ISSN 0265-2919
R 80p

# THE HOME COMPUTER COURSE

18

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

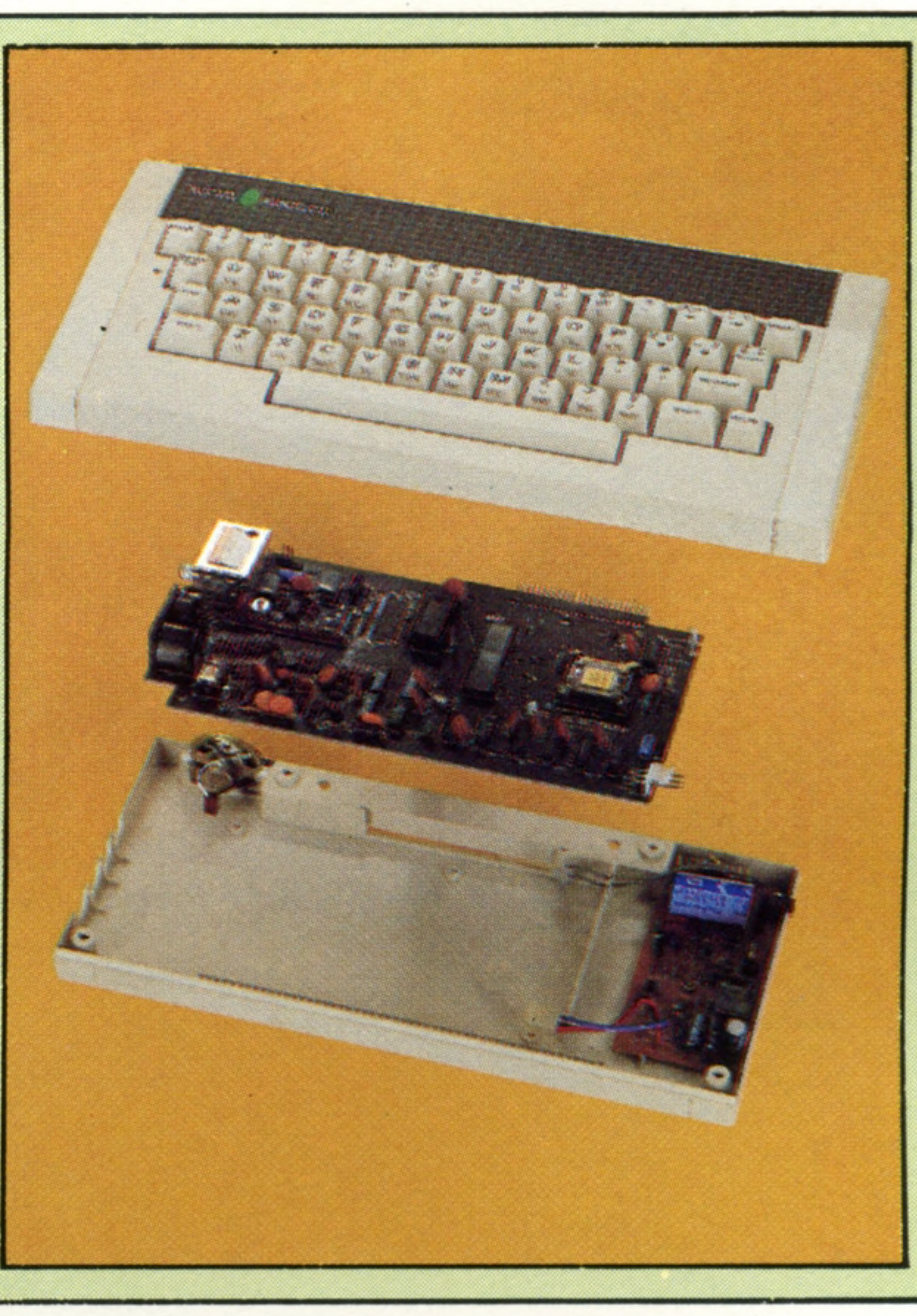
An ORBIS Publication

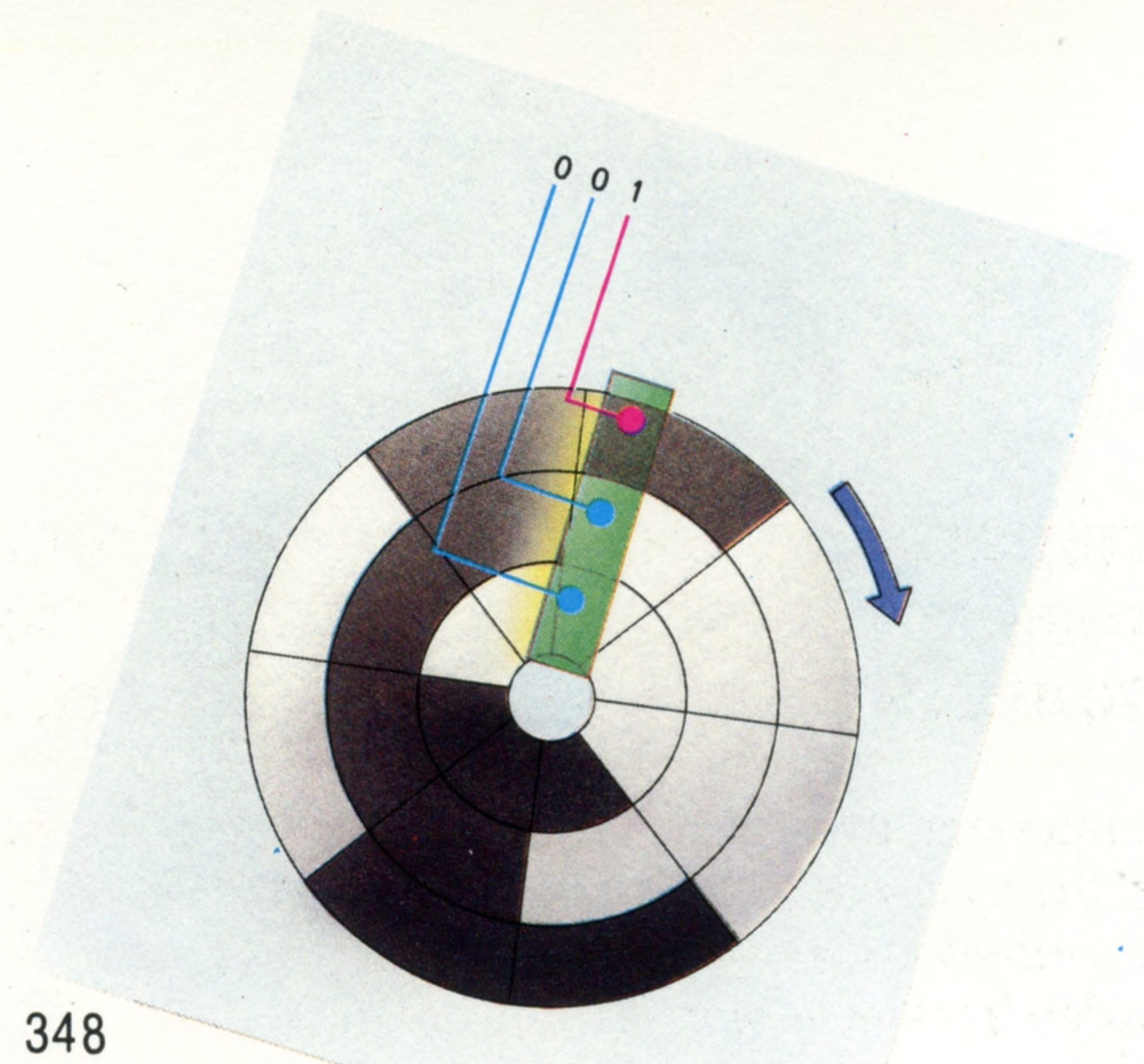
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

