ISSN 0265-2919

R 80p

THE HOME COMPUTER COURSE

15

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

An ORBIS Publication

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

CONTENTS

Hardware Focus

Software

Basic Programming

Insights

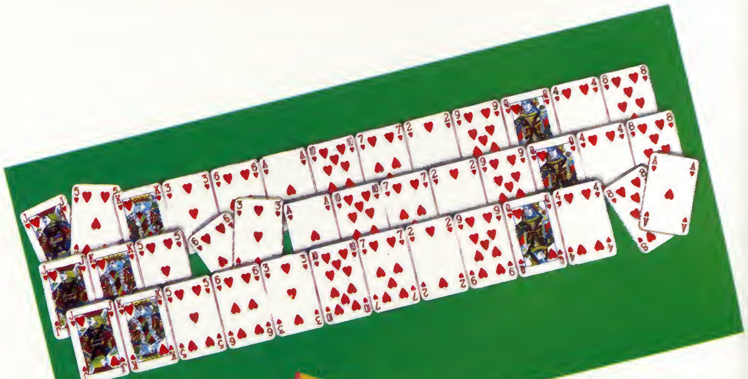
Passwords To Computing

Pioneers In Computing

Sound And Light

Next Week

- We examine the Sharp MZ-700, a low cost home-and-business computer with built-in printer and cassette player
- Inexpensive robot arms are now becoming available for a wide range of home computers. We look at the Colne Robotics Armdroid
- A modem can open up a whole new world for the home computer user. We suggest some applications



PAGE 286

PAGE 296

Editor Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistants** Liz Dixon, Safu Maria Gilbert; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Lisa Kelly, Steven Colwill, Richard King, Bob Chappell; **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.

Your Obedient Servant

Industrial robots can now visually recognise objects and learn new tasks by imitating human actions

The term 'robot' is derived from the Czech word for work, *robota.* It was coined by playwright Karel Čapek in his 1920 play *R.U.R.* (Rossum's Universal Robots) and was subsequently enthusiastically adopted by science fiction writers. Despite the many fictional accounts of the powers of robots, they are nothing more than an electromechanical extension of the computer, with all a computer's limitations and failings.

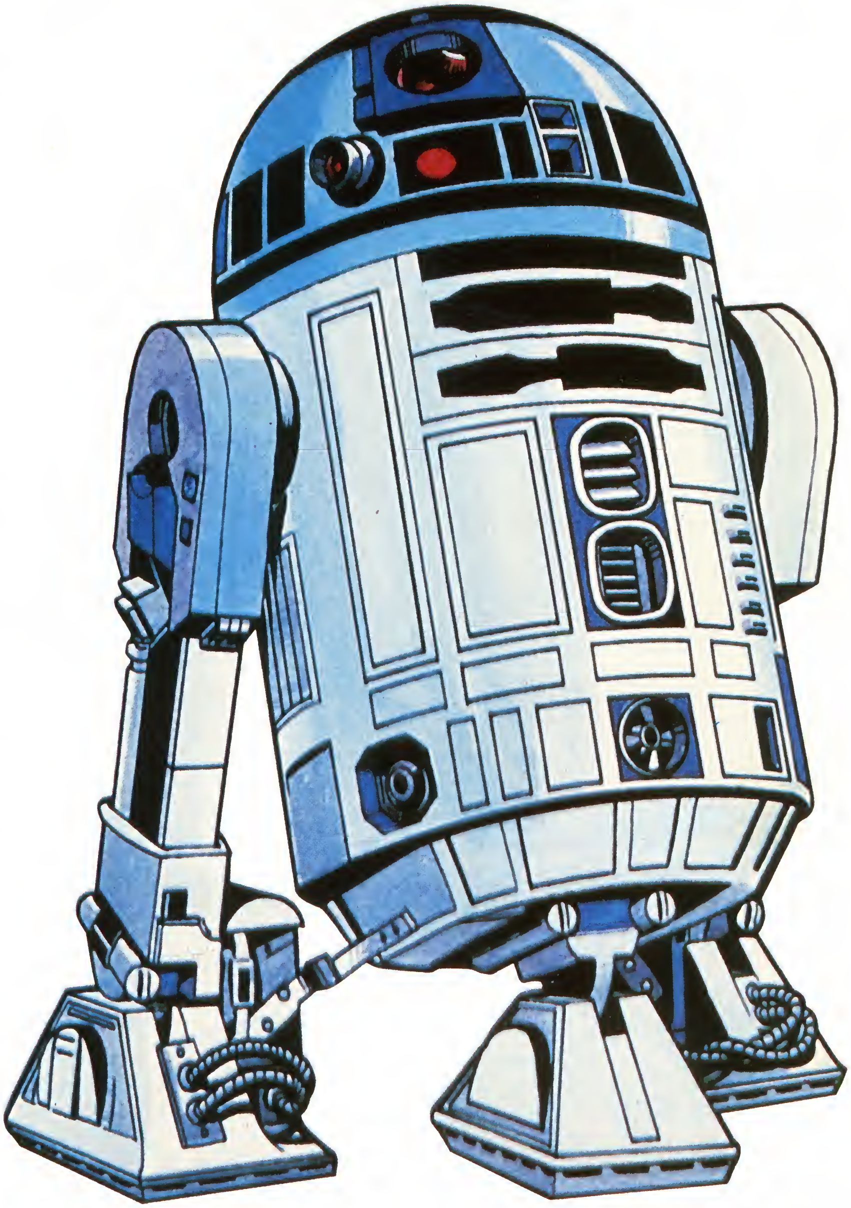
Their origins are to be found in the machine shops of the fifties, where the theory of numerical control of machine tools was first applied. These first efforts were predictably crude: machines that were controlled by five-hole paper tape (the sort used by telex machines), which at best could only move one fixed tool from point to point around the object they were working on.

The next step in their development was the introduction of the ability to change tools in mid-job. This was accomplished by the use of a 'carousel' or rotating rack of tools, all with identical fixings, which could be selected and fitted to the tool holder under program control.

Even with this refinement, a particular machine could perform only one type of task: a lathe was still a lathe, even though it could perhaps do all the turning jobs required in a particular process. At around the same time, remotely-controlled hands and arms were being developed to work in dangerous environments — beneath the ocean, for example, or in laboratories handling radioactive materials. These manipulative devices were merely extensions of the operator's own hands, but computers were soon used to control them directly. The robots that have since been developed are more accurately referred to as 'robot arms', as they consist of one tool holder mounted on an extending or articulated arm.

If we wish to understand how robots are programmed, we must first consider them in relation to the space in which they operate. Most industrial robots are fixed in position, so the space will be a sphere that is flattened at the bottom, and we can think about the question of control of the robot as a simple exercise in three-dimensional geometry. The centre of the spheroid will be the robot's 'shoulder' joint, and the radius will be the length of the extended arm, measured from the 'shoulder' to the tip of the 'fingers' — the gripper or tool holder. Any point within this space can be expressed as three co-ordinates: for example, as distances north/south, east/west and up/down, from a 'datum point' or zero position. In this case the co-ordinates are known as Cartesian, after the 17th-century French mathematician René Descartes. Alternatively, the position can be expressed in spherical co-ordinates. In everyday language this could be, say: 'at a distance of two metres in a direction north-east and thirty degrees above the horizontal'. The datum point in this case is the robot's 'shoulder'.

However, the problem of programming the robot involves giving it a set of instructions about moving from place to place, and so there is yet a third method of positioning the tool holder.



Celluloid Hero
R2D2, the endearing robot from 'Star Wars' was in fact controlled by a human operator. Its design, however, reflected what many people think robots ought to look like

Battery-Powered Robot

The Hero-1 is a completely self-contained battery-powered robot that combines some of the functions of a turtle with the manipulative ability of a robot arm. Costing some £2,500 — or £1,600 in kit form — it might appear, at first glance, to be an expensive toy. But, it is in fact, a remarkably flexible computer system in its own right, with such advanced features as speech synthesis, light level sensors, auditory input and, (because it's mobile), an ultrasonic range finder that also doubles as a movement detector

COURTESY OF COMPUTING PLUS

IAN McKINNEL

Known as point-to-point positioning, this requires the datum point to move with the tool holder.

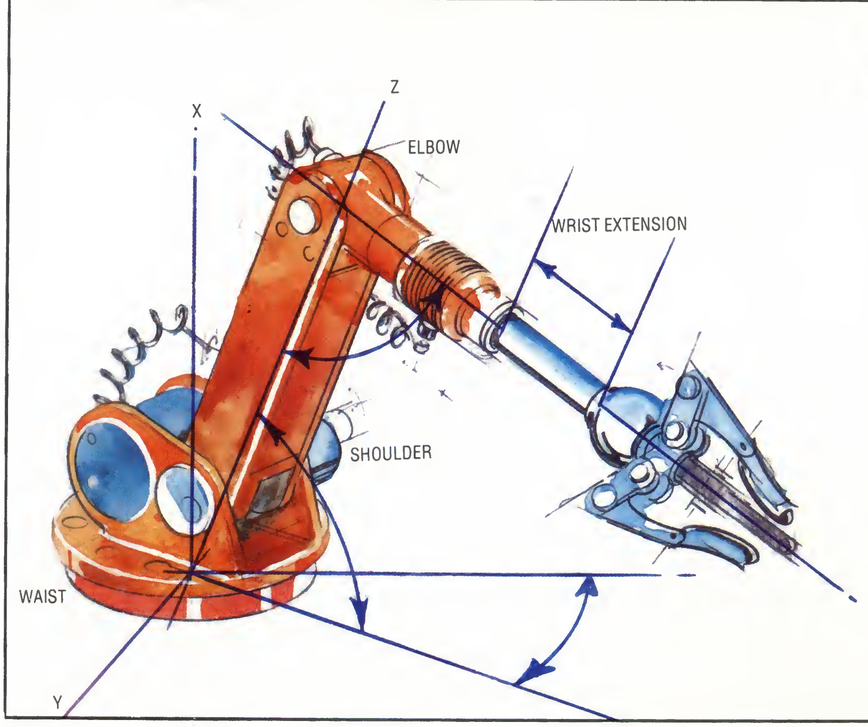
Typically, industrial robots are accurate to within one millimetre. Even the simpler models — available for a few hundred pounds and capable of being used with any home computer that has eight-bit parallel output — are accurate to within two millimetres. That observation in itself is interesting given that the cost differential is at least 50-1.

There are two generally accepted methods of driving robot arms. For those with a low payload, stepper motors (electric motors that move by a predetermined amount each time current is applied to them, as used in disk drives to position the read/write head) are sufficient. But for robot arms used on a production line, where heavier weights need to be manoeuvred, it is more common to employ hydraulic rams to move the various parts of the arm around their fulcra (the points around which they pivot). It is quite a simple matter to measure the volume of hydraulic fluid being passed into the rams, and to deduce from that the movement at the other end, to well within the operational requirements of accuracy.

Industrial robots invariably contain a purpose-built minicomputer (or a large capacity microcomputer in later models) that does nothing but control the arm, and run a programming language designed for that purpose. As there is no requirement to do more than indicate co-ordinates, and issue simple commands like CLOSE GRIPPER or OPEN GRIPPER, the programming language contains no instructions for handling text. Program instructions are entered through an enlarged numeric keypad attached to the computer by means of a long 'umbilical cord', so that the operator may move around the robot arm while entering the instructions. The more advanced versions of these 'pendant panels,' as they are called, include a precision joystick.

Another programming method, known as 'Follow Me', is especially useful in tasks that do not require particularly accurate tool placement, such as paint spraying. Here the robot arm includes a provision for the operator to grasp the tool holder, directly move it around the job, and have those movements entered directly into the computer's memory. The robot will then repeat those movements every time the program is executed.

