ISSN 0265-2919

R 80p

THE HOME COMPUTER COURSE 21

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

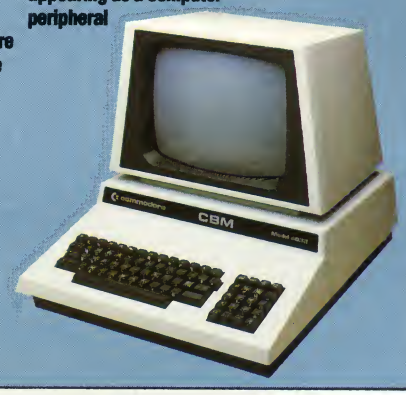
An ORBIS Publication

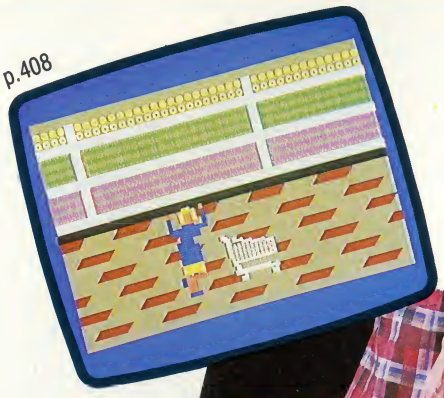
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- We review the Commodore PET, considered by many to be the first personal computer
- Computer Aided Design requires very sophisticated hardware. Many of the software techniques used, however, are now appearing in home computer packages
- Optical disc technology is now used in both video and audio players. Though it is currently a read-only device, it will soon be appearing as a computer peripheral





p.408

p.410

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; **© 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.

COVER PHOTOGRAPHY BY CHRIS STEVENS

TONY SLEEP TAKEN AT HAMLEYS

Young Minds
Children, it seems, are more receptive to the new technology than many adults, who react against the idea of having to learn new ideas. The versatility of the microprocessor means that there is virtually no lower age limit for electronic toys and educational devices

Kids' Stuff

The latest educational 'toys' contain as much processing power as your home computer and use similar programming techniques

In addition to forming the heart of all microcomputers, microprocessors have become a standard feature of many domestic appliances, such as sewing machines, washing machines, and even door locks. Toy manufacturers have experimented with microprocessors as well, especially in the control of model cars and railway trains. However, at least one computer manufacturer — Texas Instruments — has found the production of microprocessor-based teaching toys to be very rewarding. TI's first venture into this market was a calculator-like unit that posed problems in simple arithmetic. The Little Professor proved to be consistently popular, and even when superseded by Speak & Maths, still sold in significant quantities.

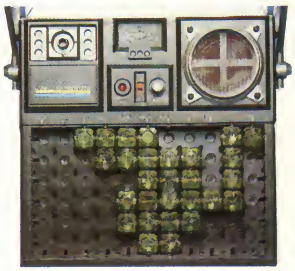
Speak & Maths was the second Texas Instruments educational toy to make use of the TI speech synthesis chip. This was also used in the TI99/4A home computer, which was taken off the market in late 1983 when Texas Instruments decided to withdraw from low-cost domestic computing. Speak & Spell, launched in 1978, has a vocabulary of a few hundred words. The unit has a full alphabetic keyboard (as well as some additional keys) made up of a multi-layer membrane similar to Sinclair's ZX81. On pressing the Enquiry key, the user is audibly prompted to spell a word. Each key depression is displayed by means of Light Emitting Diodes until the word is complete. Speak & Spell then tells the user (audibly) whether or not the spelling is correct.

Speak & Maths works in a similar way, but poses arithmetic problems. These two highly innovative products have succeeded in gaining a large share of the market for educational toys, and have taken Texas Instruments into a field quite different from the standard consumer electronics in which it started.

A third TI speaking toy, Touch & Tell, is perhaps more recreational than educational. It makes use of a number of plastic overlays, each printed with a pattern or picture and uniquely identified by a magnetic encoding. When the child touches an area of the picture, Touch & Tell identifies the selected object audibly.

While synthesised speech is by far the most sophisticated computing technique used by toy and games makers, the most popular application is for small versions of some of the most popular arcade games. There are perhaps as many varieties of this sort of game as there are arcade games proper. Another area where the microcomputer has made an impact on the toy market is in self-guided cars and trucks. Perhaps the best known is the Big Trak, which is programmed by entering instructions on a keypad mounted on its upper surface. The toy resembles a turtle (see page 176), and can also be controlled from a microcomputer, by way of its parallel port.

Other microprocessor-based toys include: Simon, which asks the child to repeat a random sequence of musical notes and flashing lights; Playskool's Maximus, an arithmetic trainer similar to Little Professor; and a variety of robots. Toys for older children (and adults) include: Electroni-Kit, Mykit Systems and Radionics which, as their names suggest, are electronic construction kits that use encapsulated components that can be plugged into a baseboard to make up a variety of simple devices.



Electroni-Kit
As its name implies, Electroni-kit is a construction kit that is used to create a variety of electronic devices, such as transistor radios, amplifiers and so on. Its components are encapsulated in clear plastic, and are plugged into a baseboard (following schematic diagrams supplied with the kit) to build up the desired result. The most sophisticated kit in the range includes the components for a rudimentary microcomputer, which is designed to teach very simple machine code operations. But with only 96 bytes of memory it can hardly be called a home computer

IAN McKINNELL

COURTESY OF PERSONAL COMPUTER WORLD

Maximus
This is a more advanced product available from MB Electronics. While it resembles a calculator, it is in fact another 'matching up' game, like Simon. Maximus, however, has the facility for operating in various modes, which enable the matching of musical notes, pictures, rhyming, spelling, and shapes

COURTESY OF MILTON BRADLEY LTD

Simon
MB Electronics' Simon is a microprocessor-based version of the playground game 'Simon Says'. The unit generates a sequence of musical notes, each one accompanied by a flashing light. The four coloured quadrants on the top surface act as switches for both the tones and the lights. The object of the game is to duplicate the sequence exactly

Texas Instruments
TI entered the educational market in the mid-seventies with The Little Professor, a calculator-like device that posed problems in arithmetic. Before the end of the decade Texas had begun to market a spelling tutor that made use of its speech synthesis chip. Pressing a key results in Speak & Spell asking for a word to be spelled. More recently these techniques have been applied to simple arithmetic games, and very basic storytelling devices for the younger child

COURTESY OF TEXAS INSTRUMENTS

Robo-1
Tomy's Robo-1 is a robot arm of conventional design controlled by means of two joysticks. It is not capable of operating under program control. The startling thing about it is the cost — less than ten per cent of the price of the least expensive teaching robot arm (see page 314). Of course, the construction is much less robust, as injection-moulded plastic is used instead of sheet metal. The arm relies on visual feedback and control by the user, rather than using precise stepper motors. Again, with ingenuity, the Robo-1 could be interfaced to a home computer

TOMY ROBO 1 COURTESY OF HAMLEYS

Big Trak
While it may resemble a Tonka Toy, or one of the other robust toy vehicles for the younger child, Big Trak is in fact a floor robot in disguise. Completely self-contained, it is programmed by entering direction and distance codes on a keypad mounted on its top surface. With a little ingenuity, a conventional home microcomputer can be interfaced with Big Trak via a parallel or serial port. The vehicle could then be guided under program control, which would introduce the possibility of branching into a different sub-program should a particular set of conditions be encountered

COURTESY OF MILTON BRADLEY LTD

IAN McKINNELL

Magic Spell

Spelling checker programs are available for many word processors, and style and grammar checkers are also starting to appear

Computer designers are still a long way off creating machines with the ability to generate and manipulate natural languages, such as English. One of the intended applications for the fifth generation of computers, which should appear in the 1990's, is machine translation between, say, English and Japanese. Machine translation facilities already exist for relatively simple prose such as government reports and proceedings, though the draft produced by the mainframe computer invariably has to be corrected and polished by hand. Stories of errors abound: the quotation 'The spirit is willing but the flesh is weak' is said to have been translated from English to Russian and back again by two different programs, with the final result of 'The wine is agreeable, but the meat is spoiled'!

Such apocryphal stories illustrate a very important point — the difficulties encountered when a computer is processing data without understanding what it means. A problem often posed to students of computer science is to consider how a computer could distinguish between the meanings of the following two sentences:

TIME FLIES LIKE AN ARROW
FRUIT FLIES LIKE A BANANA

The construction of both sentences appears identical, but in the first instance FLIES is a verb, whilst in the second it forms part of a noun phrase. The only reason why we can tell them apart is through experience. It is possible to simulate experience on a computer, given enough memory, but this comes under the field of artificial intelligence, and research in this area is not very advanced. What we are really talking about here is the difference between 'syntax' and 'semantics'. Syntax, meaning the rules concerning the construction processes used in a language, is a fairly easy subject for computers to get to grips with (as all home programmers who have encountered SYNTAX ERROR? messages know). Semantics, however, concerns the meaning which those phrases and constructs convey.

In the 1950's, Noam Chomsky developed the basis for contemporary theory about human languages and the rules of grammar, and although he was not directly involved with the computing sciences, his theories are directly pertinent, both to machine translation and to the writing of interpreters and compilers for programming languages.

One of the by-products of his research has been the creation of various software tools to assist in the writing of text. In addition to word processing packages, which assist in the creation, editing and printing of text, there are programs to proof-read documents for spelling and typing mistakes, and even to check on the grammar and style of writing. Though none of the contemporary products contain anything outstanding in the way of artificial intelligence, it is instructive to look at their operation — both in terms of the way they are presented to the user and how they are internally programmed.

All spelling checker programs make use of a dictionary held on disk, which typically stores between 25,000 and 50,000 words. If you intend buying such a product, incidentally, check that the dictionary has been compiled for English use — many originate in the USA and use American spelling. Most packages will allow you to add items to the dictionary, such as unusual jargon terms that you may use, or the names of companies and products that you wish to have checked.

A problem arises, however, in finding adequate memory space for a full dictionary. You will remember that one eight-bit byte can hold a single alphanumeric character using the ASCII code. So, even allowing an optimistic average of just five characters per word, a 30,000 word dictionary would require 150 Kbytes of storage, which is very much larger than most single disk drives for home computers. Fortunately, this kind of data can be quite easily compressed, by using two techniques.

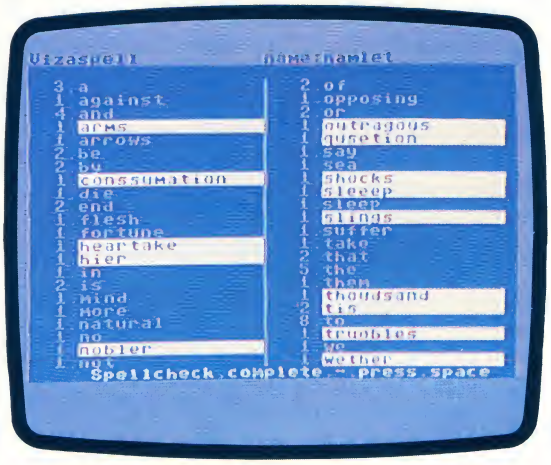
First, if we assume that our dictionary need only contain lower case letters (a routine in the spelling program will handle the conversions), and numeric digits and some punctuation symbols will not be needed, then these can be removed by the program. Subsequently, we could construct our entire dictionary using a maximum of 32 different characters, instead of the full ASCII range of 128 (or 256 if you include graphics symbols). We could therefore reduce the storage requirement of each character from eight to five bits. The word 'computer', for example, could be stored in a total of 40 bits, or five bytes. The first five bits of the first byte would specify the letter 'c', and the next three bits, plus the first two of the second byte, would specify 'o', and so on.

The second technique employed within spelling checkers is called 'tokenising'. This works on the premise that certain combinations of characters appear so frequently that they could be represented as, perhaps, a single byte. This would be 'flagged' in some way to indicate that it was a token for a group of characters and not a single character. Your home computer almost certainly uses tokenising in BASIC — each keyword, like PRINT or NEXT, is stored in RAM as a single byte to save space.