## CONTENTS

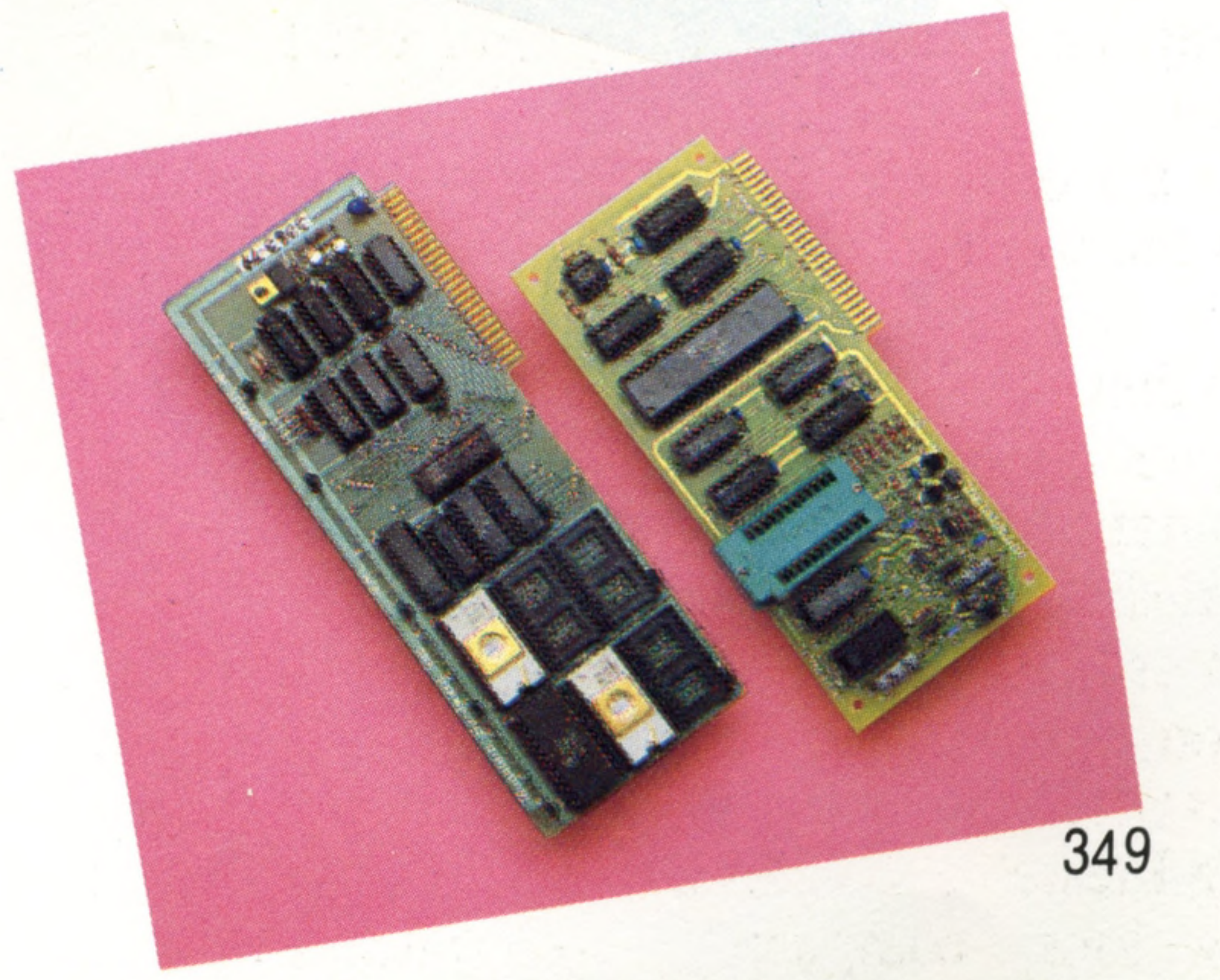
### Next Week

- The Acorn Electron is similar in many respects to the BBC Model B. We take a close look
- Solitaire and Patience used to be the most popular solo games, but since the development of the microcomputer you can play chess with a Grand Master and backgammon with an expert — any time you like
- Educational simulation software turns the computer into a science laboratory or even a whole expedition into the field. We look at programs for the BBC Micro and the Spectrum





348



349

**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; **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.

# Going Places

COURTSEY OF YRM ARCHITECTS & PLANNERS

**People Mover**
In common with many airports in the United States, Gatwick Airport to the south of London has installed a revolutionary inter-terminal transportation system that combines the directional stability and automatic driverless operation ability of the light railway with the comfort and convenience of the bus or coach. Designed and built by the Westinghouse Corporation, famous for its rail signalling systems, the People Mover can carry up to 100 passengers at a time

**Transporting goods and people from place to place in this increasingly crowded world is a difficult business. The use of computers helps matters considerably**

A century and a half ago it took three months to travel from Europe to Australia. In the 1980's that same journey could be accomplished in less than half a day. This technological miracle would not be possible, however, without extremely sophisticated methods of computerised vehicle control.