In all these methods, the position being defined is that of the tool holder itself. The operator is not concerned with the relative positions of the individual sections of the robot. The



Angular Movement

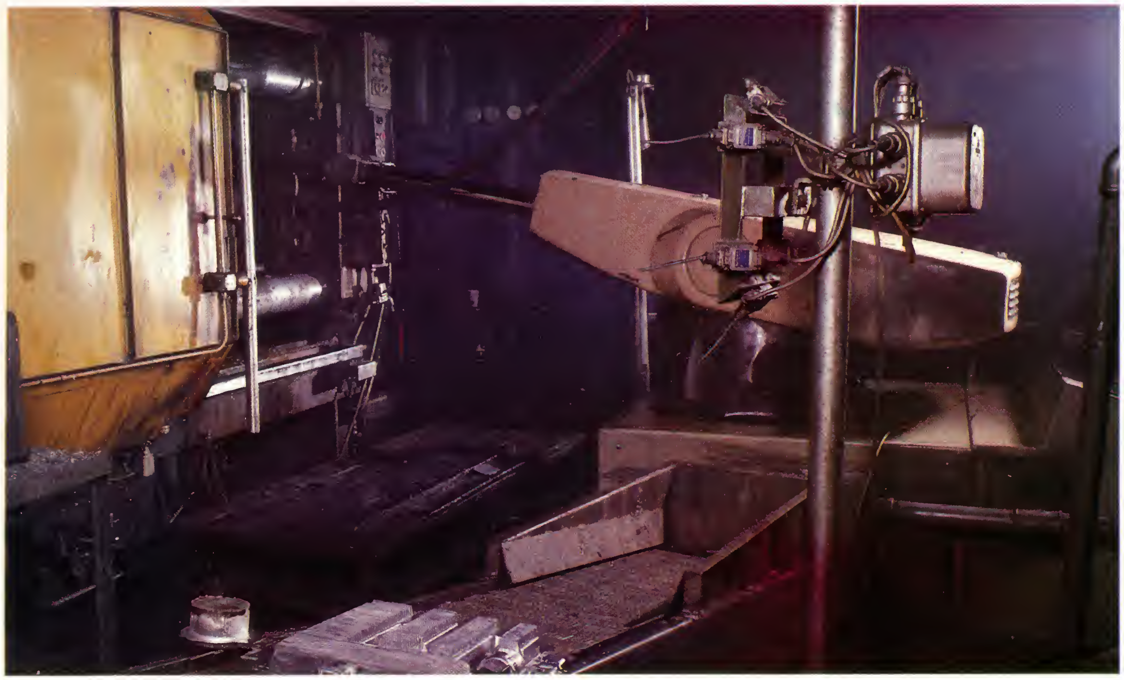
One of the most difficult aspects of programming a robot arm is converting the geometry. We are used to specifying positions using Cartesian or x,y,z co-ordinates. What the robot needs are angles for the 'elbow' joint, the 'shoulder' joint, the 'waist' rotation, and the distance that the wrist must extend. In simpler systems the programmers must give the values for all four. More sophisticated robots can perform all the conversions from Cartesian co-ordinates

BOB FREEMAN

programming language resident in the robot's control computer works out what they should be. It also performs any necessary optimisation, ensuring that the tool moves from one place to another by the shortest possible route. The orientation of the tool holder is controlled automatically, maintaining both horizontal and vertical relative positions unless instructed otherwise. The speed of point-to-point movement is also automatic: the tool holder is disengaged slowly, moves rapidly to within a short distance of the destination point, and then slows down to re-engage the workpiece at the new site.

The robots we have discussed so far are capable only of 'blind obedience', repeating the same task at exactly the same location, irrespective of external influences. Their main use is in the engineering industry, especially in the production of motor vehicles. This has long been organised into production lines, in which the component or partially completed vehicle is always precisely located in space and time. This is vitally important to the successful operation of a robot production process, for if the component is wrongly positioned, the robot will not adapt its activity accordingly. In an attempt to overcome this, a variety of sensors can be fitted to the tool holder. The simplest of these can be an ordinary on/off microswitch. Contingency plans can be built into the control program (a WAIT command, for instance), to be executed if the switch is not brought into contact with the workpiece, but more sophisticated plans will require human intervention.

An alternative to pressure sensing might involve the use of a light sensor. If a light source were positioned so as to be obscured from the sensor on the tool holder by the workpiece, the tool holder could be stopped before it reached collision point, put into WAIT mode until the workpiece was in position, and then allowed to continue. Of course, this isn't foolproof either, and for situations where complete reliability is required, it is possible to install an image recognition system based on charge-coupled device (CCD) television cameras. These cameras focus the image directly onto an array processing microchip (a chip split up into a hundred or more individual photosensors, each capable of registering not just black or white but also a range of intermediate tones). Each individual sensor requires perhaps one byte of memory to define the contrast in the grey scale. Initially each object is 'photographed' a number of times, and a learning program averages out the values. At run time, the CCD camera makes an image of the object, which is then compared with the reference image in memory. If the two match, then the operation can go ahead. By this method it is possible to check that the correct workpiece is present, and that its position and attitude are correct.



Factory Act
Robot arms, like the one seen here at work in a die casting shop, are taking over more and more of the dirty, dangerous and repetitive jobs to be found in industry. The cleaning of castings preparatory to their being machined is a good example. The casting, fresh from the mould, is much too hot for human hands, and would normally be put to one side to cool. The robot, however, is not susceptible to heat so can deal with it immediately and despatch it on to the next operation

A further use of this image processing system is in the selection of components from a 'mixed bag'. This 'picking and placing', as it is known, is an increasingly common application for small robots as an adjunct to a regular production line. In addition to the production process itself, industrial robots are commonly used in the testing and quality control stages, often in pairs to allow a greater degree of flexibility in the positioning of the product.

We started by considering the robot in fiction — and with good reason. There are few better examples of truth following fantasy than in the development of the industrial robot, and there is no reason why robots should not eventually become the self-contained and 'self-motivated' entities of science fiction. This will not happen, however, until Artificial Intelligence is more than just a concept.

Sounding Out Vic

A close look at sound generation on the Vic-20...

The Vic-20 was one of the first home computers to appear in the UK. As a consequence, its facilities may appear to be a little lacking in comparison with more recent computers. Additionally, Commodore don't make it particularly easy to construct sound or music programs as Vic-20 BASIC, in common with Commodore 64 BASIC, has no commands that relate specifically to sound. All sound control is achieved by a series of POKEs into memory locations. This principle also applies to the Commodore 64 and the techniques outlined here for the Vic-20 would be useful to the Commodore 64 user. The degree of sound control available is limited to volume (equivalent to envelope with A = D = R = 0), frequency on three oscillators and a noise generator. Output is available via the television speaker alone. In addition, due to inaccuracies in the way the Vic-20 selects frequencies it is impossible to obtain the correct pitch for all notes on the musical scale.

With only these capabilities the Vic-20 has little value for music making; although with thought, patience and a little knowledge of BASIC programming these limited features can be used to create 'tunes' of two and three note chords.

Sound Control

The Vic-20 is supplied with three square wave oscillators and a noise generator. Each oscillator covers approximately three octaves of sound, offset in frequency as follows:

Osc.1	Osc.2	Osc.3	Freq. Range (Hz)	Octave
●			(65.41-123.47)	1
●	●		(130.81-246.94)	2
●	●	●	(261.63–493.88)	3
	●	●	(523.25-987.77)	4
		●	(1046.5-1975.53)	5

This arrangement allows the user to cover five octaves in total with at least one oscillator available in each octave. Octave 3, which starts at middle C and contains the standard reference A at 440Hz, is available to all three oscillators.

Control of the oscillators is exercised by changing the contents of five memory locations as follows:

Memory Location	Oscillator
POKE 36874,X	1
POKE 36875,X	2
POKE 36876,X	3
POKE 36877,X	noise

In each case X is a whole number between 135 and 241 (0 switches that oscillator off), which refers to a table of equivalent note values on page 73 of the booklet supplied with each Vic-20. Before the selected frequency can be heard the volume level must be set, as follows:

```
POKE 36878,V
```

where V can be set between 0(off) and 15(loud) affecting all oscillators and noise. For example:

```
POKE 36874,219:POKE 36875,219:POKE
  36876,219:POKE 36878,7
```

This plays reference A at 440Hz on oscillator 1, A an octave higher on oscillator 2 and A an octave higher still on oscillator 3, all at a mid-range volume of 7. Don't forget to POKE each location to 0 to turn them off!

Notes And Pauses

Without a duration for each note and the correct pauses between them, a sequence of notes blurs one into another. To facilitate these 'wait' periods, one of two methods can be used to make the computer 'mark time' between POKEs. The first method is FOR...NEXT loops where the pause is timed by a long empty loop such as:

```
10 POKE 36878,7
20 POKE 36876,203
30 FOR P=1 TO 200
40 NEXT P
50 POKE 36878,0
60 POKE 36876,0
```

This sequence of commands plays the note D# for 200 FOR...NEXT steps. However, this method depends on careful external timing of the loop for accuracy. An easier and more elegant way to set durations and pauses is by using the Vic-20's built-in clock, which counts in 60ths of a second (jiffys) and can be referenced within a program using the variable TI. This is extremely useful, as a command can be constructed to 'wait' for an accurately measured period of time, as follows:

```
10 POKE 36878,7
20 POKE 36876,203:D=TI
30 IF TI-D<15 THEN 30
40 POKE 36878,0
50 POKE 36876,0
```

These commands play the same note as before but for a period of 15 jiffys (a quarter of a second). D is set at the value of TI when the sound is switched on. Line 30 counts off 15 jiffys before proceeding to line 40. Tunes can be constructed by using the same principle to pause before playing a different note, and so on. Next time we look at the Vic-20 in the Sound And Light series, we'll investigate how to play tunes.

Lighting Up Dragon

...and graphics capabilities of the Dragon 32

The Dragon 32 computer features a particular dialect of BASIC known as 'Microsoft Extended Colour Basic'. Several other computers on the market are also based on this version of BASIC, most notably the Tandy range of colour computers. Microsoft BASIC is easy to use and has a good range of commands to draw lines, circles, and other geometric shapes. Once drawn, these shapes may be coloured in to give impressive screen displays for little programming effort.

The Dragon 32 has seven levels of resolution, giving the user the ability to work with the screen divided into 512 individual points at the lowest level, and up to 49,152 points at the highest. There are eight colours available, but the choice may be limited to four or even two colours when working in high resolution.

Modes Of Resolution

The normal 16 rows by 32 columns character screen forms the lowest level of resolution and the PRINT@ command enables a character to be placed in any one of the 512 screen locations. As well as the normal character set there are also 16 low resolution graphics characters available in eight colours.

The next mode of resolution divides the screen into 32 rows and 64 columns. The size of each square in this mode is therefore a quarter of that of a normal character. Points of this size can be plotted on the screen by the SET command and may be rubbed out by the RESET command.

Both of the above modes can be displayed at the same time and are termed the low resolution text screens. There are also five levels of high resolution screens, but these cannot be displayed simultaneously or with the low level screens. The five high resolution modes offer choices based on the standard of resolution and the number of colours available and are selected using the PMODE command.

PMODE	Resolution	Colours Available
0	128*96	2
1	128*96	4
2	128*192	2
3	128*192	4
4	256*192	2

There is, of course, a trade-off between resolution, colour and the amount of memory needed to store the screen information and this must be taken into account when writing large BASIC programs that also use high resolution displays.

Although there are only a limited number of colours available in high resolution, the Dragon does have a facility for selecting one of two colour sets. This is accomplished by the SCREEN command. For example, SCREEN 1,0 selects a high resolution screen and colour set 0. SCREEN 1,1 again selects a high resolution screen but this time an alternative colour set is used.

PAINT
This command is very useful in assisting the programmer to produce interesting pictures. Using PAINT causes the computer to start colouring in the screen from a given point until a boundary line is reached. This means that circles, triangles and any other closed shape can be filled in simply.

DRAW
DRAW mimics the movement of the pencil on the screen, allowing the user to draw lines in any one of four directions. The DRAW command will also allow the completed picture to be rotated or enlarged.

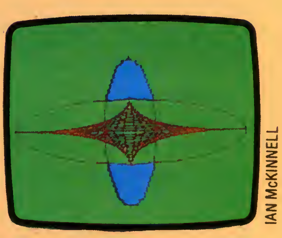
GET and **PUT**
GET instructs the computer to store a screen display in its memory and PUT causes such a display to be reprinted on the screen.

PSET and **PRESET**
These commands are the high resolution equivalents of SET and RESET discussed earlier and switch a particular point on the screen either on or off. The colour of the point can also be determined.

LINE
The LINE command joins two specified points together with a straight line in high resolution.

CIRCLE
CIRCLE allows the user to draw high resolution circles with a given centre and radius. Fractions of a whole circle may also be drawn to form arcs and the circular shape may be condensed to produce ellipses.

The Dragon 32 is a reasonably priced computer with many advanced commands to aid graphics programming. It is more suited to uses that involve static displays rather than those that require fast-moving action. The high resolution mode commands, in particular, make this an ideal computer for the adventurous-minded child. The Dragon's main drawback is its inability to display both text and high resolution graphics on the screen simultaneously. This means that it cannot be used to display statistical data in the form of bar charts or pie charts.



IAN McKINNELL

Colour Command
This display is typical of the effects that can be achieved on a Dragon using just a few of its high level commands

High Resolution
Here is a short program for the Dragon 32 to demonstrate some of its high resolution capabilities. The program uses PMODE 3; this is not the highest mode but it does allow some use of colour.

```
10 PCLS:PMODE3,1
20 SCREEN1,0
30 COLOR 0, 1
40 FOR X=0 TO 127 STEP 10
50 LINE(X,85)-(127,85-X/3),
    PSET
60 LINE(X,85)-(127,85+X/3),
    PSET
70 LINE(255-X,85)-(127,85-
    X/3),PSET
80 LINE(255-X,85)-(127,85+
    X/3),PSET
90 NEXT X
100 CIRCLE(127,85),128,4,0.3
110 CIRCLE(127,85),30,4,3
120 PAINT(130,30),3,4
130 PAINT(130,130),3,4
140 GOTO 140
150 END
```

Order Of Play

The ability to sort information into order is essential to most programs, and there are many ways of doing it

Bubble Sort
This diagram illustrates the Bubble Sort for a reduced hand of nine cards (T is the Ten card). The ordered part of the hand grows from the right-hand end with each pass. The 1 and 2 underneath the hand of cards indicates the two cards currently being compared

```
Begin Sort
2 8 9 3 T 5 K 6 7  Begin Pass 1
1 2
8 2 9 3 T 5 K 6 7
  1 2
8 9 2 3 T 5 K 6 7
    1 2
8 9 3 2 T 5 K 6 7
      1 2
8 9 3 T 2 5 K 6 7
        1 2
8 9 3 T 5 2 K 6 7
          1 2
8 9 3 T 5 K 2 6 7
            1 2
8 9 3 T 5 K 6 2 7
              1 2
8 9 3 T 5 K 6 7 2  End Pass 1
9 8 T 5 K 6 7 3 2  End Pass 2
9 T 8 K 6 7 5 3 2  End Pass 3
T 9 K 8 7 6 5 3 2  End Pass 4
T K 9 8 7 6 5 3 2  End Pass 5
K T 9 8 7 6 5 3 2  End pass 6
End Sort
```

Insertion Sort
With the Insertion Sort, the ordered part of the list grows from the left-hand end. Cards are moved directly to their correct position in the list as they are inspected

```
Begin Sort
2 8 9 3 T 5 K 6 7
2 1
8 2 9 3 T 5 K 6 7
2   1
9 8 2 3 T 5 K 6 7
    2 1
9 8 3 2 T 5 K 6 7
2       1
T 9 8 3 2 5 K 6 7
      2   1
T 9 8 5 3 2 K 6 7
2           1
K T 9 8 5 3 2 6 7
        2     1
K T 9 8 6 5 3 2 7
        2       1
K T 9 8 7 6 5 3 2
End Sort
```

Sorting is one of the most widely used computer operations, but it is a task at which computers are, by their own standards, highly inefficient. According to operational research, between 30 and 40 per cent of all computing time is spent in sorting, and if you add the associated tasks of merging data and searching for specific items, then the figure probably rises above 50 per cent.