In a spelling checker dictionary, tokenising is used at the front of words. Consider, for example, the large collection of words that begin with auto-, non-, dis- or con-. The VizaSpell package, which runs with the VizaWrite word processor on the Commodore 64, makes use of both compression and tokenising to squeeze a 30,000 word dictionary into a mere 65 Kbytes on disk.

The most dificult task of a spelling checker, however, is to look up all the words from a document in its dictionary. A binary search could be used (see page 416), but for a thousand word document this could take hours. Ideally, the word processor should check each word as it is typed, but this is impractical in programming terms and therefore a document will usually be checked as a whole, either on disk or (on larger machines) in RAM. The program works through the document and compiles a list of the words it contains in alphabetical order. It is not unusual for more than 50 per cent of a large report to be made up from just 100 different words.

Most spelling checkers use this process to provide a useful additional report on the usage of words in your document — which may help you to spot unnecessary repetition. A simple algorithm



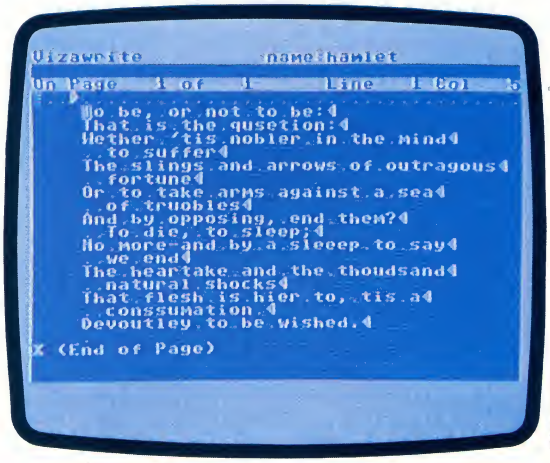
then works its way through this list and the dictionary list simultaneously, looking for matches. In this way, the time taken to complete the search will be greatly reduced and constant — four minutes in the case of VizaSpell, irrespective of the document's length.

Words that are not found in the dictionary will either be printed out as a list, or highlighted within the original document. For each highlighted word, the user is presented with three options:

1) The word has been mis-spelled or mis-typed and should be corrected;
2) The word is correct and should be added to the program's dictionary;
3) The word is correct, but is unlikely to be used again (e.g. it is part of an address), so it should be left alone, and not added to the dictionary.

Grammar and style checkers work in a similar manner. The former work on a limited number of rules (such as looking for a capital letter at the start of every sentence) and, consequently, there are many grammatical inaccuracies that won't be picked up. Style checkers are still in their infancy, and most of the packages currently available simply make use of a large dictionary of examples in order to identify bad syntax and expressions. Generally, these packages will suggest better ways of phrasing the clumsy constructions that they find, by referring to their dictionaries. They will also usually pick up an excessive use of a word phrase within a paragraph, or the use of long and inelegant sentences.

POPPERFOTO



IAN McKINNELL

To Be Or Not To Be
Think how much easier English Literature would be if spelling checker programs were allowed into the exam room! We can use Hamlet's soliloquy to illustrate how one such program (VizaSpell) works. First, the text is typed into the computer using a word processor. Then the spelling checker is invoked with a couple of simple commands, and this creates an alphabetic list of all the words used, also indicating their frequency of use. This list is checked against the dictionary on disk, and unrecognised words are highlighted. When first used, the program may highlight some seemingly common words, but these can be added to the dictionary for later use

Writing a simple form of spelling, grammar or style checker in BASIC can be a very interesting exercise even for an inexperienced programmer — though you will need a fairly good knowledge of the string-handling functions on your machine. As software sophistication increases, it would seem reasonable to expect word processing packages to come with such functions built-in, and more as well. Ah yes, what every writer would adore: 'COMMAND > GENERATE ARTICLE, LENGTH 1200 WORDS, BEGIN'

Application Form

Application generators are similar to automatic program generators, but they have games as well as business applications

IAN McKINNELL

Pinball Construction Set
This package is a kind of application generator for games. The user designs the layout and logic for a pinball game using a menu of objects, and various graphically represented tools to fix them on the board

In the last instalment of THE HOME COMPUTER COURSE, we looked at a type of computer program that would, given a set of specifications by the user, produce a program capable of performing the intended application. Such program generators can be purchased for most business micros, and a few packages are available for home computers, though the type of applications to which they are suited mean that at least one disk drive is mandatory.

A far more common way of generating programs to perform specific requirements involves using packages called 'applications generators'. Unlike program generators, these produce programs that are not free standing, but require the original applications generator package in order to run. Let's consider the creation of a program to handle invoicing using both of these types of generator, in order to highlight their differences.

If we were to use a program generator, the software would first be loaded from disk into the computer. When the user had answered all the questions relating to the files, records, fields, mathematical relationships, screen layouts and printed reports required (i.e. had specified the required applications program), the generator would ask for a blank disk to be inserted into the disk drive. It would then save the new program it had generated on this second disk. This process could be repeated, and a copy of the invoicing program made for each branch of the company.

By contrast, an applications generator initially seems less satisfactory. When you have completed the specification stage, the necessary routines will be recorded on the same disk as the generator. Alternatively, it could record the program on a separate disk, but it would do this in such a way that the original generator disk will still be needed in order to run the application. Although a single copy of the original package could be used to produce an unlimited number of different applications, it follows that they must all be used in the same physical location as the generator disk. If you want to make your application available to others, they will need to purchase a copy of the generator, as well. Of course, such generators employ several methods of program protection, in order to make unauthorised copying very difficult.

An application generator is really just a sophisticated general purpose program. When you specify your application, you are simply assigning values to a number of important variables within the generator, called 'parameters'. These control the flow of the program, the structure of the data, and layouts for screen and printer. When the application is saved on disk, what is actually being stored is a list of these variables or parameters. This list — sometimes referred to as an 'application module' — is therefore just like a set of instructions that tell the application generator how to perform a particular application.

Some packages take this a stage further, and allow you to specify your application in a form of very high level language (similar to the pseudo-language that we first use when developing a new routine in the Basic Programming course). This listing will be interpreted by the generator; and this may in turn be interpreted by the BASIC interpreter if the generating program is written in BASIC, which creates an interesting case of software hierarchy (see page 66).

It is not uncommon for applications modules to be created and marketed by companies other than the authors of the original generator. For example, dBase II (the most popular of the sophisticated database packages available for microcomputers) can really be regarded as an applications generator, containing modules

consisting of strings of high level database commands. Modules for slightly esoteric applications (such as an accounting system dedicated to stockbrokers) can be constructed without having to write the program from scratch. In view of the limited size of the market, a stockbroker package that runs under dBase II may well be better than one written in BASIC, because the program's author will have been able to concentrate all his efforts on the *operation* of the program, rather than the *writing* of the code. The parts of the program most susceptible to bugs (for example, the file handling) will have been written by the generator's authors and tested in different applications by thousands of users.

But the main difference between a program generator and an application generator is in their user-friendliness. The final program created by the former type of package will consist entirely of artificially-written code, which will probably be in a language such as BASIC. Such code will be inferior, both in efficiency and style, to code generated by humans. With the application generator, however, perhaps as much as 99 per cent of the final program will consist of code written by the software house, and this will probably be in machine code as well. This is the case with Silicon Office, one of the most sophisticated and easy to use application generators available for business microcomputers. The resulting program will be faster and more efficient, incorporate checking procedures to detect operator errors, and produce clearly laid-out menu-driven screen displays.

Furthermore, application generators are not restricted to business programs. Perhaps the best example of a non-business package is the Pinball Construction Set (see page 241), in which the application module is effectively specified by laying out the elements of the required pinball table.

There is, in fact, a great deal of overlap between this subject and object oriented programming, which we have discussed before (see page 242), but which may be broadly summarised as: encouraging the programmer to implement his applications purely by specifying the objectives required of the program. Even simple spreadsheet programs, available for home computers such as the Sinclair Spectrum, can be regarded as application generators — you simply specify the relationship between the various fields, and the package does all the routine work for you.

Magpie — produced by Audiogenic for the Commodore 64 with one disk drive — is an application generator that is geared towards business or other serious applications. This is another package that makes good use of visual object oriented programming: relationships between items of data in different records are specified when designing the layout of those records.

Although they are not strictly regarded as application generators, an increasing number of packages are now incorporating some of these principles. When first run, such 'parameter-driven' programs will ask the user a whole series of questions and record the answers on disk alongside the program. This information will determine some of the details of the program's operation. An invoicing program, for example, would ask questions relating to the information that the company likes to have included in each invoice, and the standard credit periods that it allows. An arcade game might ask how many aliens, bases and rockets the user would like to start with, or even give him the opportunity to design the invaders.

Increasingly, software is being designed to protect the user from having to learn programming, while at the same time providing a high degree of flexibility in operation. A situation where the software adjusts itself to fit the user's requirements (rather than the user adjusting to the software) is a highly desirable goal.

COURTESY OF SOFT

Magpie On The Commodore 64

The first stage in creating an application is to specify the layout of all forms, transactions and reports, such as this price list. By filling the columns with letters (A,D,P) the user specifies which fields from the database are to be used

Next, all the calculations and processing are specified in the form of a list of instructions in a high level programming language, to be interpreted by Magpie. Shown here are the routines to amend the prices and to recall (GET) them from disk

Magpie is menu-driven. As any option is selected (e.g. CREATE), another will appear beside it, showing all the CREATE options. This screen displays the result of selecting CREATE, then DISK, then DELETE, then the file to be deleted, in this case PRICE LIST

IAN McKINNELL

Sounds Incredible

The BBC Model B's ENVELOPE command gives almost unlimited control

In an earlier part of the course, the format of the BBC Micro's SOUND command was discussed. However, it is only when it is used in conjunction with the versatile ENVELOPE command that the sound capabilities of the BBC are fully explored. ENVELOPE enables the user to shape up to four sounds to the extent that quite passable emulations of conventional instruments can be programmed. In addition, sound effects for games can be refined to sound much more like the explosions or gunfire that they represent.

ENVELOPE is constructed as follows:

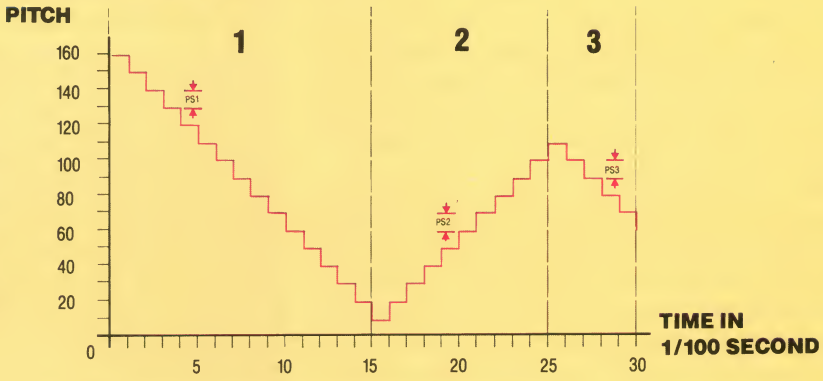
ENVELOPE N,T,PS1,PS2,PS3,NS1,NS2,NS3,
AR,DR,SR,RR,FAL,FDL

The first parameter, N, sets the envelope number and serves to identify the envelope to the related SOUND or SOUND & commands. One of up to four envelopes can be substituted for the fixed volume (V) set with a negative number (0 to −15, see page 388) by SOUND.

T (0 to 127) & (128 to 255)

This is the master timing control for the command. It sets the duration of each 'step' in the construction of the envelope in hundredths of a second. Therefore, T=5 means that each envelope step lasts for five hundredths of a second (0.05 seconds). By adding 128 to the step duration required, the auto-repeat of the pitch envelope will be suppressed, so that T set to 5 + 128 = 133 gives a step duration of five hundredths of a second in a pitch envelope that occurs once within the note.