Air travel is the form of transportation that poses the most demanding problems. Heathrow Airport in London, for example, handles more than a thousand flights every day, with 120 aircraft movements an hour at peak periods. Without computer assistance in controlling this activity, the system simply could not operate.

Let us take the case of a traveller who wishes to go from London to New York. From the travel agents where a ticket is bought and a seat reserved (using Prestel as a gateway into the airline's own reservations computer), to touchdown at Kennedy Airport, perhaps as many as 15 different computers will have been directly concerned with the journey. Let's examine computer involvement in air travel in more detail.

The first consideration must be the standard of the aircraft itself. Modern passenger aircraft cost a great deal of money. In order to maximise the return on their investment, operators must keep the aircraft in 'as new' a condition as possible, and the key to this is 'planned maintenance'. After a predetermined number of flying hours the aircraft will return to its engineering base, where personnel will have access to computerised records of the aircraft's complete history from the first day of its construction, down to the serial number of every part, both past and present. The records will detail: every engineering operation the plane has undergone; reports on its performance from flight engineers and other aircrew members; fuel consumption figures; and any other piece of data that could conceivably be of interest. The home computer user might consider applying these same methods — though not in such detail, perhaps — to the maintenance of a car.

Only when the maintenance schedule is complete and up to date will the aircraft be returned to service. It will then become a component of yet another computer system — the airline's operational control system. This allocates aircraft to routes; places orders for fuel at various points along those routes; and arranges for crew, meals, in-flight entertainment and the multitude of other arrangements necessary to transport three or four hundred people halfway around the world.

Another computerised control system operates within an airport itself, where officials must cope with the huge demand for their often limited facilities from airlines to whom even a few minutes' delay might mean a considerable loss of money. It is only by means of computerised scheduling that this operation can be successfully accomplished. The airport's computer system will also take care of calling passengers to check in for flights and displaying arrival and departure information, for instance.

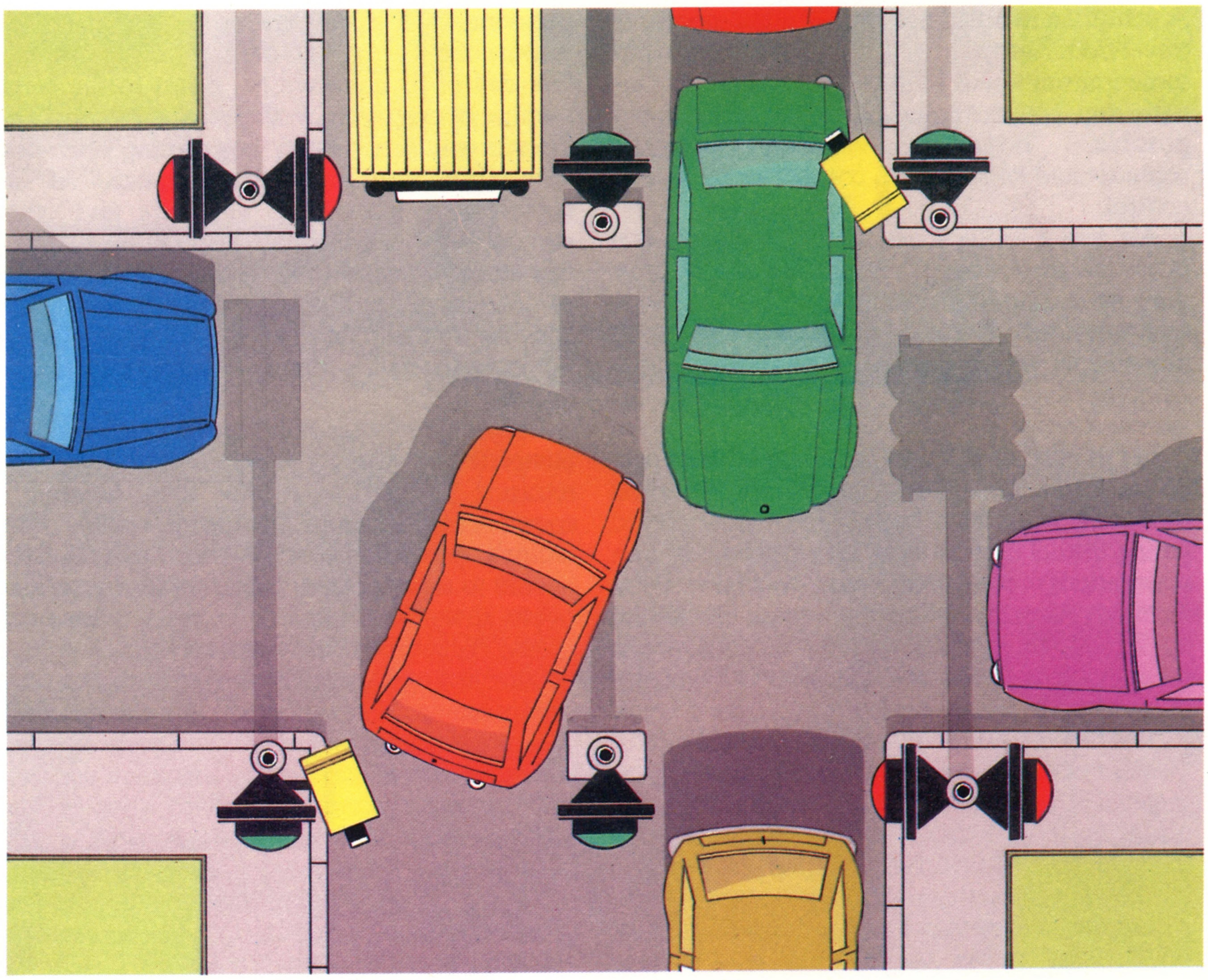
Before the passengers have even arrived at the airport the pilot will have filed a detailed flight plan with Air Traffic Control. Surprisingly, the ATC's task is only partially computer-aided. Thanks to radar transponder systems, controllers no longer have to rely on the pilot to communicate his position verbally. They see each blip on their radar screens identified by a flight number, along with a computer-interpreted altitude reading and destination code transmitted automatically from the aircraft.