Programmers have probably spent as much time inventing sort algorithms (general methods of solving problems) as computers have spent doing the actual sorting. Advanced sorting methods are extremely difficult to analyse, but it is quite easy to understand the simplest methods computers use to sort data with the aid of the example of sorting a pack of playing cards.

Lay 13 cards of the same suit on a table. Arrange them in a line, in no particular order, but the Ace and the Two should not be at the right-hand end of the line. The cards are to be sorted into descending order (King, Queen, Jack... Ace), starting at the left. This is an almost trivial task for us, and requires so little thought that it is difficult to describe exactly how we might do it. If, however, you were to specify that only one card can be moved at a time, that no card can be placed on top of another, and that the cards are to cover as little of the table as possible, the task becomes a lot less trivial, and an efficient method is hard to determine. In this analogy the cards are pieces of data, the maximum surface covered corresponds to the computer memory required, and you are the program. How do you solve the problem?

1) Put a coin below the leftmost card to act as a position marker and to remind you where you are in the sort. Compare the marked card with the card to its right. Are they in descending order? If they are not, swap their positions, leaving the coin where it is, and obeying the rule of only moving one card at a time and not placing cards on top of each other. Notice what you have to do to swap them.

2) When the two cards are in order, move the coin one place to the right and repeat Step 1. You are now in a loop that will end once you move the coin into the rightmost position. Reaching this position is called making a 'pass' through the cards.

3) At the end of the first pass look at the cards. The Ace, which is the lowest card in the suit, has found its way to the rightmost end of the line, and so is in its correct place. If you make a further pass through the cards, as detailed in Steps 1 and 2, the Two card will be moved to its correct place. This is repeated, through pass after pass, until the whole suit is in descending order.

You may have noticed several drawbacks to this method. It is very tedious; it is not economical, as simply exchanging the positions of two cards requires three different operations; and, above all, many of the comparisons made between different cards are unnecessary. For example, after one pass the Ace is in its correct place, so there's no point moving the coin into position 13 (where no comparison is possible, anyway). On the second pass, because the card on the right is in its correct place, there was no need to move the pointer to position 12. In general, each pass will end one place to the left of the endpoint of the previous pass.

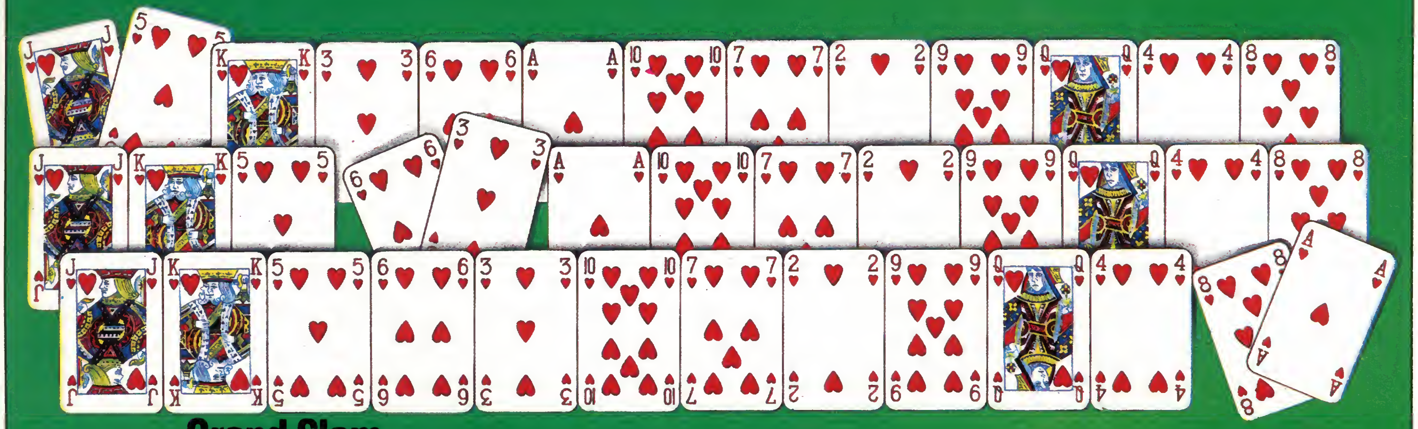
Knowing where to stop is another problem. A computer will continue comparing cards indefinitely unless it is told to stop. The only sure rule is: stop after a pass with no swaps. In other words, if you've gone through the data without altering its order, then it must be in order.

The method of sorting we have investigated is called the 'Bubble Sort'. Its advantages include simple programming techniques, little use of extra memory and reasonable efficiency with small amounts of partially ordered data. These are the criteria by which a sort algorithm must be judged, although when the data to be sorted is extensive, speed may have to be sacrificed for economy of memory simply because computer memory may not accommodate both the raw data and a sorted copy. For this reason, we'll ignore algorithms that require taking data from one array and moving it to the sorted position in a second array. The second method of simple sorting is based more directly on the way that we would sort cards.

1) Lay the shuffled cards out again and place a penny coin beneath the second card from the left. Whichever card the penny is beneath at the beginning of each pass, we will call the 'penny card'.

2) Push the penny card out of the line, leaving a gap, and place a twopenny coin beneath the card's immediate left. Call this card the twopenny card.

3) Compare the penny card with the twopenny card. If they're in order, then push the penny card back into place and skip to Step 4. If they're not in order, then push the twopenny card into the gap and move the twopenny coin one place to the left to mark a new twopenny card (if the twopenny card is at the extreme left, this will not apply, so



Grand Slam

One way to illustrate a Bubble Sort is with a complete suit of cards that have to be sorted so that the King ends up on the left and the Ace on the right. First the leftmost two cards are compared, and because they are found to be out of order, they are swopped over. Then the second and third cards are compared, and again swopped. By the fifth comparison, this sort method has picked up the Ace, and in all subsequent comparisons, the Ace is swopped from left to right, until at the end of the first 'pass' it has 'bubbled' its way to the right-hand end. By repeating this whole process for the second pass, the two will end up next to the Ace. However, it may take up to 12 such passes before all the cards are in order

TONY LODGE

place the penny card in the gap and proceed to Step 4).

Compare this twopenny card with the penny card (the displaced one). Now repeat Step 3 until the correct position for the penny card is found.

4) Move the penny one position to the right and repeat Steps 2 and 3. When you can't move the penny any further right, the cards will all be in order.

This is called an 'Insertion Sort', and is very similar to the way people sort a hand of cards. Although it is a little harder to program than a Bubble Sort it is a far more efficient method. Later in the course, we will look at some more complex algorithms for sorting data.

```
9   REM**********************
10 REM* SORT ALGORITHMS    *
11 REM**********************
100 INPUT"HOW MANY ITEMS TO BE SORTED";LT
150 IF LT<3 THEN LET LT=3
200 LET LT=INT(LT)
250 DIM R(LT),C(LT)
300 LET Z=0:LET Q=0:LET P=0
350 LET I=1:LET O=0:LET II=2:LET TH=2
400 INPUT"HOW MANY TESTS ";N
450 FOR CT=I TO N
500 GOSUB 4000
550 FOR SR=I TO TH
600 GOSUB 5000
650 PRINT:PRINT:PRINT:PRINT
700 PRINT "TEST #";CT+SR/10
750 INPUT"HIT RETURN TO BEGIN SORT";A$
800 PRINT "THE UNSORTED LIST IS"
850 GOSUB 3000
900 ON SR GOSUB 6000,7000
950 PRINT "THE SORTED LIST IS"
1000 GOSUB 3000
1050 NEXT SR
1100 NEXT CT
1150 END
2999 REM**********************
3000 REM*  PRINT THE LIST    *
3001 REM**********************
3100 FOR K=I TO LT
3200 PRINT R(K);
3300 NEXT K
3400 PRINT
3500 RETURN
3999 REM**********************
4000 REM*   RND GENERATOR    *
4001 REM**********************
4100 RANDOMIZE
4200 FOR K=I TO LT
4300 LET C(K)=INT(100*RND)
4400 NEXT K
4500 RETURN
4999 REM**********************
5000 REM*  RND REGENERATOR   *
5001 REM**********************
5100 FOR K=I TO LT
5200 LET R(K)=C(K)
5300 NEXT K
5400 PRINT:PRINT
5500 RETURN
5999 REM**********************
6000 REM*     BUBBLE         *
6001 REM**********************
6050 PRINT "BUBBLE SORT - GO !!!!!
6100 FOR P=LT-I TO I STEP-I
6150 LET F=-I
6200 FOR Q=I TO P
6250 LET Z=Q+I
6300 IF R(Q)<R(Z) THEN LET D=R(Q):
     LET R(Q)=R(Z):LET R(Z)=D:LET F=0
6350 NEXT Q
6400 IF F=-I THEN LET P=I
6450 NEXT P
6500 PRINT "BUBBLE SORT - STOP !!!!!
6550 RETURN
6999 REM**********************
7000 REM*     INSERTION      *
7001 REM**********************
7050 PRINT "INSERTION SORT - GO !!!!!
7100 FOR P=II TO LT
7200 LET D=R(P)
7300 FOR Q=P TO II STEP-I
7400 LET R(Q)=R(Q-I)
7500 IF D<=R(Q) THEN LET R(Q)=D:LET Q=II
7600 NEXT Q
7700 IF D>R(I) THEN LET R(I)=D
7800 NEXT P
7850 PRINT "INSERTION SORT - STOP !!!!!
7900 RETURN
```

High-Speed Sort

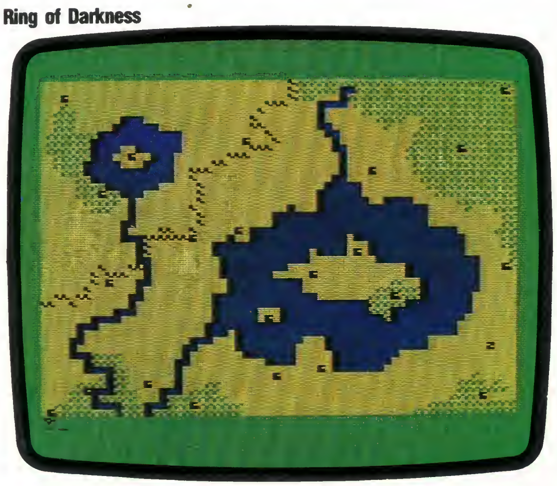
This BASIC program demonstrates the difference in efficiency between a Bubble Sort and Insertion Sort. The code has been written with speed in mind, so we have not documented the operation of the routines. The listing should run on most machines, but see page 215 for ON . . . GOSUB flavours, and page 175 for RND and RANDOMIZE

Amazing Facts

People have long been fascinated by mazes — and maze games on the home computer are no exception

Mazes have always been a source of fascination and enjoyment to both young and old alike, whether they are big enough to get lost in, or small enough to hold in the palm of the hand. The maze has, in fact, become the basis of a huge variety of computer games, ranging from a very simple two-dimensional aerial view of a maze, right up to extremely complex mazes in three dimensions. The latter sort actually simulate a view of the maze from within, so that the player is encouraged to imagine that he is inside a real maze. To help him get his bearings, or confuse him even further, some of these three-dimensional mazes also combine brief glimpses of an aerial view of the maze.