Pitch Envelope



The use of the term 'pitch envelope' may seem a little confusing as envelope has previously been used in terms of volume, but in this case it refers to the variation of pitch over the duration of a note. This facility has little value in musical terms unless a 'vibrato' is required, but it can be useful to give sound effects interesting 'warbles'. As shown in the diagram, the pitch envelope is divided into three sections. The response of each section can be set by the associated PS and NS number as follows:

Light Relief

Using sprites on the Commodore 64

One of the most exciting features of the Commodore 64 is its ability to use sprites. Sprites are built up in the same way as a user-defined graphic character, but are much larger, consisting of 21 rows of 24 pixels. Sprites are not displayed in the normal character screen matrix and this allows them to be moved a single pixel at a time, rather than requiring eight pixels to move a character from one cell to the next. Up to eight sprites can be displayed at any one time on the screen and each sprite has the following individually programmable characteristics:

Shape And Colour

A sprite is defined in much the same way as an eight by eight pixel character, but 63 bytes are needed to hold the patterns encoded in binary form. Once the shape has been defined in this way, it is held in a block of 63 consecutive locations. Each sprite has a data pointer that points to the area from which the sprite derives its shape. This means that more than one sprite can 'look' at the same area of memory; i.e. sprites can be identical. Also, a sprite can change its shape by switching its pointer to look at a different area of memory.

Each sprite may be coloured in any one of the 16 colours given. Sprites can also be multi-coloured with the usual penalty of halving horizontal resolution.

Size And Movement

Sprites can be expanded horizontally or vertically, or in both directions, to double the original size. A fully expanded sprite is 48 × 42 pixels. Again there

PS1, PS2 & PS3 (−128 to 127)

PS refers to Pitch Step. At the start of the associated note, pitch is set by the SOUND command. PS1 sets the positive or negative change of pitch per step in the first section, PS2 for the second section, and PS3 for the third section. In a similar manner to SOUND, PS is set in quarter semitones.

NS1, NS2 & NS3 (0 to 255)

NS refers to Number of Steps per section; and in conjunction with PS selects the rate at which pitch changes in a section and also the duration of the whole pitch envelope. The PS and NS values for the above example are as follows:

T = 1 PS1 = −10 NS1 = 15
PS2 = +10 NS2 = 10
PS3 = −10 NS3 = 5

In this case, pitch is set by SOUND = 160. This results in:

ENVELOPE 1,1,−10,10,−10,15,10,5,0,0,0,0,0,0

The duration of the envelope is given as (NS1+NS2+NS3)×T, which in this case is (15+10+5)×1 = 0.3 seconds. Normally, the pitch envelope will automatically repeat over the duration of a note unless disabled by the timing parameter, T.

In the next instalment of the Sound And Light course we will return to the sound features of the BBC Micro and explain the operation of the volume envelope.

is a price to be paid, in that resolution is halved in the direction of expansion.

A sprite can move one pixel at a time and the old position is automatically erased. Sprites can also move in and out of the normal viewing area of the screen.

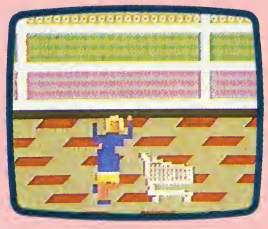
Priority And Collision

When two sprites cross each other's path, one appears to pass in front of the other. If there are any holes in the sprite that is passing in front, the sprite behind will show through. Priority can be used to achieve some interesting three-dimensional effects. Each sprite is given a number from 0 to 7 and the simple rule governing priority is that lower-numbered sprites appear to move in front of higher-numbered ones. Usually, sprites appear to move in front of any normal characters on the screen, but they can be programmed to move behind as well. Again this feature can be used to give the impression of depth on the screen.

When two sprites cross each other this is signalled in a collision register. PEEKing this register can give the programmer details of which sprites have been involved. There is another similar register that signals when a sprite has been in collision with any background characters.

As a consequence of the availability of these features, writing programs to control fast-moving games in BASIC is now possible. Unfortunately, there are no special BASIC commands to control sprite features; everything has to be done by a succession of POKEs into the Commodore 64's memory. An alternative and easier method of creating sprites is to invest in a Simon's BASIC cartridge.

Simon's Basic
For approximately £50, it is possible to purchase a plug-in cartridge to extend the high resolution and sprite handling capabilities available to the BASIC programmer. The cartridge comes complete with a weighty manual detailing the 114 extra commands. These include commands to turn on high resolution mode, select background and foreground colours, and to draw circles, ellipses, rectangles and straight lines. Sprite handling instructions include: assistance with sprite design and creation, commands to switch sprites on and off, and ways of positioning them on the screen



Step Two
These lines may be added to the Supermarket program listing given on page 359. This section of the program uses two expanded, multi-coloured sprites to make up the human figure and a further expanded sprite to make up the shopping trolley. The sprite data pointers are manipulated so that the woman changes shape. This gives the effect of the figure dancing as it crosses the screen. To use the supermarket program, as a subroutine in this program, change line 3270 to read: 3270 RETURN

```
90 REM ** SPRITES 64 **
100 PRINT"□"
110 V=53248
120 REM----READ SPRITE DATA----
130 FORI=12288TO12350:READA:POKEI,A:NEXT
140 FORI=12352TO12414:READA:POKEI,A:NEXT
150 FORI=832TO894:READA:POKEI,A:NEXT
160 FORI=896TO958:READA:POKEI,A:NEXT
170 FORI=12416TO12478:READA:POKEI,A:NEXT
180 REM----EXPAND SPRITES----
190 POKEV+23,7:POKEV+29,7
200 REM----COLOR SPRITES----
210 POKEV+39,10:POKEV+40,10
220 POKEV+41,1
230 REM--MULTI COLOR--
240 POKEV+28,3:POKEV+37,7:POKEV+38,9
300 REM----MEMORY POINTERS----
310 POKE2040,192:POKE2041,193:POKE2042,194
320 REM----SET Y COORDS----
330 Y0=150:Y1=Y0+42:Y2=Y0+34
340 POKEV+1,Y0:POKEV+3,Y1:POKEV+5,Y2
400 REM----TURN ON SPRITES----
410 POKEV+21,7
500 GOSUB3000:REM OMIT IF NO SUBROUTINE
1000 X0=20
1010 POKE2040,13:POKE2041,14
1020 POKEV,X0:POKEV+2,X0:POKEV+4,X0+48
1030 FORI=1TO500:NEXT
1040 POKE2040,192:POKE2041,193
1050 X0=X0+5
1060 POKEV,X0:POKEV+2,X0:POKEV+4,X0+48
1070 FORI=1TO500:NEXT
1080 X0=X0+5
1090 IFX0>200THEN1110
1100 GOTO1010
1110 FORJ=1TO10
1120 POKE2040,13:POKE2041,14
1130 FORI=1TO50:NEXT
1140 POKE2040,192:POKE2041,193
1150 FORI=1TO50:NEXT
1160 NEXT
1170 GOTO1170
9000 REM----DATA WOMAN TOP----
9010 DATA0,0,0,0,21,0,0,21,0,0,22,0,0,86,0
9020 DATA0,86,0,0,86,0,0,40,0,0,252,0
9030 DATA15,255,0,255,255,0,255,255,0
9040 DATA195,255,0,195,255,0,195,243,254
9050 DATA207,243,254
9060 DATA143,240,0,143,252,0,15,252,0
9070 DATA15,252,0,15,252,0
9100 REM----DATA WOMAN BOTTOM----
9110 DATA15,252,0,15,252,0,15,252,0
9120 DATA15,252,0,5,84,0,5,84,0,5,84,0
9130 DATA5,84,0,10,40,0,234,40,0,234,40,0
9140 DATA234,40,0,192,40,0,192,40,0,0,40,0
9150 DATA0,40,0,0,63,0,0,63,0,0,0,0,0,0,0
9160 DATA0,0,0
9200 REM----DATA WOMAN TOP #2----
9210 DATA0,0,0,0,20,32,32,85,32,32,105,48,48,105,48
9220 DATA48,105,48,48,105,48,48,40,48,48,252,48
9230 DATA63,255,240,63,255,240,63,255,0
9240 DATA3,255,0,3,255,0,3,240,0
9250 DATA15,240,0
9260 DATA15,240,0,15,252,0,15,252,0
9270 DATA15,252,0,15,252,0
9300 REM----DATA WOMAN BOTTOM #2----
9310 DATA15,252,0,15,252,0,15,252,0
9320 DATA15,252,0,5,84,0,5,84,0,5,84,0
9330 DATA5,84,0,10,40,0,58,168,0,58,168,0
9340 DATA58,0,0,58,0,0,10,0,0,10,0,0
9350 DATA10,0,0,15,192,0,15,192,0,0,0,0,0,0,0
9360 DATA0,0,0
9400 REM----TROLLEY DATA----
9410 DATA192,0,0,224,0,0,118,0
9420 DATA0,55,192,0,32,60,0,53
9430 DATA87,240,32,0,15,53,85,85
9440 DATA32,0,3,53,85,85,0,0,3
9450 DATA21,85,85,31,255,255,24,0
9460 DATA0,12,0,0,12,0,0,31,255
9470 DATA240,31,255,255,1,0,2,7
9480 DATA0,14,7,0,14
```