A further degree of computer assistance is available to the controller in the form of printed slips, each of which covers a segment of the planned route, derived from the pilot's flight plan. These slips, which include information about the flight path, height, payload and aircraft type, help the controller to guide the flight through his area in the quickest, most economical way.

Another area of transportation in which computers are widely used is rail transport. The railway signalman's job, although not as complex as the air traffic controller's, has some things in common with it. He too is employed to guide freight and passenger traffic through his area, safely and at the lowest possible cost. British Rail has been using computer control systems since the mid-1970's, following pioneering work undertaken in the United States by the Southern Pacific Railroad. Its Total Operations Processing System (TOPS) performs all aspects of freight control, from maintaining an accurate, up-to-the-minute inventory of each locomotive and piece of rolling stock, to assembling these units into complete trains and determining their routes.

Each goods wagon has a unique identifying number, which is recorded along with its location by the TOPS computer. When a wagon of that type is needed at some other location, the wagon is assigned to a particular train and its destination noted. When it arrives there its new status is noted by the TOPS, which can then re-assign it. Considering the volume of freight traffic that British Rail runs (there were 185,000 wagons in service at the end of the 1970's, making up almost 2,000 trains a day), such computer control systems are clearly imperative.

Many of the problems of airport operation are also shared by railway stations, and many of the same solutions have been employed. However, railways have a further factor to take into account — the increasing use of unmanned stations. There have been a number of experiments conducted using microcomputers to answer travellers' enquiries — and in some cases to make train arrival and departure announcements using synthesised speech. Computerisation also makes it possible to contemplate unmanned trains. London Transport's underground Victoria Line, for example, has this capability, although it is not used in practice because driverless trains running under



**Green For Go**
The phasing of traffic signals has been employed by highway engineers for some time in an attempt to regulate the flow of vehicles — especially through main city streets. Nowadays, individual traffic lights can monitor the density of traffic in their immediate vicinity by means of doppler radar detectors and pass this data to a computer system. The frequency of the changing of the lights can then be adjusted to suit prevailing conditions

COURTESY OF BRITISH AIRPORTS AUTHORITY

**Announcing The Departure . . .** Perhaps the most public aspect of the airport operating computer system is the arrival/departure board. These electromechanical devices are constantly updated by the central computer as new information comes to hand

computer control are regarded with suspicion by the public.

A very close parallel to a driverless train system is found in the more sophisticated model railway networks. Each model locomotive has a unique identifier code, called a 'device number', stored in a small microcomputer based on a single chip. The controller sends out signals to each train by modulating the power conducted along the rails in such a way that only one locomotive will be able to decode it. In this way a large number of trains can run on the same layout at any one time under the direct control of the central microcomputer.

The driverless train is feasible because it runs on rails, but the concept is unlikely to be used by road vehicles. Computers are, however, used in road transport, primarily in public transport and freight operations. Here they help schedule and route vehicles, as well as produce timetables — which is an incredibly complex task in a city the size of London, where there is a need to integrate bus, train and underground services into one public transport system.

The problem is to keep just enough vehicles in operation so that passengers will not experience unacceptable delays, while at the same time not use so many as to dissipate operating profits. This is a complex problem in statistics and it was to solve statistical problems that computers were first devised. Another problem that benefits from the application of statistical methods is the routing of delivery vans to minimise the distance covered between 'drops' and the allocation of consignments to each van. An interesting variant of this is to be seen in vehicle despatching — especially of taxi cabs and police cars, which 'cruise' rather than return to base. Each vehicle's location is entered as a street name, which the computer then converts into a grid reference. A request for a taxi or emergency assistance is entered in the same way, and the task of matching resources to requirements is a simple matter of comparing the two according to predetermined rules.

One of the more interesting experiments involving computers in public transport is the 'dial-a-bus' system, now operating in the suburbs of Hanover, in Germany. Based on a fleet of minibuses that follow no set route, the system allows a passenger to call up the central control station from the bus stop, and tell them his destination. The miniterminal (it looks rather like a cash dispenser terminal) then prints out the time at which the bus will arrive (never more than five minutes), the duration of the journey (taking into account the needs of the other passengers) and the fare.

Transportation of goods and passengers accounts for some 20 per cent of all the world's commerce. The use of computers in this area is more advanced than in others, and has undoubtedly contributed significantly to its growth. While most of the examples we have looked at are implemented on mini- or mainframe computers, many of them are applicable to home microcomputers as well. There are a number of software packages available for distribution, scheduling, timetabling and the like, which can have a remarkable variety of applications.

**Factory Ships**
The shipping industry uses computers far less than other transportation media, but one important applications area is that of containerisation services. Microcomputers play an important part in consolidation (a number of shippers sharing a container), in the organisation and layout of the container terminal and in the loading of the container ship to ensure even distribution of weight

COURTESY OF "THE MOTOR SHIP"

# Speaking In Tongues

**Basic has a familiar mathematical construction, and so is relatively easy to learn, but it is clumsy in relation to some of the other languages**

Unless your home computer is a Jupiter Ace (see page 150) then it will almost certainly feature BASIC as its resident programming language. But that doesn't mean to say that you are restricted to that choice, and though BASIC is acknowledged as being a particularly easy language to learn, there are other far more suitable languages for writing specific applications. To install these on your computer it will be necessary either to replace the ROMs containing the BASIC interpreter, or to load the new language into RAM — in which case you will need a machine with a reasonable memory capacity so that there is RAM left over to contain your programs. A few home computers, such as the Sharp MZ-711, have anticipated this problem by also having the BASIC interpreter cassette loaded.

