1824 – 1907), a British scientist who had proposed a general purpose machine for solving the mathematical equations involved in predicting tides.

In the early 1920's, Bush built his first machine, which he called the 'product telegraph'. This machine enabled human operators to trace the paths of waves drawn on a graph (using a potentiometer — a device that turns a position measurement into a voltage). They then fed these electrical signals into a specially adapted watt meter — the spinning disc found in any power meter, which records the amount of power consumed by integrating the fluctuating values of current and voltage to give the 'product'.

The success of this machine in solving a set of simultaneous equations suggested that it might be possible to build a device that could solve even more difficult second order differential equations. Further research by Bush led to the completion of the first differential analyser in 1931. The machine proved extremely successful, and copies were built in Britain and Europe. In America, the Moore School of Electrical Engineering at the University of Pennsylvania — which was later to build the ENIAC computer (see page 88) — commissioned one. Where Bush's product telegraph had been accurate to only two per cent, the differential analyser gave results accurate to 0.05 per cent. However, the cost of improving the accuracy of this type of mechanical device increased by a factor of ten for every extra decimal place. With the development of the digital computer, however, the cost of a machine only doubled if its accuracy were similarly increased.

Bush became dean of the engineering school and vice president of the Carnegie Institute in 1939, and his able work in administering the millionaire's estate for scientific research resulted in his appointment as chairman of the National Defense Research Committee in the following year. In this office he was responsible for wartime US military research and, in particular, was influential in authorising the Manhattan project, which led to the creation of the atomic bomb. Bush retired in 1955 to devote time to his personal hobbies. These included boating and turkey farming, as well as inventing. He died in 1974.

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.

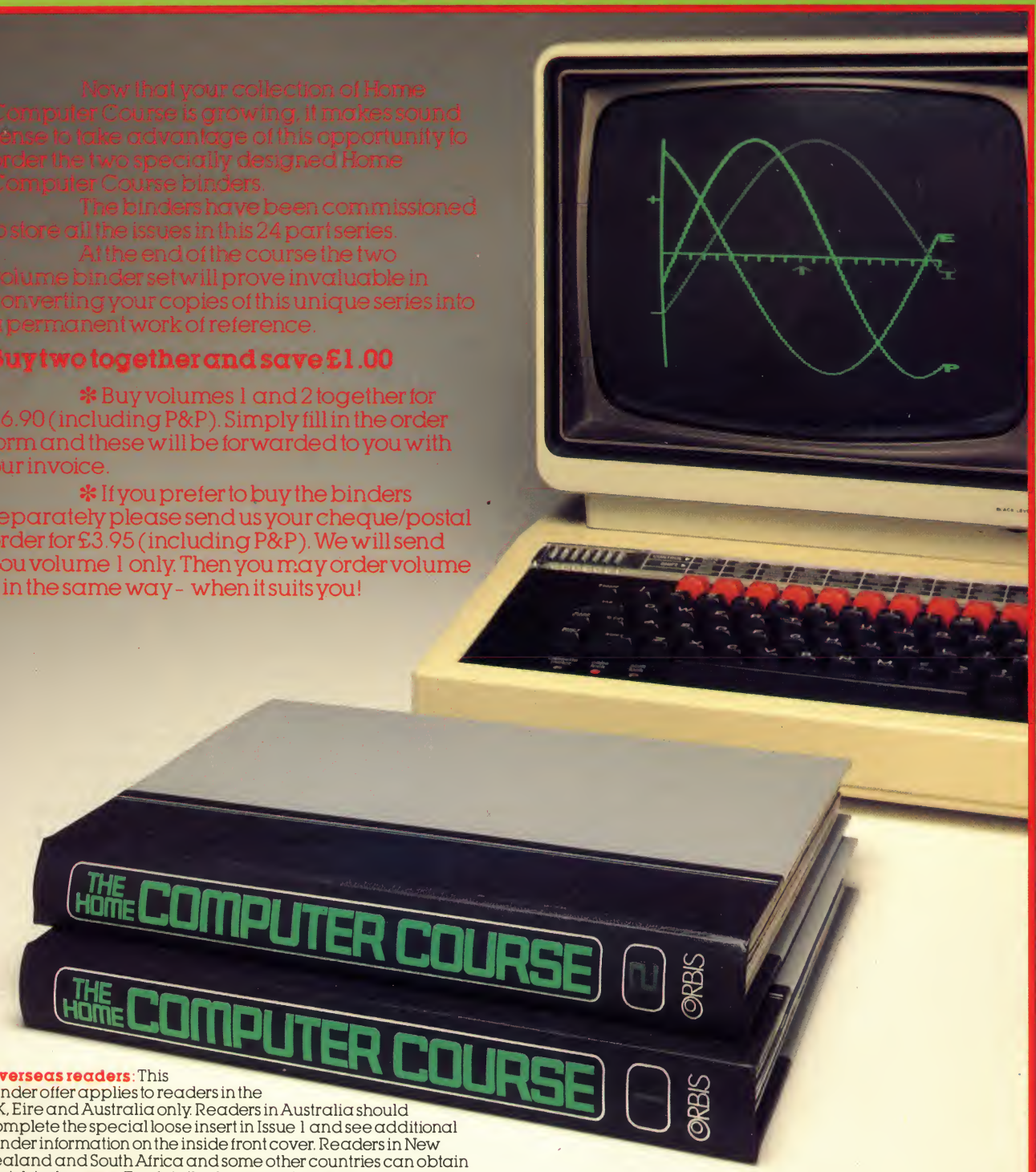
The binders have been commissioned to store all the issues in this 24 part series.

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.

NEXT TO YOUR COMPUTER...YOUR COURSE MANUALS

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