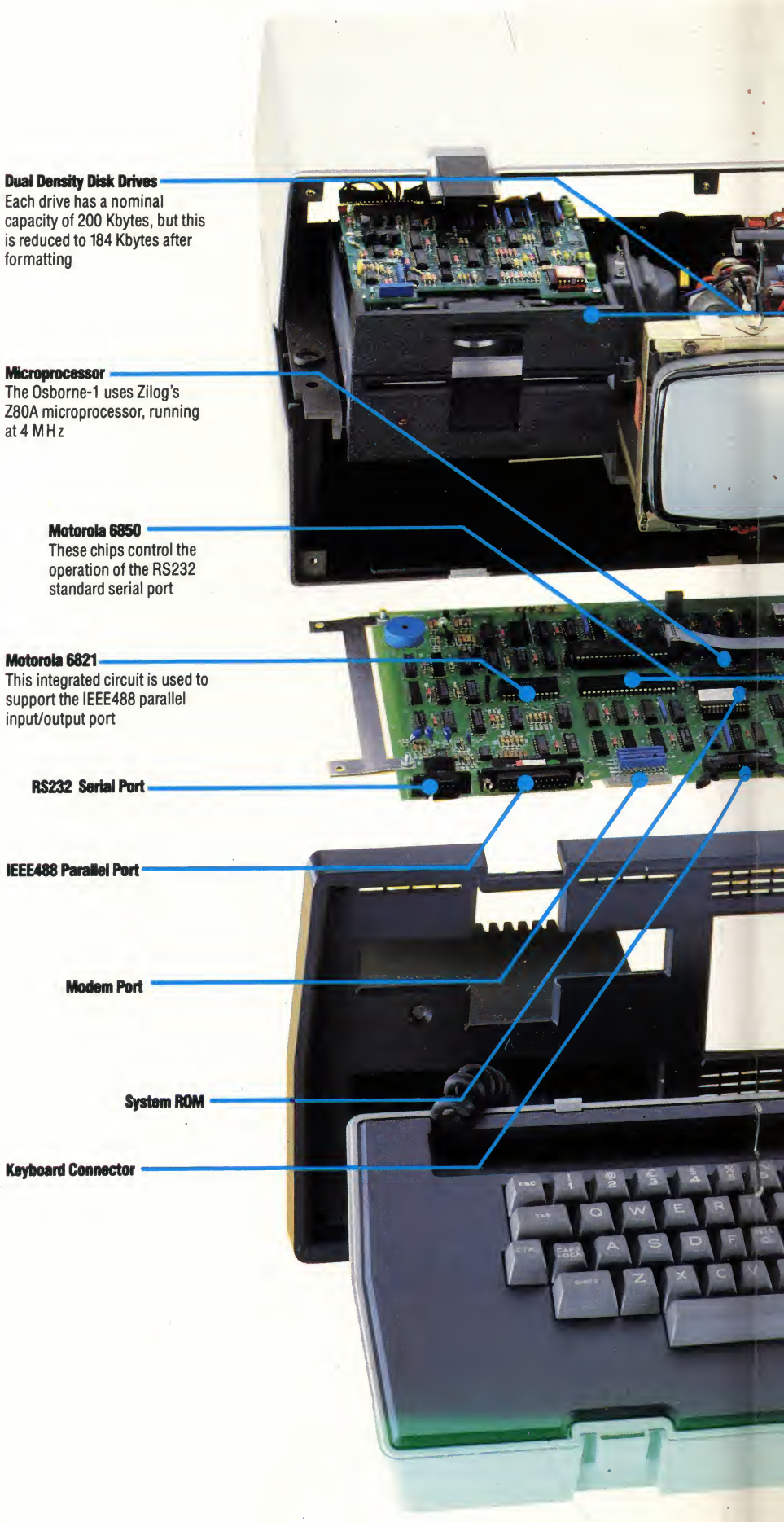
Osborne-1

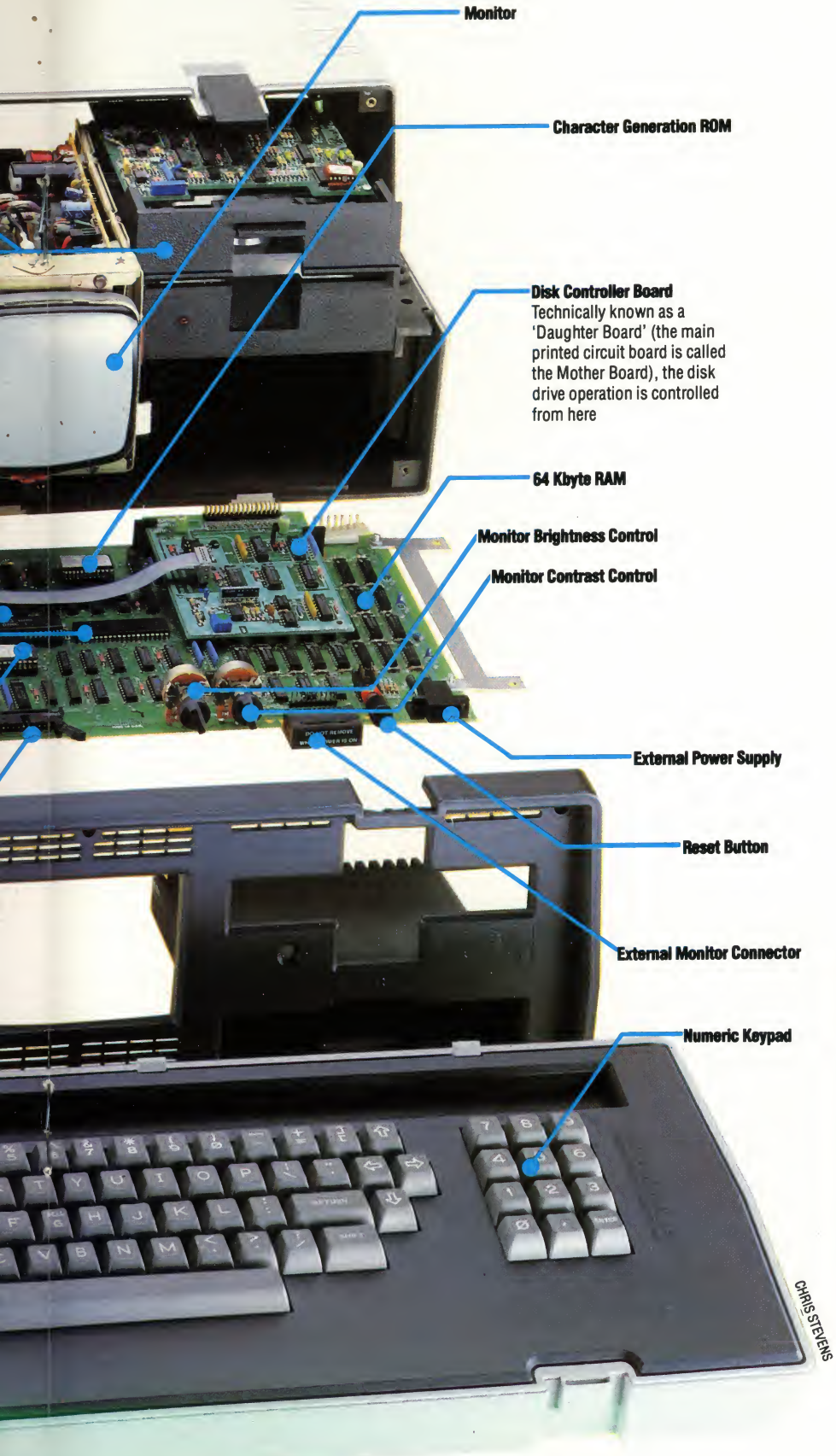
This is the first microcomputer designed to be portable, and the first to be supplied with software included in the price

Although it is not strictly classed as a home computer, the Osborne-1 is a particularly interesting machine because it was the first completely self-contained portable micro-computer. With its two built-in disk drives and small monitor, the Osborne offers its user the ability to carry his own data processing capability with him, wherever he may go. The only thing that the machine lacks is an internal battery pack, but the manufacturer reasoned that this would increase the overall weight of the machine beyond reasonable bounds — it already weighs 10.5kg (23.5lbs). There is, however, a DC socket on the front panel, along with the other interface connections. The machine needs both 12v and 5v inputs: the former for the disk drives, the latter for the logic.

The Osborne's high price — about £1,000 — also makes it hard to class as a home computer — though this does include approximately £600 worth of some of the best established business software available. This includes: Microsoft's CBASIC, a compiled version of the BASIC language, which allows much faster operation of programs; Supercalc, widely acknowledged as the best of the first generation spreadsheet programs; Wordstar and Mailmerge, the best selling of the transportable (not limited to any one type of machine) word processing packages; and, perhaps best of all, the Digital Research CP/M (Control Program/Monitor) operating system, which allows a vast range of software packages to be run on any machine that uses it.

The Osborne-1, in common with the Apple II (see page 349), requires its operating system to be loaded from disk. In addition to overseeing the internal operation of the computer, the CP/M system allows most housekeeping routines — making back-up copies of files and whole disks, initialisation of new disks, cataloguing disk contents, and so on — to be accomplished directly. But the CP/M system has other strengths as well. First of all, software can be written for the operating system, independently of the machines on which it operates. To the software house this means a much larger potential market, hence a great deal more money can be spent on production, which in turn ensures a higher quality package. Secondly, to a skilled CP/M user, the machine type is almost irrelevant, and this allows hardware to be upgraded and enhanced without the onerous task of re-entering data files and converting programs. For a short period, Osborne





OSBORNE-1

PRICE
£945 (£1145 for 80-col version)

SIZE
510×325×225mm

WEIGHT
10.5kg

CPU
Z80A

CLOCK SPEED
4 MHz

MEMORY
64 Kbytes RAM
4 Kbytes ROM

VIDEO DISPLAY
24 rows of 52 characters visible out of an actual 128×32 display

INTERFACES
RS232, IEEE, Modem

LANGUAGE SUPPLIED
BASIC, Z80 Assembler

OTHER LANGUAGES AVAILABLE
Any that will run under CP/M

COMES WITH
CP/M, Wordstar, CBASIC, MBASIC, Mailmerge, Supercalc, Manuals

KEYBOARD
Typewriter-style, 69 keys including numeric keypad

DOCUMENTATION
Adam Osborne sold his publishing company to McGraw-Hill in order to finance the production of Osborne computers, so it's not surprising that the quality of the manual is very high indeed. The only failing is the lack of a comprehensive index

Control Program/Monitor

Mainframe and mini-computers have benefited from the existence of machine-independent operating systems ever since the second generation of machines was introduced in the mid-sixties, but it was to be a dozen years before such control systems were available for microcomputers. Digital Research's CP/M (Control Program/Monitor) was the first of these systems. Designed for Intel's 8080 and the Zilog Z80 series of microprocessors, it has a range of utility and housekeeping programs, and also defines the ways in which running programs may be interrupted and continued.

Another major advantage lies in the definition of file structures and layouts, which the CP/M also handles. Using an interchange program such as BSTAM, which reduces files of any sort to their most basic form, it is possible to transfer programs written for CP/M between machines, irrespective of their type or specification. This means that a huge amount of software is available to the CP/M user

even included in the purchase price Ashton-Tate's dBase II — the most powerful of all the microcomputer-based database management programs, which normally sells for £350.

Unfortunately for Osborne, most of the US business community concentrated its attention on the IBM Personal Computer, a 16-bit machine based on Intel's 8088 microprocessor. Intended as an interim solution — it boasts 16 bit addressing, but only eight bit data transfer — the 8088 became a de facto industry standard simply by virtue of IBM choosing it to power its first entry into the microcomputer market.

The IBM PC uses a specially devised operating system called PC-DOS. In an effort to compete, Digital Research launched two new versions of the CP/M operating system: Concurrent CP/M, which allows true multi-user multi-programming; and CP/M86, designed for Intel's 8086 chip, which incorporated 16-bit addressing and 16-bit data transfer.

Unfortunately, all these developments came too late to prevent the Osborne-1 from being swamped by market forces, and in 1983 the Osborne Computer Corporation — the parent company in the United States — went into voluntary liquidation. With its 64 Kbyte memory (60 Kbytes available to the user) and twin 183 Kbyte disk drives, the Osborne-1 is still a reasonably powerful computing machine. Add to that its built-in RS232 and IEEE ports, the modem port and its ability to run from a battery pack, and it's easy to see why the computer was an instant best seller, and why it is still popular with users even after the demise of its manufacturer.

One very interesting feature of the Osborne-1, which is shared to some degree by Epson's HX-20 (see page 169), is the provision of a 'virtual screen' more than three times as large as the 52 column by 24 row display provided. The use of the control key (a standard CP/M requirement) and the cursor keys allows the display to move around the actual screen memory. To a great extent this removes most of the disadvantages imposed by the small physical size (8.75 × 6.6 centimetres, 3.5 × 2.6 inches) of the screen, although non-users often express surprise that a display whose characters are a mere two millimetres (1/10 inch) high should be legible, let alone comfortable to use.

In fact, few users are unable to come to terms with this miniaturisation, although Osborne did provide an external monitor connector that duplicates the contents of the small screen on a larger additional unit. Indeed, far from considering the character size to be too small, there was an appreciable demand from users for the entire four Kbyte virtual screen (128 columns by 32 rows) to be displayed at all times, and Osborne manufactured a modification to just that specification. This allows users to choose one of three screen 'widths': 52 characters, 96 characters or the full 128, and even at the highest density the characters are still well-defined and readable.

The Osborne-1's keyboard, which clips on to the computer's front panel as a 'lid', rendering it weatherproof, is a 69 key unit. It has normal typewriter-style keys, with the addition of Control and Escape keys, plus a 12-key numeric pad on the right-hand side that includes extra full stop and enter keys. Using a CP/M program called SETUP, the functions of the numeric keys (when used in conjunction with the Control key) can be user-defined to a maximum of 96 characters. This feature is particularly useful if a word, phrase or command string is to be used frequently. The results of the SETUP program are written on each disk, rather than being stored in memory, so the functions can be pre-programmed separately for each different software package. The computer automatically sets its functions each time the operating system is loaded.

In addition to the standard 96 upper and lower case characters, there are 32 pre-defined graphics characters available, though these can only be accessed through an applications program.