## PASCAL-ENGLISH ENGLISH-PASCAL

PASCAL was developed in the early 1970's as the successor to BASIC. Its range of data and control structures derive from the FORTRAN/ALGOL group of languages and these are intended to encourage the student to approach programming in a systematic way, and to write well-structured, easily understood code. This is very desirable for developing good programming technique, but it does mean that the early stages of learning programming are harder for the complete beginner — in part because of the discipline that the language imposes, and also because it is usually compiled rather than interpreted. Nevertheless, PASCAL programs tend to be elegant, relatively quickly developed, and much easier to understand.

Here is the PASCAL equivalent of the BASIC program:

```
VAR
NAME        : PACKED
              ARRAY (1..30)
              OF CHAR;

AGE,COUNT : INTEGER;
ANSWER      : PACKED
              ARRAY(1..3)
              OF CHAR;

RUNNING     : BOOLEAN;

BEGIN
RUNNING:=TRUE;
WHILE RUNNING DO
   BEGIN
   WRITE('What is your name?');
   READLN(NAME);
   WRITE('and how old are you?');
   READLN(AGE);
   FOR COUNT:=1 TO AGE DO
      WRITE(COUNT:3, 'Hello':10, NAME);
   WRITE('Want another go?');
   READLN(ANSWER);
   IF ANSWER(1)='N'
   THEN RUNNING:=FALSE;
   END;
WRITELN('Goodbye', Name)
END.
```

## BASIC-ENGLISH ENGLISH-BASIC

A much-maligned language, BASIC was developed out of FORTRAN (which was one of the first high-level programming languages, and is still the most popular for scientific and engineering applications) as a tutorial introduction to programming, for university students. Because it was intended for self-tutorial use, BASIC is usually interpreted rather than compiled (see page 184), and this factor has been the main reason why it has become the native language of almost all home computers. Interpreted languages are cheap to implement, use comparatively little computer memory, and are extremely suitable for program development.

BASIC is now a powerful language in its own right, but is hindered by its variety of non-standard dialects (every computer's BASIC is unique to that machine), and its lack of specialised data and control structures. These limitations mean that self-taught programmers can develop bad programming technique, and that good programming technique can be difficult to apply to specific problems in BASIC.

This short program illustrates the appeal and generality, but also the limitations, of BASIC:

```
100 INPUT"What's your name?";N$
200 INPUT"and how old are you?";A
300 FOR K=1 TO A
400 PRINT K,"Hello ";N$
500 NEXT K
600 INPUT"Want another go?";Y$
700 IF LEFT$(Y$,1)="Y" THEN GOTO 100
800 PRINT"Goodbye ";N$
900 END
```

## COMAL-ENGLISH ENGLISH-COMAL

COMAL was developed to combine the accessibility of BASIC with the powerful structures and disciplined approach of PASCAL. It therefore resembles both, and may have been a model for the development of the BBC Micro's BASIC, which has almost developed into a new language. COMAL has been very successful in school computing and in Scandinavia (where it originated), but seems unlikely to displace either of its two forebears as the introductory programming language.

This is the "Hello" Program in COMAL:

```
100  RUNNING:=TRUE
200  WHILE RUNNING DO
300     INPUT"What's your name? ";N$
400     INPUT"and how old are you? ";A
500     REPEAT
600        FOR K:=1 TO A DO
700           PRINT K,"Hello";N$
800        NEXT K
900     INPUT"Want another go ";Y$
1000    IF Y$="N" THEN RUNNING:= FALSE
1100 ENDWHILE
1200 PRINT"Goodbye ";N$
```

## LISP-ENGLISH

## ENGLISH-LISP

LISP was developed in the early Sixties as a list processing language, and has since been widely used in the field of Artificial Intelligence, which involves continually searching and comparing lists of data, relationships and responses. Unlike BASIC, where the emphasis is on the flow of the program through a sequence of instructions and procedures, LISP is a 'functional' language, in which the elementary command set can be built up to form more sophisticated functions with names determined by the programmer. For example:

```
(SETQ ARRAY1 '(4 7 2 5 1))
```

creates a list called ARRAY1 whose elements are the numbers (4 7 2 5 1).

```
(CAR ARRAY1)
```

gives the first element of the list ARRAY1 (4, in this case).

```
(CDR ARRAY1)
```

gives the list ARRAY1 with its first element removed — (7 2 5 1) in this case.

```
(SETQ ARRAY1 (CDR ARRAY1))
```

will turn the list ARRAY1 into a copy of itself excluding its first element.

LISP also lends itself to 'recursive' applications — problems that after a simple function is applied are reduced to a smaller but identical problem.

## FORTH-ENGLISH

## ENGLISH-FORTH

FORTH resembles LOGO in being a functional interactive language, but has the important distinction of being the first language other than BASIC to be implemented as native on a home computer — the Jupiter Ace. The language consists of a number of defined functions, called the 'primitives', and has the ability to define new functions in terms of these. Mathematical operations in FORTH are 'stack-oriented', which means that computer memory is treated as an expanding and contracting list of data and this results in the last operation always being at the head of the list. A further consequence of stack-orientation is that algebraic notation is not used. Instead of writing (12 + 4)/2 to find the mean of 12 and 4, in FORTH you must write 12 4 + 2 /, which is the same sum in Reverse Polish rather than algebraic notation.

All this makes FORTH a very different kind of language, forcing a very different view of problem-solving and computer processes. It's almost a step back down the high-level languages hierarchy.

This FORTH fragment defines two new words called SHOUT and CHORUS:

```
: SHOUT (prints "SHAZAM !")
  ."SHAZAM !";
: CHORUS (uses SHOUT in a loop)
  0 DO SHOUT LOOP;
```

Now typing n CHORUS will cause SHAZAM ! to be printed n times on the screen.

## LOGO-ENGLISH

## ENGLISH-LOGO

LOGO was developed by a psychologist working on Artificial Intelligence in the context of the classroom. It resembles FORTH in both its interactivity and its use of a number of 'primitives' that can be incorporated in user-defined functions. But the fundamental principle it embodies is that the way to learn something is to teach somebody else — namely the computer — how to do it. It is considered an innovative language that will create a completely new way of teaching children to think.