Ring of Darkness



Ring of Darkness
Though this game for the Dragon is really an Adventure – style game, it contains a three-dimensional maze as one of its major elements. Pits and ladders allow you to move up and down

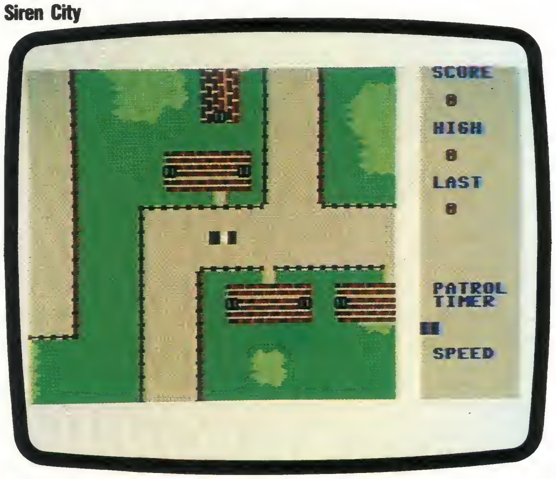
As mazes have become more sophisticated in their visual and sound effects, so have the imaginations of their programmers been allowed to wander. A player wishing to take a leisurely stroll around a maze should avoid those that conceal man-eating monsters. An example of such games is 3D Glooper (available for the Commodore 64), in which the player searches the maze for special floor tiles, and can be attacked at any moment by screen-filling monsters. The impending arrival of these creatures is announced, however, by the steady munching sounds from their approaching jaws.

Atic Atac (Spectrum) is a fully animated chase in which the player can assume any of three different characters. The maze is a multi-level series of pits, stairways and large dungeons through which you race against time. The dungeons are occupied by a variety of graphically depicted creatures and objects.

A program that comes close to simulating what it actually feels like to be travelling through a maze is Way Out. The view is in true three-dimensional perspective, and as you move your joystick fractionally to the left or right the scene shifts proportionately in that direction.

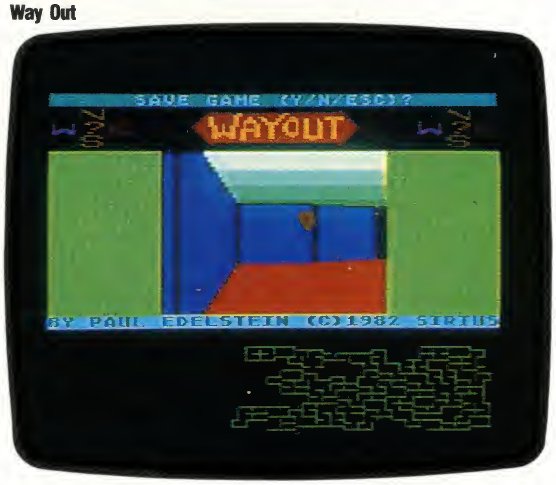
Let us now consider some of the basic programming techniques used in constructing mazes.

Siren City



Siren City
This Commodore 64 game is a development on the traditional 'aerial view' game. A police car patrols a city, complete with roads and buildings

Way Out



SAVE GAME (Y/N/ESC)?

BY PAUL EDELSTEIN (C)1982 SIRIUS

Way Out
A realistic three-dimensional image can be achieved on a Spectrum with Way Out. Move the joystick fractionally, and your view will change also

IAN McKINNELL

Making Mazes

The usual way of storing information about a maze is by using a two-dimensional array — M$(ROW,COLUMN), for example. Each cell of the array would define the characteristics of that cell of the maze. You could, for example, use a string of four characters to represent south, west, north and east. Zero could indicate the absence of a wall and one the presence of a wall. Thus, if M$(5,6) contained the string "1011" this would indicate that the cell in row five, column six was bounded by walls to the south, north and east.

To save on memory space, the array could be numeric rather than string, and the four-digit number regarded as a binary number. In our example above, the cell containing north, east and south walls would contain the number 11 (1011).

All cells would start with four walls. By randomly generating the entrance from anywhere along the perimeter, the next cell could be chosen at random from any of the three adjacent cells. When that cell is chosen, the sequence continues — randomly selecting a cell from any of the three adjacent cells, disregarding the one you have just come from.

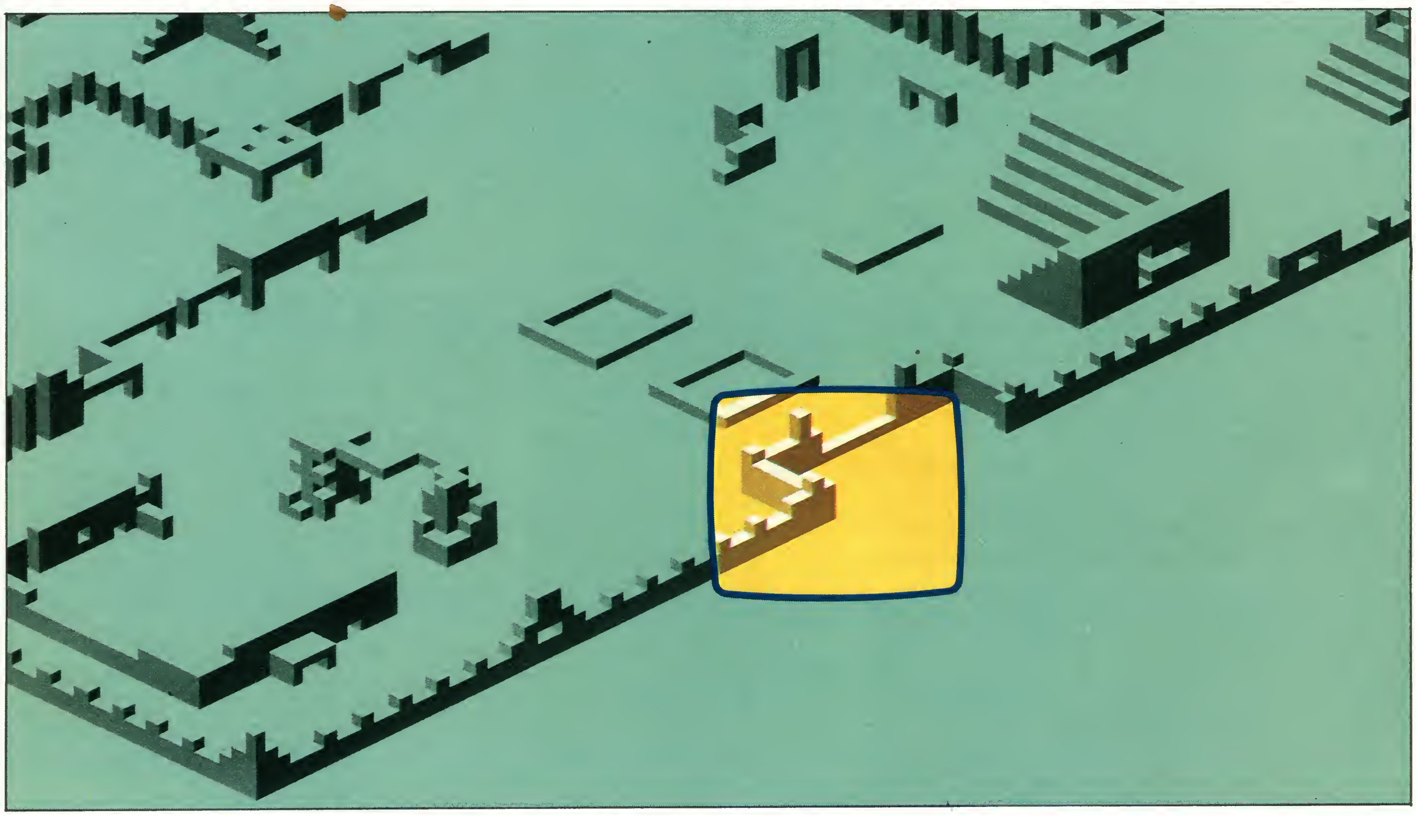
As each new direction is chosen, the appropriate 'wall' is removed from the cell you are about to leave and the cell you are about to enter. Checks must be made to ensure you don't move outside the boundaries of the maze (unless the particular perimeter cell is to be the exit point), or create closed circuits (all parts of a maze should be accessible from any other part).

When a cell that has fewer than four walls (i.e. a cell that has already been visited) is encountered, the program must choose another of the remaining adjacent cells. If all adjacent cells have been visited, the program has to 'step back' to the previous cell visited and take a new branch.

Another method of recording the characteristics of a cell is by a more sophisticated use of binary numbering, which is especially useful for displaying three-dimensional perspectives. There are 16 possible ways a cell can be built: no walls, walls on all sides, a single wall (four possibilities), walls on opposite sides (two possibilities), two adjacent walls (four possibilities), three adjacent walls (four possibilities).

Using the appropriate binary number (0-15) for each of these, it is possible to 'rotate' the number left or right to obtain the appropriate view faced by the player. For example, a north wall could be represented by 2 (0010), south wall by 8 (1000), east wall by 1 (0001) and west wall by 4 (0100). If the player facing north in a 'west wall only' room (4) turned to face the west, his view of the room would now be one bounded by a north wall (since facing forward in a three-dimensional display is always to the 'north'). As the player turned to his left (the west), moving the bit pattern one place to the right supplies the description we want, i.e. west wall binary 0100 (decimal 4) becomes binary 0010 (decimal 2 — a north wall!) The bits are moved in the opposite direction when turning right, twice for an about turn. It is necessary, of course, to include a system for 'wrapping around' the bits that are lost from the left- or right-hand end of the half-byte during this process, otherwise the identifying characteristics of a cell will be changed each time the player turns within it. A cell originally defined as 0011, for instance (walls to the north and east), must become 0110 if the player turns to the right, and 1100 if he turns completely around.

In machine code, there are special instructions for rotating binary numbers left and right. In BASIC, a four-bit binary number expressed as a decimal number in the range 0-15 can be shifted left by multiplying the number by two, and then subtracting 15 if the result exceeds 15. To rotate right: divide by two if it is an even number, or add 15 and then divide by two for an odd number.



Ant Attack
When this game is run on the Spectrum, the computer's screen acts as a window onto a large maze-like field of play. As the play moves, the screen will move, revealing more of the scene

Aquarius

It comes from a company famous for their toys, but the Aquarius is a serious computer at a bargain price

With its Z80 processor and button-type keyboard, the Mattel Aquarius is in the Spectrum class of microcomputer. However, in many ways it is a much more flexible machine, largely because its built-in expansion bus has been well-exploited by its designers.

A variety of expansion modules can be connected through this bus, ranging from small RAMpacks of 4 Kbytes to a large expansion chassis. Perhaps the most useful of these is the 'small expansion chassis', which has two slots for extra memory or program packs, as well as two extra sound channels and two hand controllers. Plugging a 16 Kbyte RAMpack in one slot and a proprietary ROMpack, such as Finplan, in the other would give a quite versatile system.

The 4 Kbytes of RAM built into the machine is hardly generous, but with expansion of up to 64 Kbytes of RAM with the large expansion chassis, it's possible to run as large a machine as any home computer.

The keyboard and display of the Aquarius, however, lack the quality of larger machines. There's no space bar, and the keys don't respond very sensitively or quickly, so it's not suitable for touch-typing. The 24 line by 40 character screen, though bigger than some, is not adequate for small business use.

The display has 16 colours that can be used for either the text or the background. Though lacking user-definable characters, it has 256 displayable symbols, including upper and lower case letters, and a selection of graphics symbols. It can also be used as a 320 × 192 pixel high-resolution screen. The display is output to the television, with no provision for monitor output. The quality is average with a noticeable bias towards blue shades and slightly blurred characters, but the picture is steady and bright, with a good range of colour.

Aquarius Printer
This low-cost printer uses a thermal printing mechanism and so requires special thermal paper. It can print at a rate of 80 characters per second, across a total width of 40 columns. A four-colour printer/plotter is also available

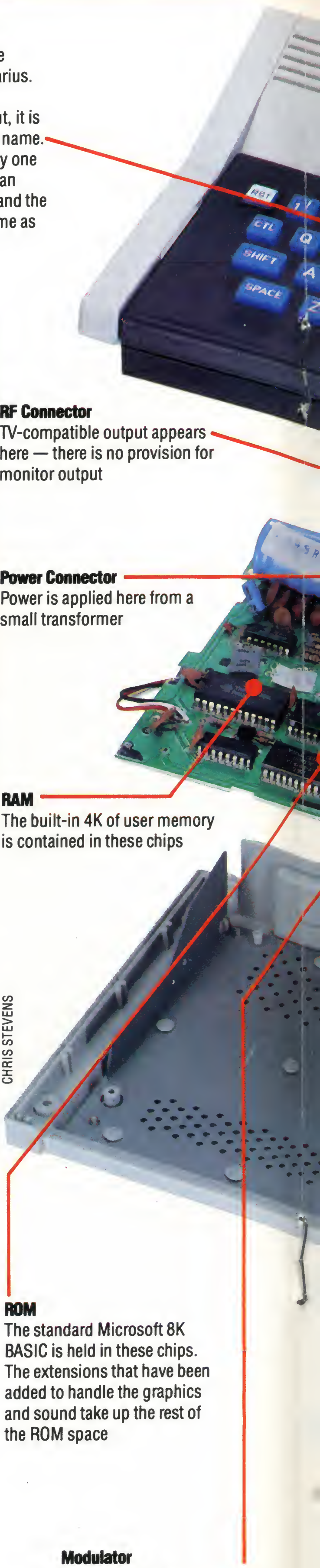
Mini Expander
This device features two cartridge ports, allowing a program cartridge and memory pack to be connected simultaneously. It also features the two 'hand controllers' and three additional sound channels

CHRIS STEVENS

Sound is available on this machine, although it lacks the sophisticated envelope and waveform controls found on others. A fairly standard Microsoft BASIC is built in, but Extended BASIC and an Aquarius LOGO are promised.