Because the Osborne-1 uses 6800 series support chips, rather than their counterparts from the 8080 family (as one would expect of a CP/M machine), the keyboard polling method is slightly different. There is a portion of memory set aside to interpret key depressions, and the system ROM continually checks to see if a key has been depressed. There is no decoding logic in the keyboard itself. It is this implementation which allows easy programming of the function keys, and because these functions are stored in the Random Access Memory, they can be accessed and changed from within a program.

Though the Osborne Computer Corporation went into voluntary liquidation, the British division set up as a separate company and continued to trade. Whatever the future holds for this machine, its quality is undeniable.

Carried Away
In addition to its excellent price/performance ratio, the Osborne-1 has the added advantage of portability. At 10.5kg it is no lightweight, but it is completely self-contained and well balanced, so carrying it about is not too much of a problem.
One constraint on its physical dimensions was that it should fit under the seat of an aeroplane

IAN McKINNELL

Sorting Code

The Shell Sort is more efficient than either the Bubble or Insertion Sorts for long arrays. It works by dividing the data into a series of 'chains'

On page 286 we looked at two methods of sorting an array into order — the Bubble and Insertion Sorts. Generally, the Bubble Sort is easier to carry out, but the Insertion Sort is faster. Experience of these two methods shows that what takes the time is swapping cards around over short distances: it's usually far better to swap once over a long distance than several times over short distances.

```
7999 REM**********************
8000 REM*       SHELL        *
8001 REM**********************
8025 PRINT "SHELL SORT - GO !!!!!"
8050 LET LK=LT
8100 FOR Z=0 TO I STEP 0
8150 LET LK=INT(LK/II)
8200 FOR LB=I TO LK
8250 LET LL=LB+LK
8300 FOR P=LL TO LT STEP LK
8350 LET D=R(P)
8400 FOR Q=P TO LL STEP-LK
8450 LET R(Q)=R(Q-LK)
8500 IF D<=R(Q) THEN LET R(Q)=D:LET Q=LL
8550 NEXT Q
8600 IF D>R(LB) THEN LET R(LB)=D
8650 NEXT P
8700 NEXT LB
8750 IF LK=I THEN LET Z=I
8800 NEXT Z
8850 PRINT "SHELL SORT - STOP !!!!!"
8900 RETURN
```

To add this routine to the sorting demonstration program on page 287, change line 350 to:

```
350 LET I=1:LET 0=0:LET II=+:LET TH=3
```

and change line 900 to:

```
900 ON SR GOSUB 6000,7000,8000
```

A better method than either of these two is called the 'Shell Sort' (named after its creator, D Shell). This method ensures that the disorder in the array is reduced early in the sort (so that items are not a long way from their true positions), and enables swaps to operate over relatively long distances. Here is a method for this sort:

1) Lay out all the cards of one suit in any order. They are to be sorted into descending order so that the King will be the leftmost card, and the Ace the rightmost. Count the cards, divide that number (in this case, 13) by two, ignoring any remainder, and write the result (i.e. six) on a piece of paper labelled 'The Link'.

2) Place a fivepence piece under the leftmost card (call this Position One), and a tenpence piece in the Link position (i.e. Position Six in the first instance). All of the cards from the First to the Link position are each to be the leftmost end cards in a series of 'chains' of cards. The number of chains will equal the current value of the Link. Each chain is formed by starting with its end card, adding the Link to the end card's position number to get the position of the next card, adding the Link to get the position of the next card, and so on until the end of the array has been reached or exceeded. The first chain, therefore, comprises the cards in positions One, Seven, and Thirteen; the second chain is the cards in positions Two and Eight; the third is the cards in positions Three and Nine. The last chain is the cards in positions Six (the present value of the Link) and Twelve.

3) Now, having marked the boundaries with the five- and ten pence pieces, push the cards that comprise the first chain out of the array so that you can see them in isolation, and sort them into order using either the Bubble or Insertion Sort as described on page 286 (the listing with this article uses the Insertion method).

4) Push the ordered chain back into the gaps in the array, and repeat the above with the next chain, and the next, and so on, until all the chains whose leftmost cards lie between the five- and ten-pence pieces have been sorted.

5) When all the chains have been sorted, divide the Link by two, ignoring any remainder. If the Link is now less than one then the array will be sorted. Otherwise, repeat from Step Two above with the new value of the Link.

Shell Sort Panel

Position No.	Link Value	Comments
1 2 3 4 5 6 7 8 9		
2 8 9 3 T 5 K 6 7	(9/2)=>4	Begin Pass
* + @ $ * + @ $ *		Form chains
T 7 2		Sort Chain 1
8 5		Sort Chain 2
K 9		Sort Chain 3
6 3		Sort Chain 4
T 8 K 6 7 5 9 3 2		Begin Pass
T 8 K 6 7 5 9 3 2	(4/2)=>2	End of Pass
* + * + * + * + *		Form Chains
K T 9 7 2		Sort Chain 1
8 6 5 3		Sort Chain 2
K 8 T 6 9 5 7 3 2		End of Pass
K 8 T 6 9 5 7 3 2	(2/2)=>1	Begin Pass
* * * * * * * * *		Form Chain 1
K T 9 8 7 6 5 3 2		End of Pass

KEY	
*	Member of Chain 1
+	Member of Chain 2
@	Member of Chain 3
$	Member of Chain 4

Shell Sort
The example of the Shell Sort for a reduced hand that we show in the panel demonstrates its unique method of dividing the array into a series of chains (with spacings based on the current Link number). These chains are separately sorted, in this case using the Insertion method, before a pass is completed.

The program listing for a Shell Sort given here must be used in conjunction with the testbed program on page 287. When we tested it, there was a significant improvement over the other sorting methods once the number of items to be sorted exceeded 40

Single Handed

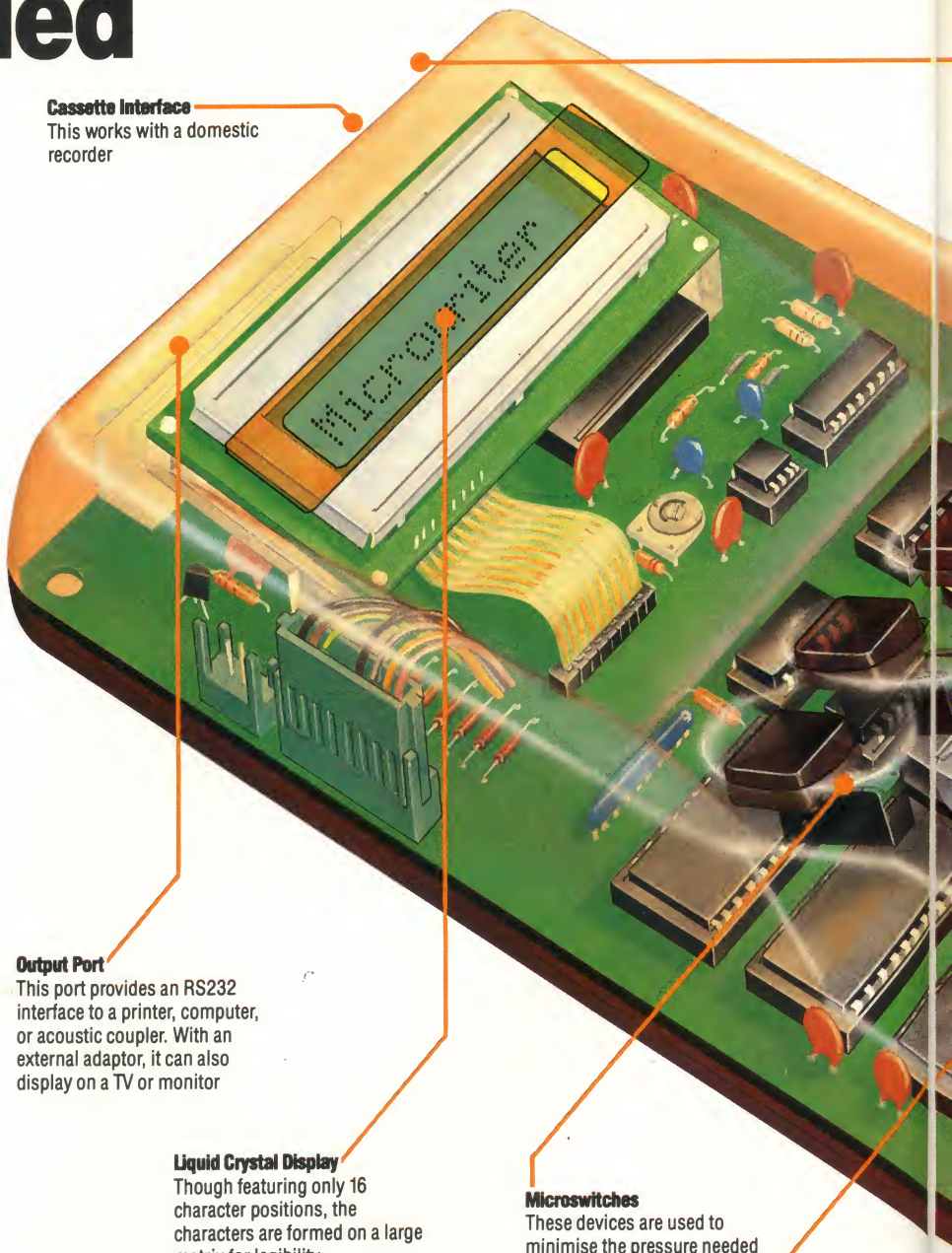
The Microwriter is a portable word processor that can be operated with only one hand. The six-button keyboard may soon be used on computers

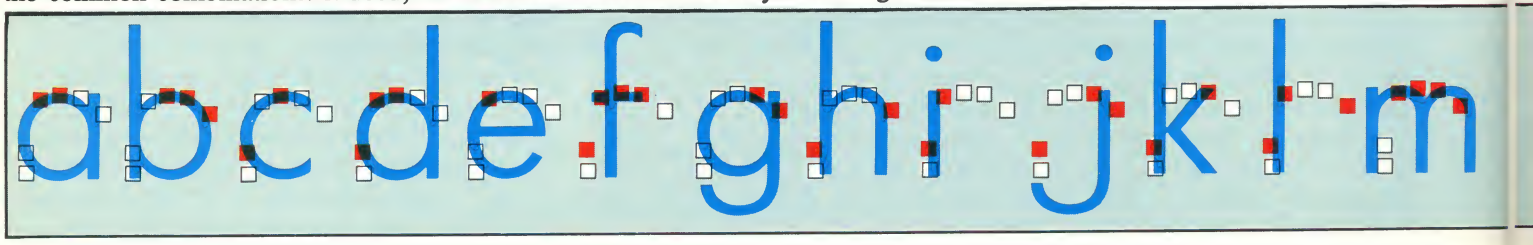
Having a word processor at the office, or a home computer with a word processing program, can be an excellent idea. Apart from taking the drudgery out of producing routine paperwork and letters, they can help with program documentation, quickly produce copies of notices, or handle the contents of an address book. Indeed, they can become so useful that whenever anything needs to be written down, your fingers will tend to drift toward the keyboard rather than pen and paper. A problem arises, however, when you want to take notes away from the home or office in a form that a computer can understand.

There is a growing market in portable computer systems like Tandy's Model 100 and the Epson HX-20. While these have the advantage of being able to act as portable word processors, or remote terminals for bigger systems, they are hardly as handy as a notepad or dictaphone. What about a word processing system that is small enough to carry in your pocket? A system so compact that it is battery powered and only needs one hand to use, yet can be connected to a printer or even another computer.

Such a device, called a Microwriter, has been available for nearly four years. Originally conceived by Cy Endfield, an expatriate American, it shuns the QWERTY keyboard in favour of a unique system of multiple key presses using only six push-button keys. The concept first arose out of a desire to create a hand-held game based on words, for which even a miniature keyboard would be both too big and too expensive. The obvious answer was to create a special kind of keyboard that used just a few keys with enough combinations to specify all the alphanumeric symbols. The breakthrough came with the invention of a symbolic code system that's unique to the Microwriter.