LOGO is usually called a 'turtle' language because it is often used to control a small wheeled robot called a turtle (see page 34).

Here is a LOGO fragment that draws a symbolic house as a square of specified size with a triangle on top:

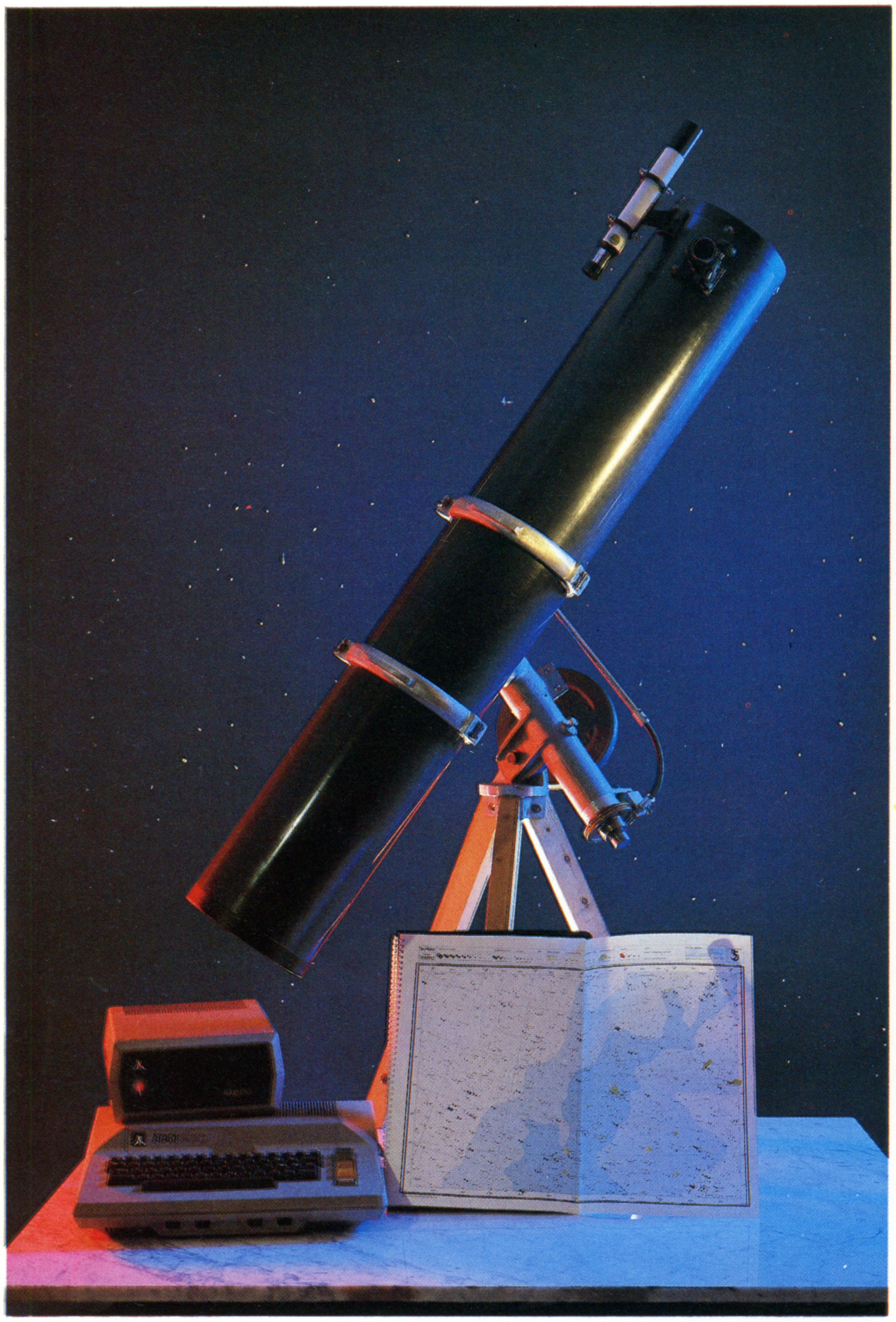
```
TO TRIANGLE LENGTH
   REPEAT 3(FORWARD:LENGTH RIGHT 120)
END
TO SQUARE LENGTH
   REPEAT 4 (FORWARD:LENGTH RIGHT 90)
END
TO HOUSE LENGTH
   RIGHT 30
   TRIANGLE :LENGTH
   LEFT 90
   SQUARE :LENGTH
END
```

Now typing HOUSE 15 will cause a 'house' with a side length of 15 units to be drawn.

On these pages, we give you an overview of the most common programming languages available for home computers. As with human languages, the more programming languages you've mastered, the easier it is to adopt a new one.

# Shooting Stars

**Computers have two main uses in the field of astronomy: maintaining a database of observed objects, and calculating their current position to aid accurate telescope alignment**


MARCUS WILSON-SMITH

**Guiding Light**
Optical astronomy is made considerably easier when an accurate prediction can be made of the location of a given star or planet on a given day. A home computer can easily contain the database of star locations, and can quickly perform the necessary calculations. With the addition of stepping servo-motors it can even position the telescope

When you look up at the night sky and locate the nearest star, you are actually seeing that star as it was four years ago, because that is how long it takes for the light to reach us from Proxima Centauri. In those same four years the microcomputer has advanced from a rare and expensive novelty to a common peripheral in the British household. Astronomy is an application that utilises to the full a home computer's potential for data handling, computing and robotics.

There are three main types of problem that an astronomer faces when studying the night sky: the initial observation of the celestial object, the handling of data obtained through observation, and the meaningful analysis of this data. In each of these areas the computer is a useful aid.

Let's begin by looking at how a home computer can help in making the calculations needed to construct the astronomer's basic tool — the telescope. The actual arrangement and design of the lenses and mirrors in a telescope is critical to the quality of the final image, and can be mathematically calculated to give the best results. Amateur astronomers often like to build their own optical systems, but before home computers became widely available, it was often quicker and easier to put together an experimental design than to perform the painstaking calculations needed for light ray diagrams. With a computer, the calculations that might have taken a week can now be completed in a few minutes.