One of the most interesting add-ons planned for the Aquarius is the BSR X-10 system, which can control a range of household appliances. This system allows up to 255 different electrical devices to be controlled in response to signals generated by a central unit. No additional wiring is needed, since these signals are in the form of pulses sent down the domestic ring main. The pulses aren't large enough to make any difference to the mains current, but an X-10 detector plugged into any mains wall socket can pick up the code and alter the current supplied to its local appliance according to the command sent.

The controller unit is programmed in weekly cycles by the Aquarius, and during this programming operation the computer is unavailable for other uses. Provided the preset program is satisfactory, the computer is free for ordinary use at any other time.



The Aquarius Keyboard
The keyboard is one of the weaker points of the Aquarius. Though claimed to be a 'standard' QWERTY layout, it is only just deserving of the name. There is no space bar, only one SHIFT key, RETURN is in an unconventional position and the spacing isn't quite the same as on a typewriter

RF Connector
TV-compatible output appears here — there is no provision for monitor output

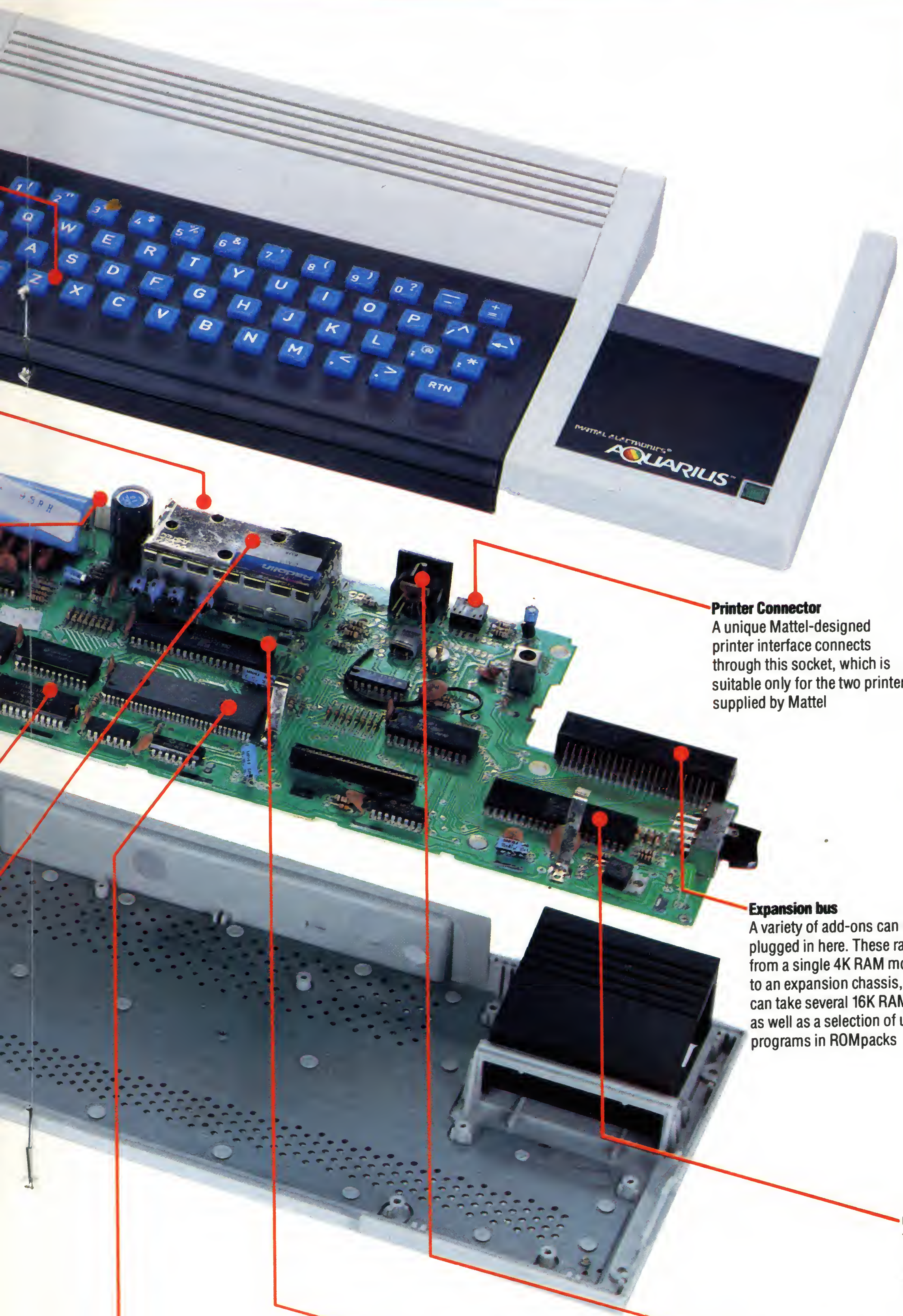
Power Connector
Power is applied here from a small transformer

RAM
The built-in 4K of user memory is contained in these chips

ROM
The standard Microsoft 8K BASIC is held in these chips. The extensions that have been added to handle the graphics and sound take up the rest of the ROM space

Modulator
The screen display signal is converted into a standard TV signal, and appears on Channel 36

CHRIS STEVENS



Printer Connector
A unique Mattel-designed printer interface connects through this socket, which is suitable only for the two printers supplied by Mattel

Expansion bus
A variety of add-ons can be plugged in here. These range from a single 4K RAM module to an expansion chassis, which can take several 16K RAMpacks as well as a selection of useful programs in ROMpacks

CPU
The processor is a Z80, which runs at a clock frequency of 3.5 MHz

CRT Controller
Designing the electronics that control the video display is now the most important aspect of computer design. This controller chip is larger than the microprocessor itself

Security Chip
This custom-designed chip is intended to make it very difficult for anyone other than the manufacturers to produce program cartridges that will run on the Aquarius

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

AQUARIUS

PRICE
£49.95

SIZE
345 × 150 × 55mm

CLOCK SPEED
3.5 MHz

MEMORY
10 Kbytes of ROM, plus 4 Kbytes of RAM, expandable to 64 Kbytes

VIDEO DISPLAY
24 lines of 40 characters, 16 colours with background and foreground independently settable; 256 pre-defined characters but no user-definable characters

INTERFACES
Cassette, printer, expansion bus

LANGUAGES SUPPLIED
Microsoft BASIC

OTHER LANGUAGES AVAILABLE
Microsoft Extended BASIC and Aquarius LOGO have been promised by Mattel. These will be in ROMpack form

COMES WITH
Installation manual and BASIC manual, TV lead

KEYBOARD
49 button-style keys. The reset button is physically shielded to prevent it from being accidentally pressed

DOCUMENTATION
The documentation is particularly good for beginners, with a useful set of flip-cards that describe each major function of the machine and the built-in BASIC. There is a shortage of technical detail, but for the market that the Aquarius is aimed at it sets a good example

Branching Out

As a long program is developed, its structure takes on the appearance of a tree, with more branches at each successive stage of refinement

In the last instalment of the Basic Programming course, we took a look at some of the problems involved in searching through a list to find a specific item — assuming that the list had already been sorted into order. This is a topic to which we will return in more detail when the time comes to start writing search routines. In the meantime, however, we will develop the theme of top-down programming to produce code for the second two parts of the main program. This contains four calls to subroutines or procedures:

MAIN PROGRAM

```
BEGIN
    INITIALISE (procedure)
    GREET (procedure)
    CHOOSE (procedure)
    EXECUTE (procedure
END
```

The first procedure, *INITIALISE*, will involve numerous fairly complex activities — setting up arrays, reading data into them, performing various checks and so on — and we will leave the details of this procedure until later. The next two parts of the main program comprise the GREET and CHOOSE procedures. In developing these procedures, we will suggest a methodology that helps prevent the many layers involved in top-down program development from becoming disorganised and confusing.

The problem with the top-down refinement approach to program development is that the number of steps needed before we are ready to start coding into a high level language is indeterminate. Two or three steps may be enough for simple procedures, but more difficult procedures may require many steps before the problem has been sufficiently analysed to allow 'source code' (as the high level language program is called) to be written. This means that writing a program using this method is similar to drawing a tree lying on its side. As the 'branches' proliferate (that is, as the refinements become more detailed) they take up more space on the page. Eventually, it becomes impossible to fit everything onto a single sheet, and that is the point where it becomes easy to lose track of what's going on.

One very effective way to organise the documentation of the program is to number the stages of its development systematically. We have used Roman numerals to indicate the level of refinement and Arabic numerals to indicate the subsection of the program. A separate sheet of loose-leaf paper is then used for each level of refinement and the pages for each program block or module can be easily kept together. Here is the numbering system for our program:

I MAIN PROGRAM

```
BEGIN
    1. INITIALISE
    2. GREET
    3. CHOOSE
    4. EXECUTE
END
```

As mentioned above, we are leaving the development of INITIALISE for the moment, and concentrating on developing the GREET and CHOOSE procedures.

II 2 (GREET)

```
BEGIN
    1. Display greeting message
    2. LOOP (until space bar is pressed)
         ENDLOOP
    3. Call *CHOOSE*
END
```

III 2 (GREET) 1 (display message)

```
BEGIN
    1. Clear screen
    2. PRINT greeting message
END
```

III 2 (GREET) 2 (LOOP wait for space bar)

```
BEGIN
    1. LOOP (until space bar is pressed)
            IF space bar is pressed
            THEN
         ENDLOOP
END
```

III 2 (GREET) 3 (call *CHOOSE*)

```
BEGIN
    1. GOSUB *CHOOSE*
END
```

At this point it should be clear that **III-2-1** and **III-2-3** are ready to be coded directly into BASIC, but that **III-2-2** needs another stage of refinement:

IV 2 (GREET) 2 (LOOP)

```
BEGIN
    1. LOOP (until space bar is pressed)
            IF INKEY$ is not space THEN continue
         ENDLOOP
END
```

We are now at the point where all the coding into

BASIC for GREET can be tackled with little further refinement:

IV 2 (GREET) 1 (display message) BASIC CODE

```
REM *GREET* SUBROUTINE
PRINT
PRINT
PRINT
PRINT
PRINT TAB(12);“*WELCOME TO THE*”
PRINT TAB(9);“*HOME COMPUTER COURSE*”
PRINT TAB(6);“*COMPUTERISED ADDRESS
   BOOK*”
PRINT
PRINT TAB(5);“(PRESS SPACE BAR TO CONTINUE)”
```

V 2 (GREET) 2 (LOOP wait for space bar) BASIC CODE

```
LET L = 0
FOR L = 1 TO 1
IF INKEY$ < > “    ” THEN LET L = 0
NEXT L
```

IV 2 (GREET) 3 (call *CHOOSE*) BASIC CODE

```
GOSUB *CHOOSE*
RETURN
```

Notice that we have now started to initialise variables in the various routines that we write, by using statements of the form LET I = 0. Strictly speaking, this is unnecessary in some of the circumstances in which we have used it. Nevertheless, it is a good habit to get into if you can remember, and if you have enough RAM space available. There are three reasons: first because having a list of LET statements at the start of any routine serves as a useful reminder of what local variables that routine uses. Secondly, because you cannot be sure of what was left in a variable from the last time it was used in a routine (though this does not always matter). Thirdly, as we shall be explaining to you later in the course, putting in statements of the form LET I = 0 in the right order can speed up the execution of a program.

We have changed the way in which we use a FOR...NEXT loop to simulate a DO...WHILE or REPEAT...UNTIL structure from previous instalments of the course. Instead of using FOR I = 0 TO 1 or FOR I = 0 to 1 STEP 0, we are now using FOR I = 1 to 1. This will run correctly on all the home computers we regularly cover, where the other methods required 'Basic Flavours' for various machines. FOR I = 1 TO 1...NEXT I will execute the loop just once. However, if anywhere in the body of the loop I is set to 0 then the loop will execute again, and so on. We can either insert a LET I = 0 statement as the result of an exit condition failing or we can set I to 0 immediately after the FOR statement, and set it to 1 if the exit condition succeeds. Thus, both the following loops achieve the same objective:

```
FOR I = 1 TO 1
IF INKEY$ < > “    ” THEN LET I = 0
NEXT I
```

or

```
FOR I = 1 TO 1
LET I = 0
IF INKEY$ = “    ” THEN LET I = 1
NEXT I
```

The BASIC code we have just produced is all that is needed for the complete GREET block in the main program. We haven't put in line numbers because we can't really do that until all the program modules are ready for final coding. For instance, we do not know at this stage what the appropriate line numbers are for the GOSUB commands. If you want to test the module at this stage, it will be necessary to create some dummy inputs and dummy subroutines. Some points to note about this program fragment are the use of the TAB function and the 'clear screen' statements. TAB moves the cursor along the line by the number (the 'argument') specified in the brackets. The numbers we have given will print the message neatly centred in a screen 40 characters wide. If your display has less than this (for example, the Spectrum displays 32 characters per line) or more (larger computers usually display 80 characters), these TAB arguments will need to be altered accordingly. The instruction to clear the screen in many versions of BASIC is CLS, but the version of Microsoft BASIC used to develop this program does not support this. Instead, we have used PRINT CHR$(12), since our machine uses ASCII 12 as its 'clear screen' non-printable character — others commonly use ASCII 24 for the same function.

```
10 REM DUMMY MAIN PROGRAM
20 PRINT CHR$(12)
30 GOSUB 100
40 END
100 REM *GREET* SUBROUTINE
110 PRINT
120 PRINT
130 PRINT
140 PRINT
150 PRINT TAB(12);“*WELCOME TO THE*”
160 PRINT TAB(9);“*HOME COMPUTER COURSE*”
170 PRINT TAB(6);“*COMPUTERISED ADDRESS
     BOOK*”
180 PRINT
190 PRINT TAB(5);“(PRESS SPACE BAR TO
     CONTINUE)”
195 LET L = 0
200 FOR L = 1 TO 1
210 IF INKEY$ < > “    ” THEN LET L = 0
220 NEXT L
230 PRINT CHR$(12)
240 GOSUB 1000
250 RETURN
1000 REM DUMMY SUBROUTINE
1010 PRINT “DUMMY SUBROUTINE”
1020 RETURN
```

We will now use exactly the same approach to refine the CHOOSE procedure.