At first sight it seems impossible that the letters of the alphabet, not to mention numeric and punctuation symbols, can be created by combinations of just six keys, but these are indeed sufficient. And a few hours is all it takes to learn the common combinations. Indeed, the makers claim, with some justification, that it is a lot easier to learn than a QWERTY keyboard. The combination of keys required for each of the letters is based on the physical shape of the letter, a code that is often found easier to learn by non-typists. Because only one hand is needed, the Microwriter also opens the way to word processing for those disabled people who can't handle the multiple key presses often needed on a conventional keyboard to generate commands.



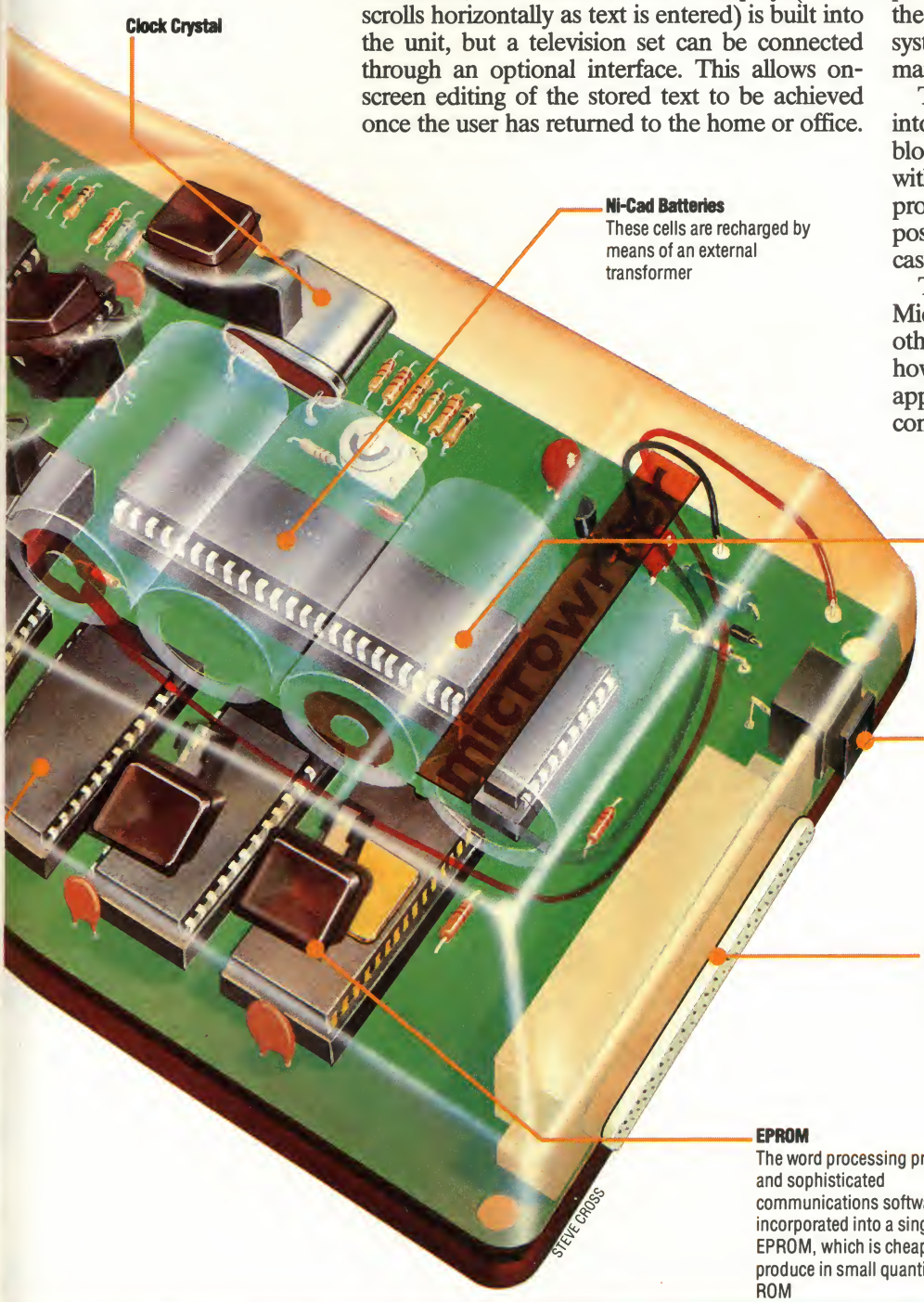


Internally, the Microwriter is designed to be as portable as possible. Both the internal microprocessor and its memory are CMOS (Complementary Metal Oxide Silicon) devices, which help reduce the power consumption. Enough power for 30 hours of use is provided by a rechargeable Ni-Cad pack. To display the characters, a 14-character LCD display (which scrolls horizontally as text is entered) is built into the unit, but a television set can be connected through an optional interface. This allows on-screen editing of the stored text to be achieved once the user has returned to the home or office.

As well as an RS232 serial interface for connection to a printer, the Microwriter is equipped with a cassette interface, which allows the text stored in its memory to be permanently saved or loaded back. The serial interface also enables the Microwriter to be used as a single-handed terminal to an ordinary computer or word processor. Documents keyed in while away from the home or office can be loaded into a full-sized system for more complete editing or manipulation.

Text within the Microwriter can be partitioned into separate documents, and this enables several blocks of unrelated text to be entered and dealt with separately. Simple editing facilities are provided: text can be added or deleted and it is possible to move large blocks around by using the cassette interface as a temporary buffer.

The designer's intention was that the six-key Microwriter keyboard would be incorporated into other electronic devices. For all its good qualities, however, the Microwriter has had only limited appeal, and it remains to be seen whether home computer manufacturers will take up the idea.



STEVE CROSS

On/Off Switch
Switching off and on will not lose data, and you can immediately resume writing the same document

Clock Crystal

Ni-Cad Batteries
These cells are recharged by means of an external transformer

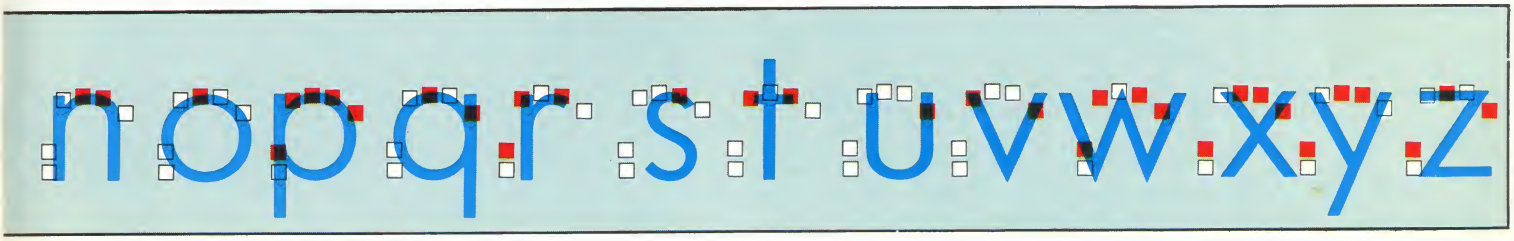
CPU
Both CPU and RAM are CMOS (Complementary Metal Oxide Silicon) devices to reduce power consumption

Power Socket
For recharging or mains operation with an external transformer

Expansion Interface
For future expansion, this port includes the microprocessor's address and data lines

EPROM
The word processing program and sophisticated communications software is incorporated into a single EPROM, which is cheaper to produce in small quantities than ROM

A Bunch Of Fives
Microwriter's documentation includes mnemonics and illustrations to help the user learn the different combinations of keys necessary to create the alphabet. The sixth key is used in combination with the others to provide further punctuation and editing commands



LIZ DIXON

Search Warrant

The time taken to locate a particular record can be greatly reduced using the 'binary search' — provided that the file has already been sorted into an appropriate order

The three most important activities in the address book program — adding new records, saving the file on tape or disk, and reading in the file from mass storage when the program is first run — have now been developed. But an address book is no use if you can only add information and cannot extract any. What is needed next is a routine to find a record.

Finding a complete record from a name is likely to be the most frequent activity, and that's why the first option on the choice menu (*CHOOSE*) is FIND RECORD (FROM NAME). Searching is a highly important activity in many computer programs, especially in database programs where specific items of data often need to be retrieved from a file. Broadly speaking, there are two search methods — linear and binary. A linear search looks at each element in an array, starting at the beginning, and carries on until the particular item is located. If the data items in the array are in an unsorted state, a linear search is the only type that can be guaranteed to work. The time to locate the item using a linear search in an array of N items has an average value proportional to N/2. If there are few items to be searched through, N/2 may be perfectly acceptable, but as the number of items increases, the time taken to perform the search may become excessive.

If the data in the file is known to be in a sorted state, however, there's a far more efficient searching method, known as the 'binary search', which works in the following way. Suppose you want to find the definition of the word 'leptodactylous' in a dictionary. You don't start at the first page and see if it's there, and go on to the second page if it's not, working your way through the dictionary until you find it. Instead, you put your thumb roughly in the middle of the book, open the page and see what's there. If the page you open happens to start with 'metatarsal', you know you've gone too far, so the second half of the book is irrelevant and the word you want will be somewhere in the first half of the book. You then repeat the process, treating the page you originally opened as though it were the end of the dictionary. Again you split the first part of the dictionary in two and open the page to find 'dolabriform'. This time you know that the page selected is too 'low' and (for the purposes of our search for 'leptodactylous') can be considered as though it were the first page — all earlier pages are irrelevant as they are known to be too 'low' in the sense that 'l' is 'higher' than 'd'. The 'first' and 'last' pages of the dictionary can now be considered as the ones starting with 'dolabriform' and 'metatarsal' respectively. Again you put your thumb in the middle of the 'relevant' section and open up at 'ketogenesis'. Again this is too 'low' so the word we are looking for must lie between this page and the 'metatarsal' page. Repeating this process often enough is guaranteed to locate the word we are looking for — as long as it is in the dictionary!

In the example we have just considered, 'leptodactylous' was the 'search key'. The search key is the entry we are trying to find. Each time we examine a record, we will compare the search key against the 'record key' to locate the 'target' or 'victim'. Together with the record key we can expect to find what is called 'additional information', logically enough. The additional information for the record key 'leptodactylous' would be the dictionary definition of the word — in this case, slender-toed.

The analogy with searching through a file in a database for a target record is a close one, provided that the records have been previously sorted as the entries in a dictionary have. Think how difficult a dictionary would be to use if the entries were in the order the lexicographer first thought of them!

The search routine required for our address book will need to be more complicated than we might first appreciate for reasons that will become apparent. The first thing the search routine — let's call it *SCHREC* for the time being — will do is request the name to be searched for. This is called the search key. Suppose that somewhere in the file there is a record for a person called Peter Jones. The record for this person will have a field (with the name in standardised form) containing JONES PETER. The search routine might prompt us with a message such as WHO ARE YOU LOOKING FOR?, and we would respond with PETER JONES, or perhaps P. JONES or Pete Jones. Before this gets too complicated, let's assume that we respond with the full name, Peter Jones. The first thing the search routine will do will be to convert this response to the standardised form, JONES PETER. Next, it will compare our input, the search key, with the various contents of the MODNAM$ fields. If the program were using a linear search, the search key would be compared with each MODNAM$ field in sequence until a match was found or until it was discovered that an exact match did not exist.