Locating a star in the sky is another problem that a computer can help us to solve. Stars are not fixed objects — they trace trajectories across the sky in the course of the night (an effect caused by the earth's rotation), and their positions in the sky are subject to seasonal variations. The method of locating a star is similar to the system of latitude and longitude used in terrestrial geography. If we imagine a co-ordinate system projected across the inner surface of the night sky, each celestial object can be located by two co-ordinates called the 'declination' and the 'right ascension'.

Every object can then be marked down on a star map according to its co-ordinates, and the individual maps combined into an atlas of the skies. Such atlases are very important for observing planets and other objects that move against the backdrop of fixed stars, or for discovering completely new objects like comets. These star atlases are now being put into databases for home computers. These database programs also include information about the brightness or luminosity of individual celestial bodies, the spectral quality of the light they omit (obtained when the light is passed through a prism or analysed by a spectrometer), and the type and age of the star. All of this information can be displayed on the computer's television set or monitor, in the form of maps showing the sky from any latitude at any desired time.

Because of the earth's rotation, stars will drift out of view in a few minutes, even on a telescope with a large field of view. Using one that can focus on only a narrow area of sky, it is essential to track the telescope continuously to compensate for the earth's movement. Mechanical drive motors have been used for many years, but recently the amateur has had access to computer-controlled systems. The telescope is mounted on an 'equatorial' axis pointing to true north, and the drive motor rotates the axle at precisely the same speed as the earth moves, which keeps the object in the field of view. Attached to the axle and telescope are shaft encoders and digitisers, which send a digital signal to the computer for the celestial co-ordinates and monitor the activity of the drive motor. In this way, a telescope can be left all night, under computer control, to track a faint star, while its image slowly builds up on a photographic plate. An adaptor is available in America called the Celestial Navigator Mk II, which connects a telescope to the home computer through the expansion interfaces.

Computers can also assist telescopes by allowing for those atmospheric changes, like variations in temperature and humidity, that refract and bend the light as it passes through the atmosphere. They can also compensate for and correct distortions of the image received through the telescope, by various techniques of image enhancement.

Astronomy has not only used computers for its own advancement, but has also contributed to the development of computer science. The language FORTH was devised by an astronomer, Charles H Moore, in 1971 at Kitt Peak observatory in Arizona for controlling the radio telescopes and processing data.

Books are now appearing devoted to home programming for astronomy enthusiasts, and there is even a magazine called *Apex,* in which home computer owners exchange and publish programs. Software has been written to calculate the times of Easter, the conversion of historical dates into the Julian calendar, the rising and setting of the moon and daily tables to pinpoint the exact position in the sky where astronomers expect to see the return of Halley's comet in 1986.

**Grammar School All-Stars**
The pupils and staff of Kettering Grammar School in Northamptonshire have long been held in high esteem for their work in astronomy, especially in the field of satellite tracking. On a number of occasions, KGS has been the first observer station to detect the presence of a new satellite in orbit

JOHN DRYSDALE/COLORIFIC

In America, some amateur astronomers have gone as far as to adapt the newer technologies of speech recognition and speech synthesis to help them. A computer is programmed to recognise certain command words so that when an observer walks into the building and calls out 'open', the dome slides open, and a further call 'rotate the dome' engages the motors to revolve the dome on its wheels. Speech synthesis is also very useful in the darkness of an observatory to output information from the computer. It could, for example, count time signals out loud to help the observer make accurate photographic exposures.

Professional astronomers make use of telescopes that register part of the electromagnetic spectrum other than light — such as radio waves or X-rays. With the increase in size of the telescope aperture and the accompanying increase in weight, engineering problems become critical. In 1964, the Jodrell Bank Mark II, a 38 by 25-metre (125 by 83-foot) elliptical disc near Manchester, was the first telescope to use a digital computer to convert the celestial co-ordinates into the instructions needed by the various drive motors, and the process was done continuously four times per second.

**Radio Stars**
Following the discovery of the presence of radio-emitting objects in remote galaxies, this massive radio telescope was built at Jodrell Bank, to the south-west of Manchester. Radio telescopes, which operate like huge aerials, can detect objects invisible through even the most powerful optical telescopes

# Step By Step

**When it is necessary to measure linear or angular movement by means of an optical sensor, binary coding proves fallible, so the Gray code was devised**

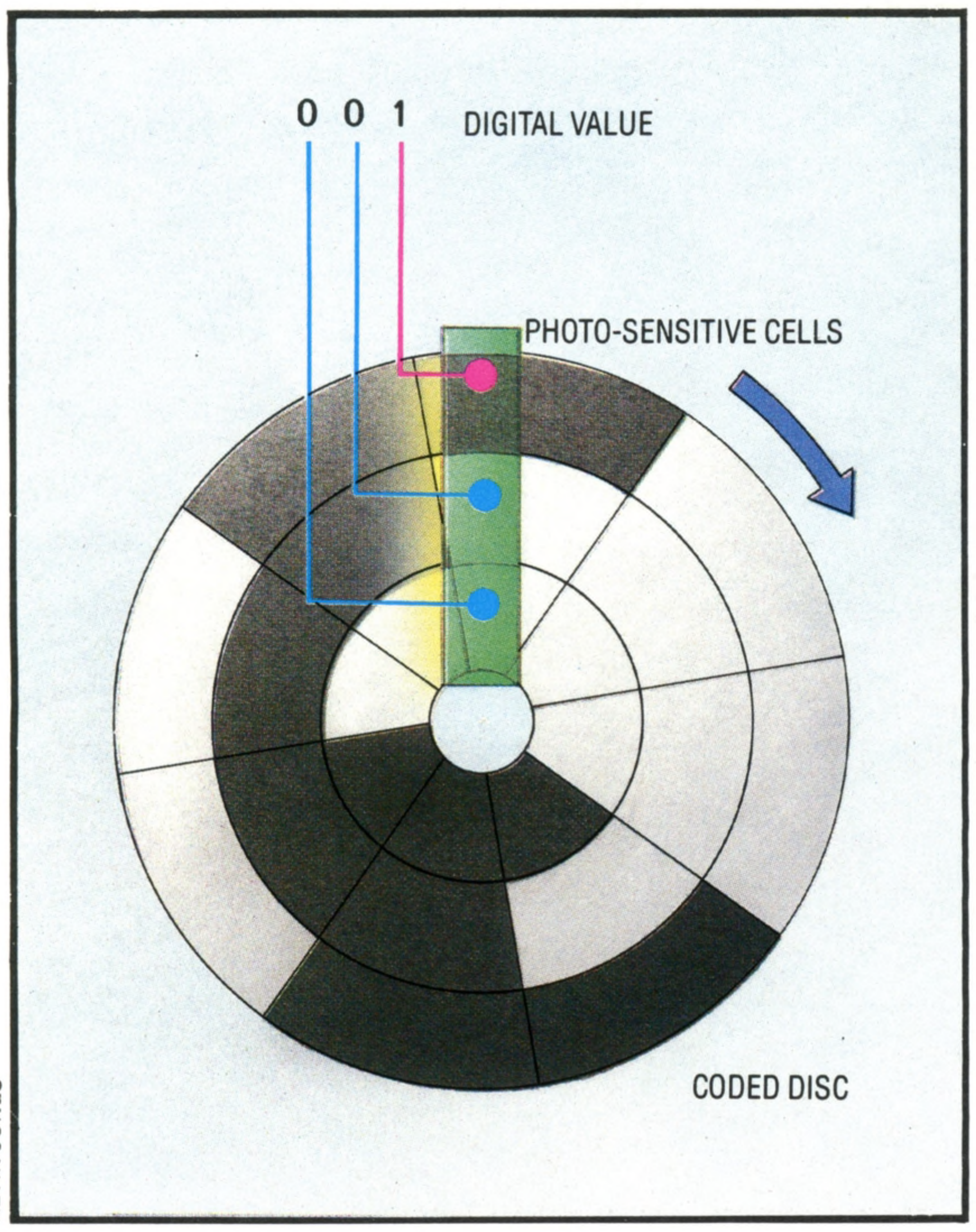
**Odd One Out**
This table shows the binary and Gray code equivalents to decimal 0 to 15.