II 3 (CHOOSE)

```
BEGIN
   1. PRINT menu
```

```
    2. INPUT CHOICE
    3. Call CHOICE subroutine
END
```

III 3 (CHOOSE)1 (PRINT menu)

```
BEGIN
    1. Clear screen
    2. PRINT menu and prompt
END
```

III 3 (CHOOSE) 2 (INPUT CHOICE)

```
BEGIN
    1. INPUT CHOICE
    2. Check that CHOICE is within range
END
```

III 3 (CHOOSE) 3 (call CHOICE)

```
BEGIN
    1. CASE OF CHOICE
         ENDCASE
END
```

III-3-1 (PRINT menu) can now be coded into BASIC:

IV 3 (CHOOSE) 1 (PRINT menu) BASIC CODE

```
REM CLEAR SCREEN
PRINT CHR$(12): REM OR 'CLS'
PRINT
PRINT
PRINT
PRINT
PRINT "1. FIND RECORD (FROM NAME)"
PRINT "2. FIND RECORD (FROM INCOMPLETE
   NAME)"
PRINT "3. FIND RECORD (FROM TOWN)"
PRINT "4. FIND RECORD (FROM INITIALS)"
PRINT "5. LIST ALL RECORDS"
PRINT "6. ADD NEW RECORD"
PRINT "7. CHANGE RECORD"
PRINT "8. DELETE RECORD"
PRINT "9. EXIT & SAVE"
```

III-3-2 (INPUT CHOICE) and **III-3-3 (call CHOICE)**, however, need further refinement. Let's look first at the next level of development of **III-3-2**.

Assigning a numeric value to the variable CHOICE is perfectly simple: after the prompt, an INPUT CHOICE command will do this. However, there are only nine possible choices. What would happen if we mistakenly entered a 0, or 99? Since the CHOICE we make will determine which part of the program is called next, we want to be sure that unwanted errors are not caused, so we need to perform a 'range checking' procedure. This is a small routine that checks to see if the number input is within the acceptable range before allowing the program to continue. Here is a sample routine designed to trap an erroneous input.

RANGE CHECKING ROUTINE

```
1 REM ROUTINE
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTER 1–9"; CHOICE
40 IF CHOICE <1 THEN LET L = 0
50 IF CHOICE >9 THEN LET L = 0
60 NEXT L
70 PRINT "CHOICE WAS ";CHOICE
80 END
```

Many versions of BASIC can make this routine simpler by including a boolean operator in the test like this:

```
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTER 1–9";CHOICE
40 IF CHOICE < 1 OR CHOICE >9 THEN LET L = 0
50 NEXT L
60 PRINT "CHOICE WAS ";CHOICE
70 END
```

These routines also illustrate another point about the INPUT statement. INPUT causes the program to stop and wait for an input from the keyboard. BASIC does not know when the whole number has been entered until the RETURN key has been pressed, so you will also have to remember to press RETURN after entering the number.

A more 'user friendly' approach would be to have the program continue as soon as a valid number had been entered. This is possible using the INKEY$ function. Here, BASIC reads a character from the keyboard whenever INKEY$ is encountered. The program does not stop, however, and will proceed to the next part of the program without pausing. It is usual, therefore, for INKEY$ to be used within loops. The loop to check for a key being pressed can be IF INKEY$ = "" THEN... — in other words, if the key being pressed is 'nothing' (that is, no key is being pressed), go back and check again. A suitable loop for our purposes would be:

```
LET I = 0
FOR I = 1 TO 1
LET A$ = INKEY$
IF A$ = "" THEN LET I = 0
NEXT I
```

The only disadvantage of using INKEY$ is that it returns a character from the keyboard, rather than a numeric. When there is a CASE OF construct, where one out of several choices are made (a multi-conditional branch), it is easier in BASIC to use numbers rather than characters. This is where BASIC's NUM or VAL functions come in. They convert numbers in character strings into 'real' numbers (that is, numeric values, not ASCII codes representing numerals). They are used like this:

```
LET N = VAL(A$) or LET N = NUM(A$)
```

By using the NUM or VAL functions, we can have the program convert inputs, using INKEY$, into numeric variables. This removes the need to use the RETURN key after the number key has been pressed. Out-of-range checking is still advisable, however.

The following program fragment involves two loops, one nested within the other. The inner loop waits for a key to be pressed; the outer loop converts the string to a number and checks that it is within range:

```
FOR L = 1 TO 1
   PRINT "ENTER CHOICE (1–9)"
      FOR I = 1 TO 1
         LET A$ = INKEY$
         IF A$ = "" THEN LET I = 0
      NEXT I
   LET CHOICE = VAL(A$)
      IF CHOICE < 1 THEN LET L = 0
      IF CHOICE > 9 THEN LET L = 0
NEXT L
```

Finally, we reproduce a complete program in BASIC for the `*CHOICE*` module, including dummy input and subroutines for testing purposes. We should stress again that the line numbers are for testing purposes only, and will need to be replaced when the final program is put together.

```
10 PRINT CHR$(12)
20 PRINT "SELECT ONE OF THE FOLLOWING"
30 PRINT
40 PRINT
50 PRINT
60 PRINT "1. FIND RECORD (FROM NAME)"
70 PRINT "2. FIND NAMES (FROM INCOMPLETE
   NAME)"
80 PRINT "3. FIND RECORDS (FROM TOWN)"
90 PRINT "4. FIND RECORD (FROM INITIALS)"
100 PRINT "5. LIST ALL RECORDS"
110 PRINT "6. ADD NEW RECORD"
120 PRINT "7. CHANGE RECORD"
130 PRINT "8. DELETE RECORD"
140 PRINT "9. EXIT & SAVE"
150 PRINT
160 PRINT
170 LET L = 0
180 LET I = 0
190 FOR L = 1 TO 1
200 PRINT "ENTER CHOICE (1–9)"
210 FOR I = 1 TO 1
220 LET A$ = INKEY$
230 IF A$ = "" THEN LET I = 0
240 NEXT I
250 LET CHOICE = VAL(A$)
260 IF CHOICE <1 THEN LET L = 0
270 IF CHOICE >9 THEN LET L = 0
280 NEXT L
290 ON CHOICE GOSUB 310,330,350,370,390,410,
     430,450,470
300 END
310 PRINT "DUMMY SUBROUTINE 1"
320 RETURN
330 PRINT "DUMMY SUBROUTINE 2"
340 RETURN
350 PRINT "DUMMY SUBROUTINE 3"
360 RETURN
370 PRINT "DUMMY SUBROUTINE 4"
380 RETURN
390 PRINT "DUMMY SUBROUTINE 5"
400 RETURN
410 PRINT "DUMMY SUBROUTINE 6"
420 RETURN
430 PRINT "DUMMY SUBROUTINE 7"
440 RETURN
450 PRINT "DUMMY SUBROUTINE 8"
460 RETURN
470 PRINT "DUMMY SUBROUTINE 9"
480 RETURN
```

In the next instalment, we will look at file structures and begin refining the INITIALISE procedure.

Basic Flavours

SPECTRUM

In the dummy main program, and throughout, replace PRINT CHR$(12) by CLS, and END by STOP.

RANGE CHECKING ROUTINE

```
1 REM ROUTINE
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT"ENTER 1–9 ";CHOICE
40 IF CHOICE < 1 THEN LET L = 0
50 IF CHOICE > 9 THEN LET L = 0
60 NEXT L
70 PRINT"CHOICE WAS ";CHOICE
80 STOP
```

FINAL LISTING

```
10 CLS
```

then copy the list in the main text until:

```
240 NEXT I
250 LET CHOICE = CODE A$ – 48
260 IF CHOICE<1 THEN LET L=0
270 IF CHOICE>9 THEN LET L=0
280 NEXT L
290 GOSUB (CHOICE*20 + 290)
300 STOP
```

then copy the main list from line 310 to line 480.

TAB

Some versions of the Oric-1 do not obey the TAB command, even though it is part of Oric-1 BASIC: in this case, insert this line at the start of the program:

```
5 LET S$="                    "
```

Between the quotes in this line there should be as many spaces as there are characters on a complete screen line — 40 for an Oric-1. Then whenever the program says TAB(12) replace it by LEFT$(S$,12), copying the number in the TAB statement into the LEFT$( ) function.

CHR$(12)

On the Oric-1, the Dragon 32, the Lynx and the BBC Micro, replace PRINT CHR$(12) by CLS. On the Commodore 64 and the Vic-20 replace CHR$(12) by PRINT"shiftkey+CLR/HOME key": this should result in a 'reverse field heart' being printed. See the manual if you're puzzled.

ON.. GOSUB

This is not available on the Lynx, but can be replaced by line 290 in the final Spectrum listing above.

VARIABLES

See 'Basic Flavours' page 257.

INKEY$

See 'Basic Flavours' page 175, and Commodore owners replace LET A$=INKEY$ by GET A$, and replace IF INKEY$="" THEN by:
GET A$:IF A$="" THEN

TONY LODGE

Mice And Men

Computer designers want to abandon the keyboard in favour of something easier to use. One approach is the mouse

Not long ago computers could only be accessed through large electromechanical typewriters called 'teletypes'. These were noisy, cumbersome and unreliable devices that have since been replaced by the swift and silent Visual Display Unit (VDU) with keyboard. The VDU eliminated many of the problems associated with the teletypes — not least of which was the production of large amounts of punched tape waste paper as the information was keyed in. However, both the mechanical terminal and the VDU-plus-keyboard are restricted by their character-by-character, line-by-line format. The user cannot move quickly around the screen — selecting items from a menu here, altering data there, or changing files and programs — without being faced with the limitations of the keyed cursor format. Freedom from the keyboard is attained when using graphics terminals or playing computer games with trackballs and joysticks, but how can a serious user benefit from these?

Most of the home computers currently available are equipped with four direction cursor controls that can be moved around a program listing or a text document to the position where an amendment needs to be made. But the cursor can be moved only in character- or line-sized steps; the user cannot move it directly to its destination. If the text cursor could be moved like a graphics cursor, which can be freely manipulated under the control of a joystick or trackball, movement of data would be considerably faster.

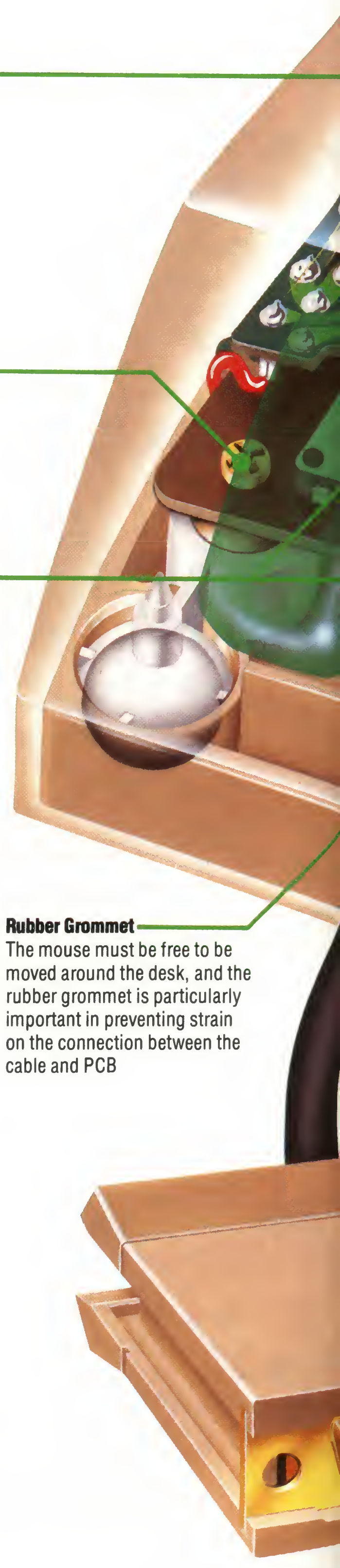
Three Blind Mice
Many of the most recent business microcomputers feature a mouse as standard, and some companies offer units as add-ons to existing machines. Most work by means of a rotating ball on the underside, and feature either one, two or three 'SELECT' buttons on the top

IAN McKINNELL. MICE COURTESY OF MICROSOFT, APPLE & XEROX

A solution to this problem was first explored in the 1960's at the Stanford Research Institute in California; and the first 'mouse' — as the new kind of controller that was developed was called — was patented in 1970. The device was given the name 'mouse' because of its appearance: a mouse is small enough to fit into the palm of the hand; it has a 'tail' (the cable); and the first devices usually had two 'ears' (control buttons). Conventional trackballs and joysticks aren't used because the precision that they provide in positioning the cursor isn't needed.