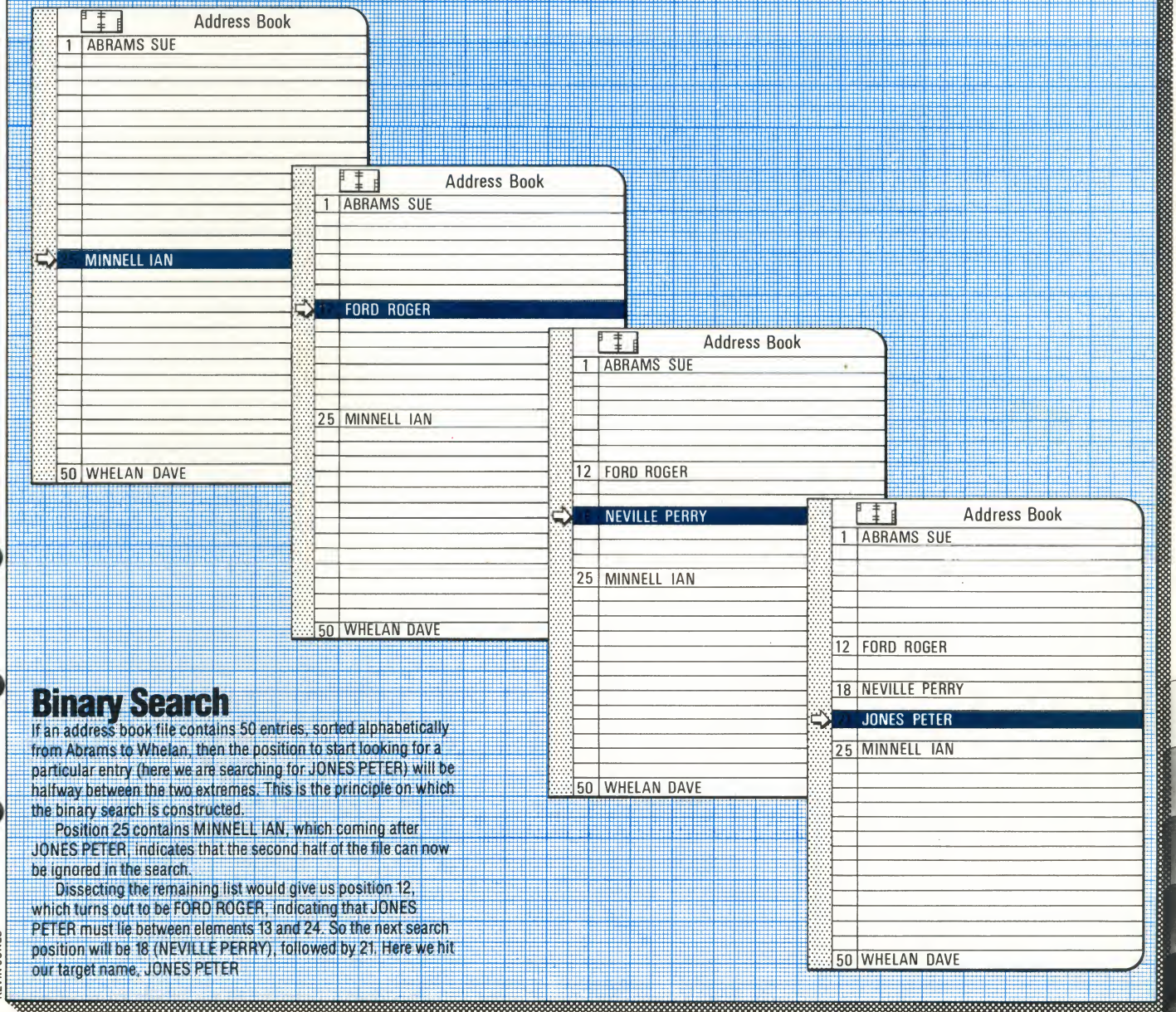
As we have already noted, however, a linear search is not efficient compared with a binary search if the data is already sorted. The search routine can ensure that the records are sorted by starting with an IF RMOD = 1 THEN GOSUB *SRTREC*. The program knows that the lowest element in the array to be searched will be MODFLD$(1) and the highest will be MODFLD$(SIZE – 1). To conduct the search, we will need three variables: BTM for the bottom of the array (MODFLD$(1) at the beginning); TOP for the top of the array (MODFLD$(SIZE – 1) at the beginning); and MID for the value corresponding to the middle element.

Using the dictionary analogy, we can assume that BTM = ARRAY(1) and TOP = ARRAY(SIZE – 1). In other words, the array we have to consider for the search starts with the 'smallest' element and ends with the 'largest' element. We can therefore LET BTM = 1 and LET TOP = SIZE – 1 (remember that SIZE is always one larger than the number of records currently in the address book).

Suppose that there are 21 valid entries in the address book. SIZE will have a value of 22. BTM will have a value of 1. TOP will have a value of 21. The value of MID, the position of the middle element, can be derived in BASIC from INT((BTM + TOP)/2). If the BTM value is 1, and the TOP value is 21, the MID value will be 11.

To conduct a binary search, we first assume that the whole file is valid and find the mid point INT((BTM + TOP)/2) inside a loop that is terminated either if the target is found or if there is no match. Then we check to see if the search key (SCHKEY$) happens to be equal to the MID value of the array. If the MID value of the array is too small, we know that ARRAY(MID) is the lowest part of the array we

Address Book Search



Binary Search

If an address book file contains 50 entries, sorted alphabetically from Abrams to Whelan, then the position to start looking for a particular entry (here we are searching for JONES PETER) will be halfway between the two extremes. This is the principle on which the binary search is constructed.

Position 25 contains MINNELL IAN, which coming after JONES PETER, indicates that the second half of the file can now be ignored in the search.

Dissecting the remaining list would give us position 12, which turns out to be FORD ROGER, indicating that JONES PETER must lie between elements 13 and 24. So the next search position will be 18 (NEVILLE PERRY), followed by 21. Here we hit our target name, JONES PETER

KEVIN JONES

need to consider, so BTM could be set to MID. Slightly more efficient, however, is to set BTM to MID + 1, since we already know that ARRAY(MID) is not equal to the search key. Similarly, IF ARRAY(MID) > SCHKEY$, TOP may be set to MID – 1.

As an interim step towards developing a fully working routine, the program shown can take a dummy input (which needs to be in exactly the same format as the MODFLD$ fields) and will either print RECORD NOT FOUND if there is no match, or RECORD IS NO (MID) if there is a match. As the routine starts with line number 13000, it can be added on to the end of the program as presented on page 399, and will work as long as line 4040 is changed to IF CHOI = 1 THEN GOSUB 13000.

Line 13240 contains the STOP statement. This will stop the program temporarily as soon as the RECORD NOT FOUND or RECORD IS NO (MID) messages are displayed. The program can be re-started at the same line number, without losing data, by typing CONT. Without STOP, the program would rush on to the RETURN statement in line 13250 and the message would appear too briefly to be legible.

Let's consider this program fragment in more detail. Line 13100 sets BTM to 1, the position of the lowest element in the MODFLD$ array. TOP is set to SIZE–1 in line 13110. This is the position in the MODFLD$ arrays where the highest element is located. Line 13120 initialises a loop that will only be terminated when either a match is found or no match is known to exist.

Line 13130 finds the mid point of the array by halving the sum of the bottom and top index of the array (INT is used to round off the division, so that MID cannot assume a value such as 1.5). There's a chance that the contents of MODFLD$(MID) will be the same as the search key (SCHKEY$), but if they are not the same, as is likely, L will be set to 0, ensuring that the loop will be repeated. If the test in line 13140 fails, MODFLD$(MID) will either be lower or higher in value than SCHKEY$. The value of BTM will then be set to one more than the old value of MID (line 13150), or the value of TOP will be set to one less than the old value of MID. The reason the value of MID itself is not used is that the failure of the test in line 13140 has already demonstrated that MODFLD$(MID) is not the target we are searching for and there is no point in looking at that element of the array next time round the loop.

If no match is found, the value of BTM will eventually exceed the value of TOP. The loop can be terminated (line 13170) and a RECORD NOT FOUND message printed (line 13200).

This program fragment is presented for demonstration purposes and to enable the search routine to be tested. As it stands, its use is rather limited. Without the STOP in line 13240 we wouldn't even have time to see the message flashed on the screen. What is required is a display of the full record, as it was originally typed in. Once the record number is known, it is a simple matter to retrieve any of the additional information required — NAMFLD$, STRFLD$ etc. Below the display of the record, we would probably want a message such as PRESS SPACE BAR TO CONTINUE (back to the main menu) and perhaps further options such as PRESS "P" TO PRINT.

Not so easy, unfortunately, is deciding how to handle the input of *FNDREC*. In the program fragment, the input expected (in line 13020) must be in the standardised form — JONES PETER, for example. This is clearly not good enough. People don't think of names in inverse order, and it's an unreasonable burden on the user to have to enter the name in upper case letters. Additionally, the slightest deviation between the name input originally would result in a RECORD NOT FOUND. The first two problems could, one would expect, be handled by *MODNAM*. The third problem of how to cope with an approximate match is far more interesting, but very much harder to solve.

Before considering this problem, let us see why *MODNAM* will not solve the first two problems. If you go back and look at *MODNAM*, which starts at line 10200, you will discover a good illustration of one of the commonest traps into which programmers fall — lack of generality. This subroutine ought to be able to handle conversions from 'normal' names to 'standardised' names whenever this operation was needed. Even though it was written as a separate subroutine, it was clearly written with *ADDREC* in mind. It assumes that the name to be converted will always reside in NAMFLD$(SIZE) and that after conversion the modified name will always be stored in MODFLD$(SIZE). Faced with this situation, the programmer has three choices: either completely rewrite *MODNAM* to make it general, which would in turn involve further changes in other parts of the program. Or write an almost identical routine just to handle the input for *FNDREC*, which represents wasted effort and takes up more space in memory. Or resort to some bad programming technique to allow the unmodified *MODNAM* routine to be used. This last alternative is in some ways the least attractive. It will solve the problem, but the actual working of the part of the program that has been modified is likely to be unclear, even to the writer of the program, and a nightmare to anyone else trying to use the program.

The moral of the story is: make subroutines as general as possible, so that they can be called by any part of the program.

To illustrate bad programming technique, or 'dirty' programming as it is often called, and to show how unclear it can make the program, consider line 13020 of the program fragment, INPUT "INPUT KEY ";SCHKEY$ and then look at the modification or 'fix' that would allow *MODNAM* to be used:

```
13020 INPUT "INPUT KEY";NAMFLD$(SIZE)
13030 GOSUB 10200: REM *MODNAM*
   SUBROUTINE
13040 LET SCHKEY$ = MODFLD$(SIZE)
13050...
```

Luckily, SIZE is always one bigger in value than the highest valid record. In other words, there is no record at position SIZE in the arrays, so this fix will not modify any existing record. But without some extensive REMs explaining what's going on, think how confusing these three lines would be to someone who had not been involved in the development of the program!

Back to the more interesting problem of dealing with 'near misses'. Suppose we had entered someone's name as Pete Jones during an *ADDREC* operation, but as Peter Jones during *FNDREC*. These would be converted to the standardised forms JONES PETE and JONES PETER respectively, and no match would be found during the search, even though the record we wanted was there. We will not attempt to solve this problem, because a satisfactory solution would represent a major programming task. For readers interested in experimenting, however, here are some pointers:

```
BEGIN {search array for exact match}
   IF exact match found
      THEN PRINT full record
      ELSE search array for close-match
         IF close-match found
            THEN PRINT record for close-match
            ELSE PRINT "NO RECORD FOUND"
         ENDIF
   ENDIF
END
```

The procedure for close-match could be something along the lines of:

```
BEGIN {close-match}
   Search array for exact surname match
   IF exact surname match
      THEN search forenames for max-match
      PRINT record for max-match
      ELSE search surnames for max-match
         IF surname max-match found
            THEN PRINT record for max-match
         ENDIF
   ENDIF
END
```

The procedure for max-match could be roughly defined as finding the target string with the maximum number of characters in common with those in the key string. Or it could accept a situation in which the key string was wholly contained within the target string, or vice versa. There are no simple solutions, but plenty of scope for enterprising programming.