The mouse operates by detecting its motion across any flat surface in the up/down and left/right directions, as well as combinations of the two. These movements are directly converted to movements of the cursor — or pointer, as it is often called — on the screen. There are two main methods of generating the electrical signals from the movement of the mouse. In both cases, the underside of the mouse features a large ball that rests on the surface across which the mouse is being moved.

The rotation of the mouse's ball-bearing is transferred to internal cylindrical rollers. In one system, the ends of these cylinders are fitted with code wheels that have alternating tracks of conducting and non-conducting material. The pulses received are counted by the mouse's operating software and enable it to give a direct reading for the cursor's position on the screen. In



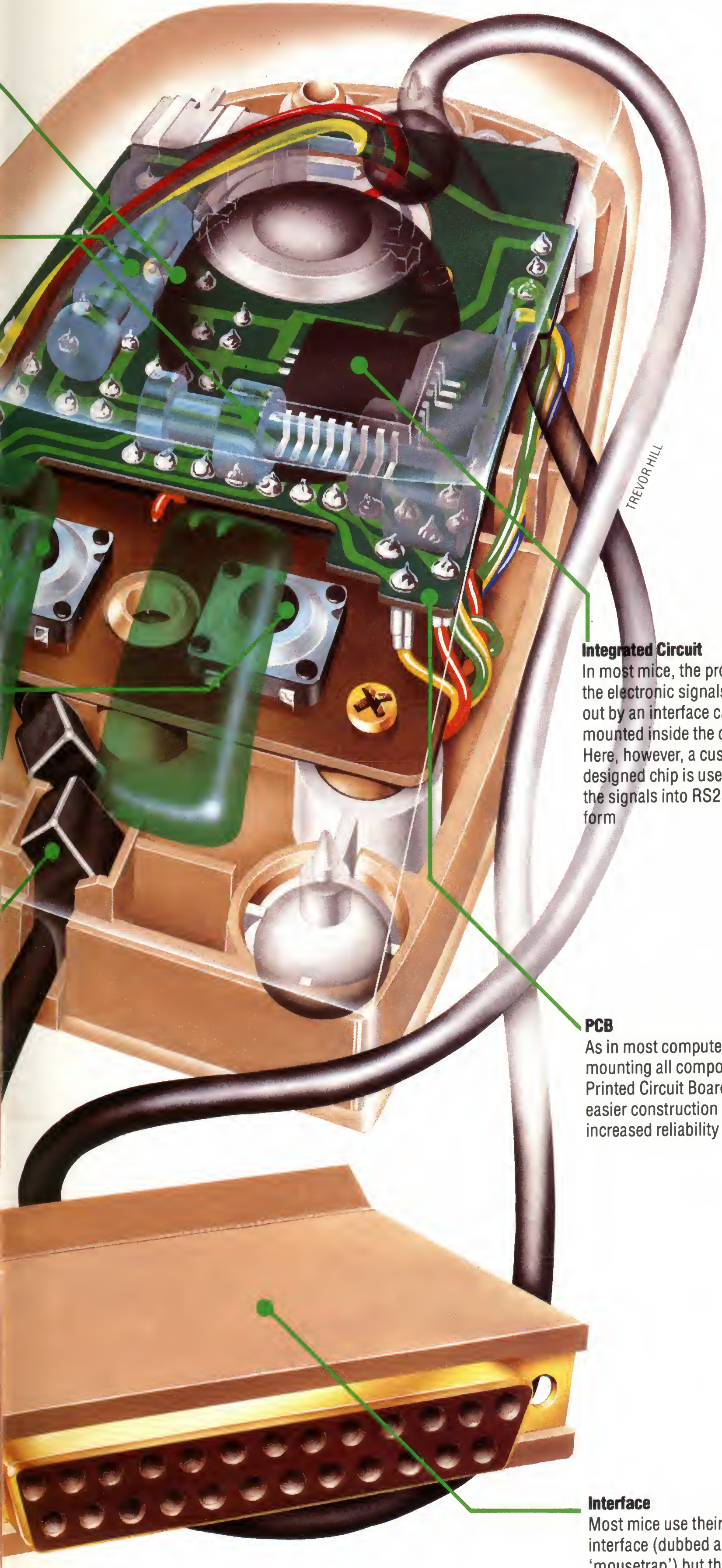
Main Ball
A large steel ball-bearing rests on the surface across which the mouse is moved. On some mice the ball is made from hard rubber to prevent it from slipping

Encoding Wheels
These two wheels make constant contact with the ball to pick up its movement in two directions. The wheels are mounted on shafts; at the end of these shafts are encoding devices that produce electrical pulses as the shafts are turned

Buttons
The function of the two buttons will depend on the software package in use. Usually, one is used to select an item, and the other to move objects around the screen

Microswitches
These are mounted on the PCB beneath the buttons, and require only a tiny movement to make or break the circuit

Rubber Grommet
The mouse must be free to be moved around the desk, and the rubber grommet is particularly important in preventing strain on the connection between the cable and PCB



TREVOR HILL

Integrated Circuit
In most mice, the processing of the electronic signals is carried out by an interface card mounted inside the computer. Here, however, a custom-designed chip is used to convert the signals into RS232 (serial) form

PCB
As in most computer devices, mounting all components on a Printed Circuit Board makes for easier construction and increased reliability

Interface
Most mice use their own special interface (dubbed a 'mousetrap') but this one can plug into any RS232 port, using the standard 25-way connector

the other system, two slotted discs are fitted to the rollers. A light is continuously directed at the discs and the beam is detected optically on the other side of them by a photocell. The pulses of light passing through the slots are then converted to electrical signals, which are treated in the same way as those of the mechanical system.

There are other systems, as well. In one case, for example, the mouse is used in conjunction with a special pad covered with a pattern of dots. A light inside the mouse's body illuminates the area of the pad covered by the mouse and this pattern is detected by a special optical processing chip. Any movement of the mouse will change the pattern that the chip detects and it can instantly calculate how far the device has moved in any direction. This system has the advantage of having no moving parts, but it is much more expensive than the others.

Once the cursor has been moved to the required place on the screen its position can be entered into the computer by pressing one of the 'ears' (buttons) on the mouse. The number of buttons fitted varies from one manufacturer to another. Some systems use as many as three; Microsoft have chosen to fit two, while the Apple Lisa mouse has only one. The buttons can also be used to select items from a menu — programs such as Microsoft's MultiTool Word use this facility — and give the mouse control of the normal cursor motion. These devices can be used with highly sophisticated software such as that provided on the Apple Lisa. Here the button is pressed once to select an 'icon' (see page 262) from a screen menu, and twice to open out that particular application.

The main advantage of all mice, and the software that has been produced to complement them, is that they can be used by those who have no keyboard skills. Rather than having to type in the name of a program or press certain letters or numbers to select a function, the user simply moves the mouse so that the screen cursor points to the application or course of action that is required, and presses a button to activate it.

Unfortunately, the mouse doesn't completely eliminate the need for a keyboard — new text and numbers still have to be fed into the computer — but it does make the manipulation of that information much simpler. Tests conducted by Apple during the development of the Lisa showed that a user entirely unfamiliar with a computer can learn to work with the Lisa's mouse-driven software in as little as 15 minutes. Similar software running on a conventional system takes nearly 20 hours to become familiar with, mainly because of the problems involved in learning to use the keyboard, and the need to learn lengthy and complicated commands. Electronic mice will soon be an integral part of home computers. They are efficient and simple to use and they don't frighten the faint-hearted as much as the sight of a traditional QWERTY keyboard.

Detective Work

When data is passed from one computer to another it runs the risk of becoming corrupted. Hamming codes can detect and correct these errors

We must all have heard stories about computers making dreadful mistakes — like mailing 500 copies of the same company leaflet to one person. The truth is, of course, that the machine is not to blame: the mistake will have originated from a human failing, perhaps as simple as a typing error. The computer merely serves to amplify the problem. Occasionally, errors arise because the applications program hasn't been written to cope with all eventualities — as in the case of computer-generated final demands for gas bills of £0.00.

Sometimes, though, computers make mistakes that can't be attributed to human intervention, and these are usually manifested in the form of 'bit errors'. A bit error occurs when a single bit in a section of data is transposed from a 1 to a 0 or vice-versa. Bit errors can be caused when a hardware component, such as a RAM chip, fails. That's why many home computers go through a 'diagnostic' error checking software routine whenever the power is turned on.

Most bit errors, however, are 'soft errors' — bits get 'flipped' even though all the RAM has passed the diagnostic test. Home computers are designed to operate in domestic environments, but during a summer heatwave it is quite possible for the temperature to exceed the operating temperature range of the components. Damage is unlikely to be permanent, but bit errors may result in a character on the screen suddenly changing from an 'A' to a 'B', for example, or if the bit happens to form part of an important pointer, it may 'crash' the program, requiring the computer to be reset.

Bit errors can also arise during periods of high sunspot activity, when sub-atomic particles can penetrate the atmosphere and interfere with the flow of electrons in a miniature circuit. In applications such as military systems, industrial control, scientific experimentation or international banking, errors could bring disastrous consequences, so a variety of methods have been adopted to detect them.

The simplest is parity checking (see page 253). An alternative method is the 'checksum', which is widely used when writing data onto magnetic tape or disk. Data is typically handled in blocks of 128 bytes, the last of which to be read or written will be a checksum byte. This byte represents the sum of all the other bytes (each having a value in the range 0 to 255) modulo 256 — meaning the remainder of the sum when divided by 256. Here's an example:

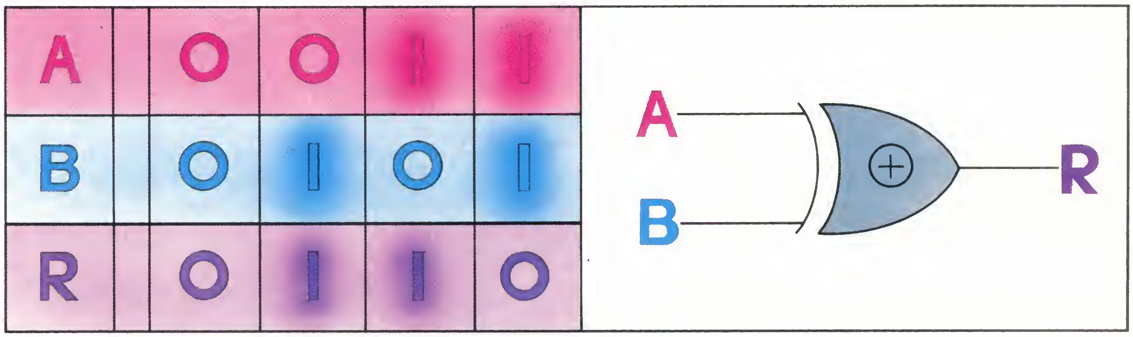
Data: 114,67,83 . . . (121 other values) . . . 36,154,198
Total of these 127 bytes = 16,673
Total divided by 256 = 65, remainder 33
Therefore checksum = 33

The total of the bytes (16,673) is equal to 65 lots of 256 plus a remainder of 33 — the value that is written into the 128th byte as a checksum. When the computer reads the block back again, it performs its own checksum calculation on the data and if this value differs from 33 then it knows that a bit error has occurred in the recording process.

With both parity and checksum, the computer has no way of knowing which bit of the data has been corrupted. If the error occurred in transmission, then the receiving computer can request a particular byte or block of bytes to be transmitted again; in the case of a recording error, there may well be no way of retrieving the uncorrupted data.

Where errors would be unacceptable, a system must be used that will both detect and correct them. Hamming codes, named after their inventor R W Hamming of Bell Telephone Laboratories, perform this function.

All error correction systems work on the principle of redundancy. Human languages contain a high degree of redundancy — if a typing



A	0	0	1	1
B	0	1	0	1
R	0	1	1	0

Exclusive-Or
A simple Exclusive-Or gate has two inputs and one output. If both inputs are at logical 0 then the output is 0. If either input is 1 then the output is 1. However, if both inputs are 1 then the output is 0. This last condition differentiates the Or gate from the Ex-Or (for short). The operation can be represented by a truth table. Where an Ex-Or has more than two inputs, the output will be 1 if there is an odd number of 1s at the input. Such devices are the means by which parity and error-checking bits are created

KEVIN JONES

error occurs in a manuscript, or a crackle obliterates words in a telephone conversation, it is often possible to recreate the words by considering the context of the sentence. Sometimes we build in extra redundancy for use in 'noisy' environments: the use of 'alpha', 'bravo', and 'charlie' in place of 'a', 'b', and 'c' in radiotelephony, for example.

Suppose that on our computer we send a word of x bits in length, consisting of y bits of real data and z redundant bits (i.e. $x = y + z$). In our explanation of parity we had a value of seven for y and one for z. For Hamming codes, z will need to be proportionately larger. Now let's assume that a single-bit error can occur in any of the x bits (our z redundant bits are of course just as prone to error as the y data bits). If the chance of a bit error in a word is, say, one in a million, then the chance of two errors in a word is one in a million million, so we'll ignore this possibility.

When the data is received at the other end, there will be x+1 eventualities. Either there will be no errors, or the first data bit will be in error, and so on up to the xth bit. Now, with z redundant bits we can represent 2^z situations, so that for the word to be proof against one bit error:

$$2^z \geqslant y+z+1$$

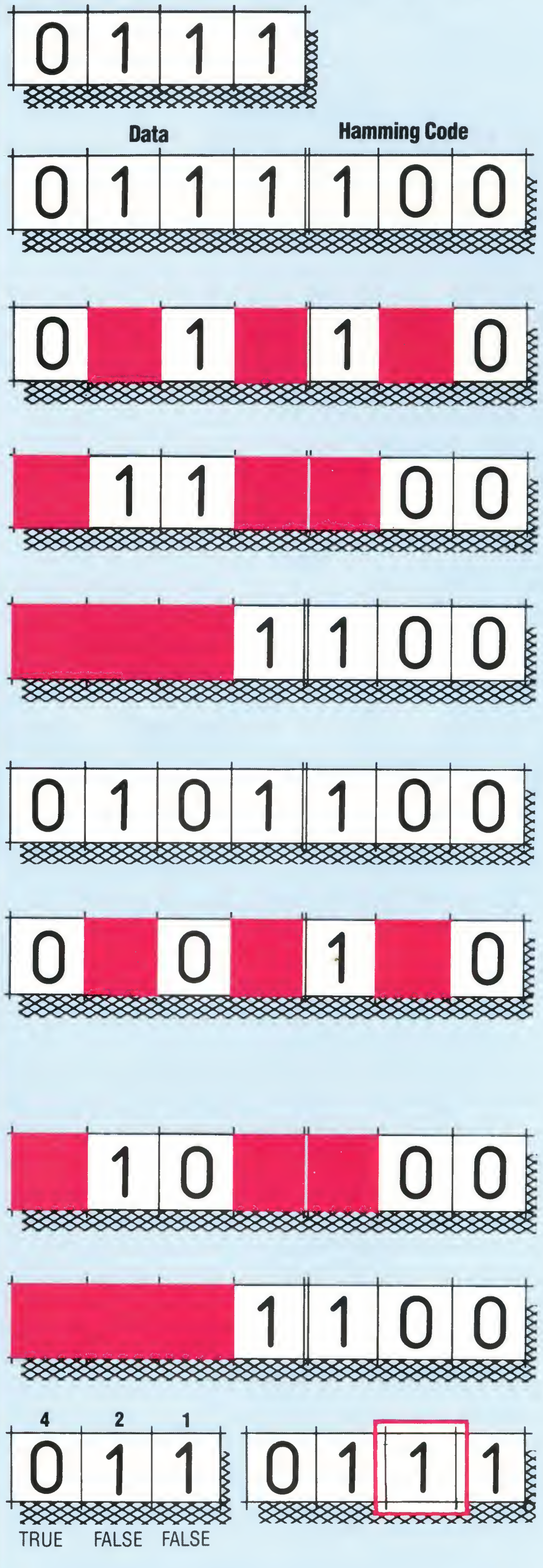
If y is seven (for ASCII codes), then z will need to be four. If y is four (as in our example in the panel) then z will need to be three. However, if y is 16 then z need be increased only to 5. It follows that Hamming codes are far more efficient for longer word-lengths than for short ones.

In a Hamming code, each of the redundant bits acts as an even-parity check on a different combination of bits in the word. If any bit is flipped in transmission then one or more of the check bits will be wrong and the combination of these bits will point to the erroneous bit in the word (see example). The receiving computer's software can then simply flip that bit back again.

The key to the way that Hamming codes work is the different combinations of bits upon which each Hamming bit acts as a parity check. The total number of bits is effectively divided into several different but overlapping sets — devised so that no two bits appear in the same combination of sets. The receiving computer performs parity checks on the same sets as the sending device did to create the Hamming code. If any one of the bits, including the Hamming bits, has been flipped in transmission, then one or more of these sets will not pass the parity test. The combination of tests failed points to a unique bit.

Some computers employ Hamming codes even for their internal memory operations. When this is the case, it is possible to remove one whole RAM chip and watch the computer continue to function! Some military computers take the principle of redundancy to the extreme of duplicating every single component in the computer, and comparing the results from the two halves after each operation.

How A Hamming Code Works



Suppose we wish to send these four bits of data

To them we must add a three bit Hamming code, a unique pattern of bits generated by the computer to fulfill the following conditions:

Looking at just these four of the seven, there must be an even number of 1s visible

Similarly, out of these four there must be an even number of 1s

And in this set of bits, there must be an even number of 1s, too. Working out the three bits that will fit these conditions requires the computer to solve three simultaneous equations

But let's suppose that during transmission, the third bit from the left is corrupted, i.e. is flipped from 1 to 0

If the receiving computer performs the first of the three tests on the data, it now fails because there is an odd number of 1s visible. This tells us that there has been an error, but we still don't know which bit was affected

Similarly, the second test produces a false result

However, the data still passes the third test — an even number of 1s is visible

It is the combination of tests passed and failed that indicates the bit in error. If we express a failed test as a 1 and a passed test as a 0, then writing the results in reverse order, we get the binary for three — indicating that the third bit was corrupted, and should be flipped back from 0 to 1

This principle will still work even if it is one of the Hamming bits that gets corrupted. If all three tests fail, for example, 111 would indicate that the rightmost bit was corrupt, whereas if all three pass, there has been no error. This type of correcting code fails only if there is more than one error in the seven bits

Norbert Wiener

The child prodigy whose study of mathematics resulted in the birth of the science of cybernetics

Norbert Wiener was born in 1894 in Missouri, USA. After taking a degree in mathematics at the age of 14 and receiving a doctorate in logic at 18, he went to study with David Hilbert at Göttingen, Germany.

Wiener's contribution to computer science came late in his life. For many years he worked at the Massachusetts Institute of Technology, studying the new probabilistic physics, and concentrating on the statistical study of the motion of particles in a liquid (a phenomenon known as Brownian movement). The particle movements were so unpredictable that it was impossible to describe them using the traditional physics of deterministic forces. So a 'probabilistic' method, by which only the *probable* location of a particular particle at a given time could be predicted, was the best that could be applied.

When the Second World War broke out he offered his services to the US government and began work on the mathematical problems implicit in aiming a gun at a moving target. The development of automatic gunsight guidance systems, his studies in probabilistic physics and his broader interest in subjects ranging from philosophy to neurology all came together in 1948 when he published a book entitled *Cybernetics.*

Cybernetics is the study of the self-governing controls that are found in stable systems, be they mechanical, electrical or biological. It was Wiener who saw that information as a quantity was as important as energy or matter: copper wire, for example, can be studied for the energy it can transmit or the information it can communicate. The revolution that the computer promises is based in part on this idea: a shift in the source of power from the ownership of land, industry or business to the control of information. His contribution to computer science was not a piece of hardware but the creation of an intellectual environment in which computers and automata could be developed.

The word 'cybernetics' is derived from a Latin word meaning 'governor'. Wiener had studied the 'governor' of James Watt's steam engine, which automatically regulated the machine's speed, and he realised that for computers to develop they must be made to imitate the ability of human beings to regulate their own activities.

The thermostat in a house is an example of a control system. It regulates the heating according to fluctuations in temperature above or below an optimum level. A human is needed only to set this level. Wiener called this faculty for self-regulation and control 'negative feedback' — 'feedback' because the output of the system (the heat) affects the future behaviour of the system and 'negative' because the changes the thermostat brings about are made to restore the temperature to the one set.

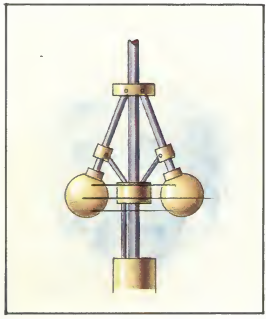
A system that can do this and also choose its own temperature (and other goals) is called a 'positive feedback' system. When an automaton can do all this and reproduce itself as well, then it approaches the human condition.

Wiener's theory of cybernetics could be regarded as a super science — a science of sciences — and it has encouraged research into many areas of control systems and systems that deal with information. Everything is information. What we know about the changes in the world comes to us through our eyes and ears and other sensory receivers, which are devices for selecting only certain data from a totality that would otherwise engulf us.

Information can also be studied in a statistical way, independent of any meaning it may have. For example, by observing the frequency with which certain symbols occur it is possible to break many types of codes. In the English language the letter 'e' occurs most often, and the letter 't' is the second most frequently used. By analysing large samples of a code and comparing the results with typical samples of English, it is possible to identify key letters and thus begin deciphering the code.

Wiener died in 1964, before the microcomputer revolution began, yet he foresaw and wrote about many of the problems that would arise in this new technology.

Speed Restriction
Wiener was fascinated by the idea of the steam governor — one of the best and simplest examples of negative feedback. Two weights are connected by pivoting arms to a spinning shaft, which is in turn connected to the flywheel of the steam engine. As the speed of the engine increases, the weights will fly outward. This movement, by means of a suitable linkage, shuts off the throttle of the engine slightly. This has the effect of stabilising the speed of the engine at any level set by the operator. Modern computers can implement far more sophisticated types of control, but the principle is still the same



KEVIN JONES

Mentathlete

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

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

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

The Spectrum does just that.

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

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

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

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

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

sinclair

THE HOME COMPUTER COURSE BINDER

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

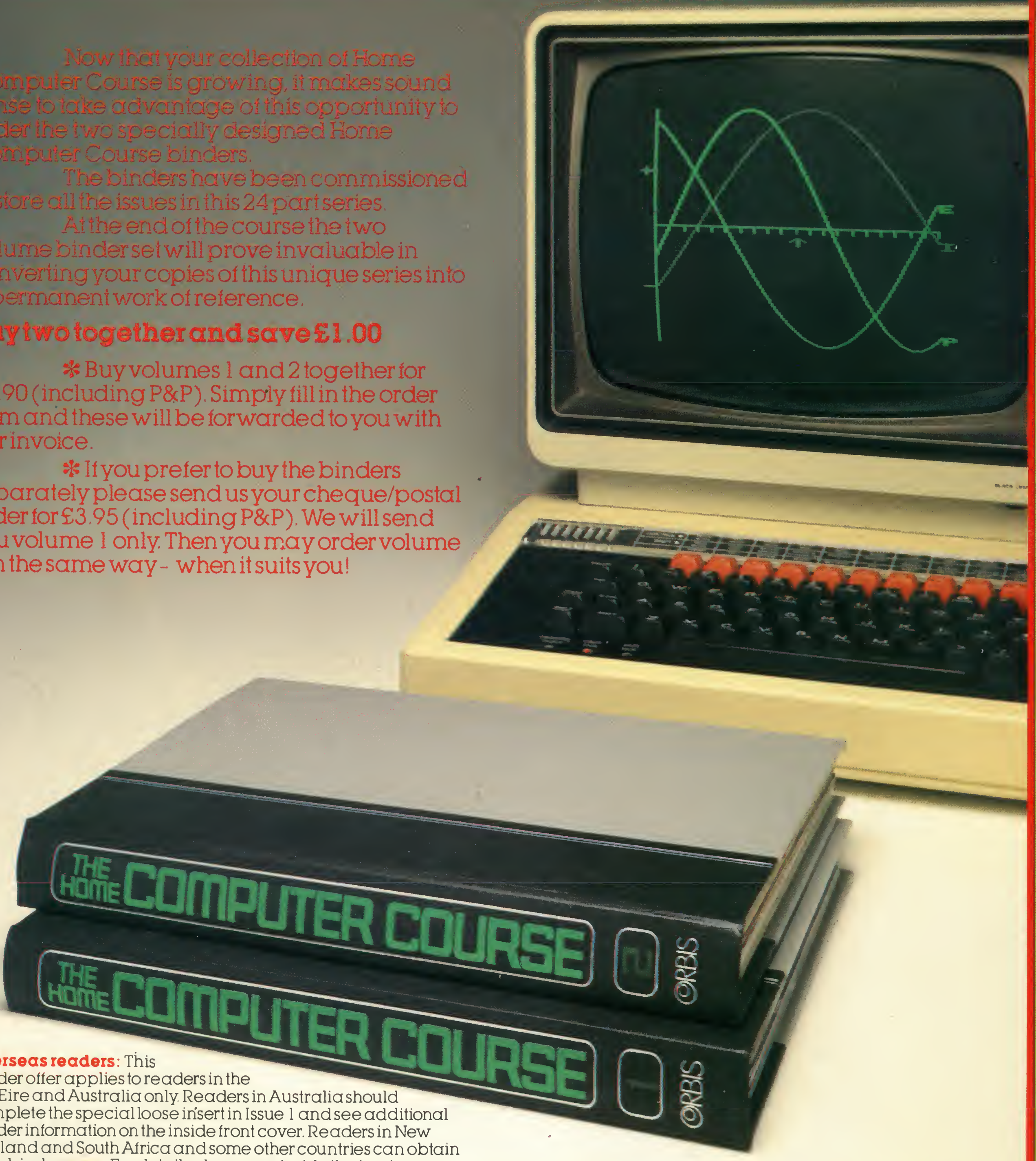
The binders have been commissioned to store all the issues in this 24 part series.

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

Buy two together and save £1.00

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

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



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

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

NEXT TO YOUR COMPUTER...YOUR COURSE MANUALS