There is one 'side effect' of the program fragment presented. Suppose the following sequence of events takes place. There are ten records in the data file. You run the program and then use *ADDREC* to add a new record, followed by *FNDREC* to locate a record. When *EXPROG* is finally run, to save the file and terminate the program, the record you added will not be saved (although all the other records will be). This is a direct result of something that happened in the execution of *FNDREC*. Can you see why the record added will not be saved?

In the next instalment of the course we will explain how to prevent this loss of data; show what the CURR variable is used for, and describe how to delete or modify a record. Other options on the main menu (*FNDTWN* etc.) are closely similar to routines we have already worked out. Readers will be left to implement them for themselves if they are required.

Finally, consider what would happen if there were exactly 50 records in the data file and the modified *FNDREC* routine (that calls *MODNAM*) were used. (Hint: SIZE will have the value 51.)

Basic Flavours

ZX81 SPECTRUM

For Sinclair machines, the following modifications are required:

```
13000 REM *FNDREC* TEST VERSION
13010 IF RMOD = 1 THEN GOSUB 11200
13020 PRINT "INPUT KEY"
13030 INPUT S$
13100 LET BTM = 1
13110 LET TOP = SIZE - 1
13120 FOR L = 1 TO 1
13130 LET MID = INT((BTM+TOP)/2)
13140 IF M$(MID) <> S$ THEN LET L=0
13150 IF M$(MID) < S$ THEN LET BTM =
      MID + 1
13160 IF M$(MID) > S$ THEN LET TOP = MID
      - 1
13170 IF BTM > TOP THEN LET L = 1
13180 NEXT L
:
:
13200 IF BTM > TOP THEN PRINT "RECORD
      NOT FOUND"
13210 IF BTM <= TOP THEN PRINT "RECORD
      IS NO ";MID
:
:
13240 STOP
13250 RETURN
```

Notice once again the problem of single-letter string variable names: here S$ and M$ have been substituted for SCHKEY$, MODFLD$

Erratum
In the ZX81 and Spectrum Basic Flavours on page 257, lines 9990 to 9992 should not have been included in Step 3

```
13000 REM VERSION OF *FNDREC* FOR TESTING
13010 IF RMOD = 1 THEN GOSUB 11200
13020 INPUT "INPUT KEY ";SCHKEY$
13030 REM
13040 REM
13050 REM
13060 REM
13070 REM
13080 REM
13090 REM
13100 LET BTM = 1
13110 LET TOP = SIZE - 1
13120 FOR L = 1 TO 1
13130 LET MID = INT((BTM + TOP)/2)
13140 IF MODFLD$(MID) <> SCHKEY$ THEN L = 0
13150 IF MODFLD$(MID) < SCHKEY$ THEN BTM = MID + 1
13160 IF MODFLD$(MID) > SCHKEY$ THEN TOP = MID - 1
13170 IF BTM > TOP THEN L = 1
13180 NEXT L
13190 REM
13200 IF BTM > TOP THEN PRINT "RECORD NOT FOUND"
13210 IF BTM <= TOP THEN PRINT "RECORD IS NO ";MID
13220 REM
13230 REM
13240 STOP
13250 RETURN
```

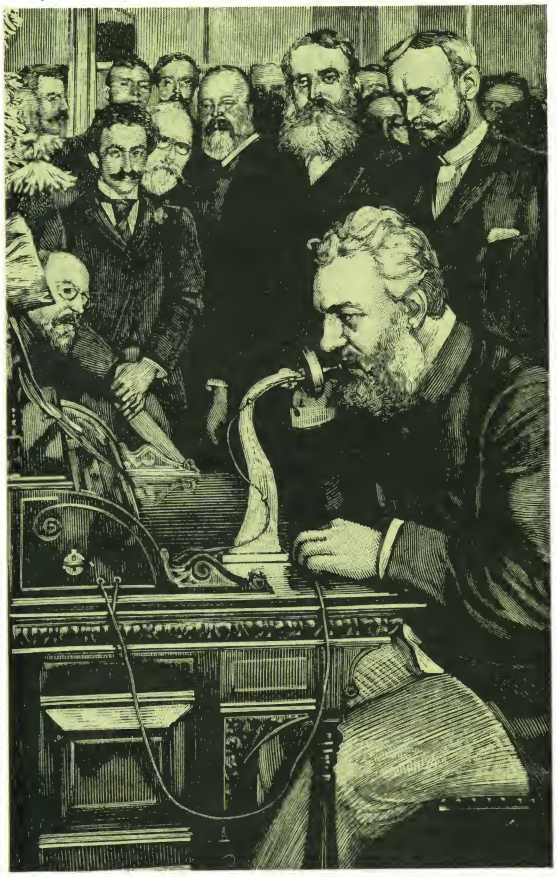
TONY LODGE

Ma Bell

Bell Laboratories has been responsible for numerous developments in the history of the computer — both in hardware and software

A hundred years ago, Queen Victoria was greatly amused by a new invention that allowed her to speak with her ministers in London from the Isle of Wight. The telephone has been greatly improved since those days of the hand-cranked set through research and development, and one of the spin-offs from this work has been the computer. In the early stages of the telephone's development, the American Telephone and Telegraph Company decided to set up an organisation that would research ways of improving the telephone system. Thus, in 1925, Bell Laboratories (known as 'Ma Bell') was born at Murray Hill, New Jersey.

Bell Labs is an unusual institution since it is solely devoted to doing research, and yet is owned by a corporation whose only purpose is to make profit. The scientists are deliberately kept away from the day to day engineering problems encountered in running such a business because Bell consider research to be a long term speculative investment. Gifted scientists are allowed to pursue those aspects of research that they think are important because, the corporation believes, a few of their ideas will be worth the investment. Over the years, Bell Labs has collected two Nobel prizes and made discoveries in quite diverse areas of scientific research. Here, we consider some aspects of their research that were particularly relevant to the development of the computer.



The Bells Are Ringing
Bell Laboratories takes its name from Alexander Graham Bell (1847–1922), who is generally credited with the invention of the telephone in 1876. It is generally believed that the first words ever transmitted over wires by electrical means were from Bell to his assistant, situated in the next room; they were 'Come here, Mr Watson, I want you!'

By the 1930's, telephone systems were becoming increasingly automatic and sophisticated. Messages were sent in analogue form over the telephone cables and the calls were connected using information contained in a digital dialling code. The number dialled was first converted at the exchange from an analogue signal into a sequence of digital pulses. This was temporarily stored in a memory made out of relay switches until the connection was completed by a bank of crossbar switches. These counted the pulses in the dialling code and converted them into co-ordinates on an electromechanical switchboard. All the ingredients of a computer were included — they were just waiting for the right person to come along.

George Stibitz was a mathematician employed by Bell who noticed the similarity between 'counting' pulses and adding them together. Working at home on his kitchen table with some old crossbar switches and electromechanical relays, he made the first relay computer circuits.

Stibitz then began working with an experienced switching engineer, Samuel B Williams, who had been building switching circuits for 25 years, and the two men created a Complex Number Calculator (complex numbers involve the so-called 'imaginary' numbers — the square roots of negative numbers — and are needed to obtain complete solutions to polynomial equations). Work was begun in 1937, and the device consumed 450 relays and 10 crossbar switches. It operated in binary notation and was able to divide two eight-digit numbers in 30 seconds. The Complex Number Calculator became operational on 8 January 1940, and in September of the same year it was demonstrated to the American Mathematical Society at Dartmouth College (where BASIC was later formulated). The calculator had the facility of remote and multiple access through typewriter keyboards connected by telephone wires to the calculating mechanism in New York. People were particularly impressed by its 'human' form of operation: after the calculator was asked a question it would seem to pause for some seconds before giving the answer!

Many minor hardware devices also originated at Bell, such as the floating air-cushions used in magnetic tape heads, and negative feedback amplifiers. But the most famous invention was the transistor, created in 1947 by Bardeen, Brattain and Shockley (see page 47). It was the transistor that made possible the second generation of computers.

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.

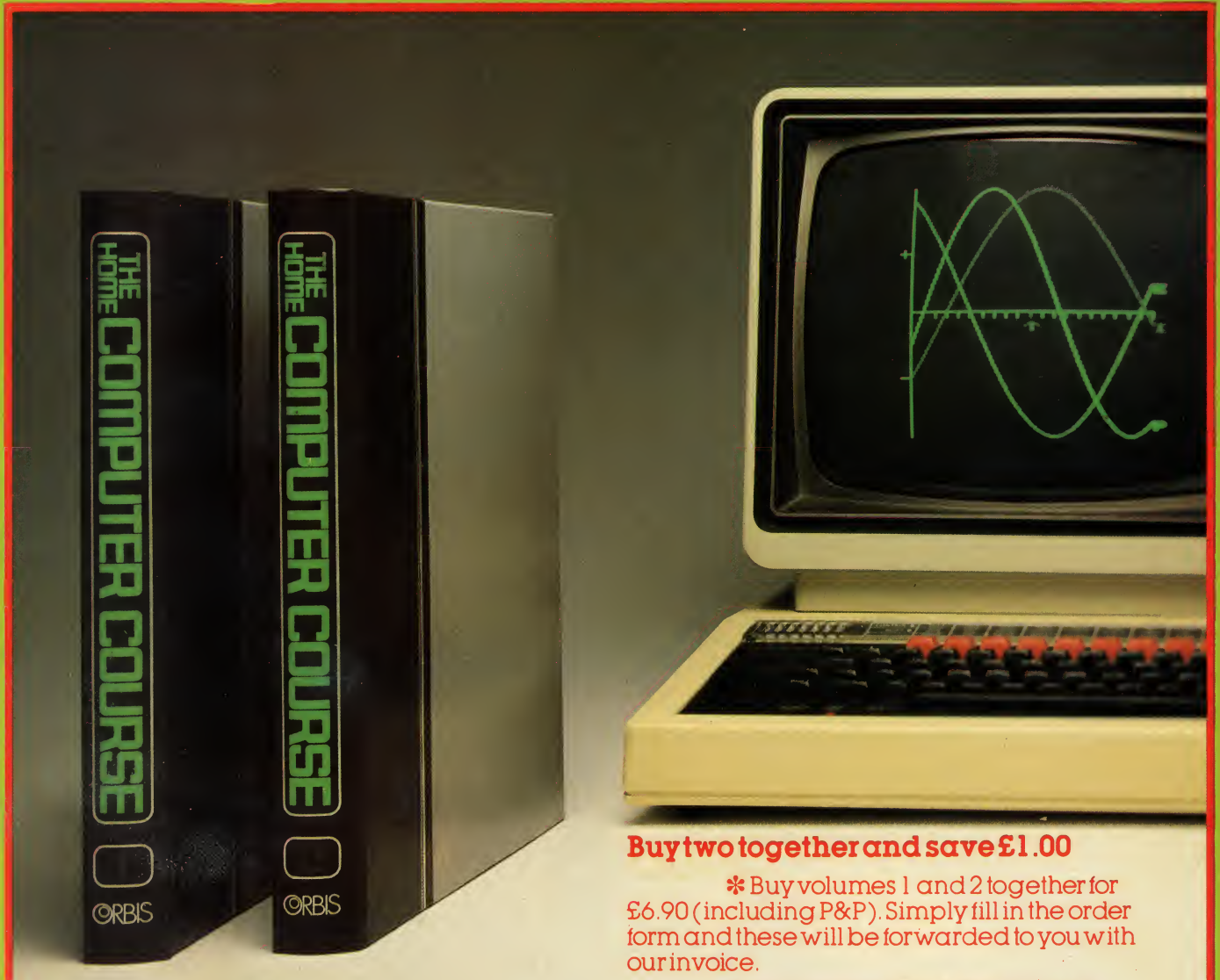
And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

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