| Decimal | Binary | Gray Code |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

There are many tasks in which the physical position of a movable object must be precisely determined and relayed to a computer. In robotics, for example, the computer must be aware of the positions and orientations of all its limbs; and in computer-controlled machine tooling, the position of the milling table must be accurately determined. How then can a position be converted into a binary value for computer processing?

One method is to use an analogue system. This could involve connecting the moving device to a variable resistance and then feeding the resulting voltage into an analogue-to-digital convertor (or directly into the analogue port if your computer has one). However, this is not a very accurate system, and the mechanical parts are prone to wear and tear.

The alternative is to print a binary code onto the moving device and read this directly into the computer. The code is usually printed as a pattern of black and white blocks along the top of the device, and is read by means of a light source shining onto a pattern in conjunction with a line of photo-sensitive cells, each responsible for one digit in the binary pattern. As the workpiece moves, the pattern under the light cells changes and this gives a binary output that defines the device's position. In addition to linear arrangements, radial patterns are also used to encode angular movement, such as that of a joint in a robot arm.



**Angle Of View**
The angular position of a device can be read into a computer by means of a disc on which a code is printed. A light shines on the device, and a line of photo-sensitive cells detect the pattern. A parallel digital signal is produced that changes as the device moves. The problem with using binary as the code is that if the disc stops on the join between two values, a meaningless result can be produced. Gray code overcomes this problem

Problems arise, however, when the device moves from one binary code to another — and particularly if it should come to rest halfway between two of them. The printing is accurate to only a finite tolerance, and when the device stops on a join between two codes, the light cells could pick either of them to read. If the workpiece comes to rest with the light cells shining on the join between binary position 11 (1011) and 12 (1100), for example, then it is only the most significant bit (that is, the leftmost 1) that can be relied upon to give the correct output, whereas the other three light cells will have conflicting values to read. There are certain situations when all the bits change, such as the join between binary seven (0111) and eight (1000), so the minor inaccuracies in printing could produce incorrect readings on all the cells. The result could be a totally false value for the position, and the computer would have no way of knowing that this had happened. The consequences could be disastrous.

What is needed therefore, is an alternative counting system to binary, where only one bit changes with each increment. This will mean that only one bit can be in doubt on any join, and the output can be in error by no more than one position. This alternative code is called the Gray code, and is uniquely determined by these requirements: moving from one value to the next, only a single bit changes, and this should be the rightmost bit that will still result in a unique pattern. So if we start with 0000, as with binary, the number one will be represented by 0001. However, to represent two, we must change the second bit to get 0011. For three it is now possible to change the first bit, and get 0010. Note how this differs from the binary sequence for the same numbers: 0000, 0001, 0010, 0011.

The panel shows this process extended to the equivalent of decimal 15, together with the binary equivalents for reference. As an exercise you might like to work out the Gray code values beyond this.

Computers could be designed to perform arithmetic and function internally in Gray code, but this would be inefficient and unnecessary. Some means is therefore needed to convert from Gray to binary, which could be done in hardware or software.

# Apple IIe

**Many wonder at its longevity, but the Apple II is still amongst the most versatile of small microcomputers, and has more add-ons available than any other**

In many ways the Apple II is the machine that started everything, because although it wasn't the first microcomputer available, it was as a result of the release of this machine that facilities such as colour, high resolution graphics and built-in sound became accessible to the home computer user. However, although these features are important, the really significant features of the Apple II, which made (and continue to make) the machine so phenomenally popular, are much less obvious at first glance.

The most apparent feature was the standard of the documentation, which deliberately set out to make every aspect of the machine as clear as possible to its users. This was in direct contradiction to the attitude of the few other companies in the field at the time, who maintained (and in some cases still do maintain) a strict silence about the inside of their products. As a direct result of this freely accessible information, the other significant feature of the Apple II became well known.

This second facility was the row of 'expansion connectors' that occupy the rear section of the machine's motherboard. It is the flexibility of organisation of these slots that has made possible the wide range of add-on products that are available for the machine. This diversity has in turn led to the use of the Apple II in a multitude of different ways.

Many computers have some form of expansion connector, of course, but generally only one or two, and these often appear as a fairly small area of memory. The Apple has seven on the latest version (the IIe) and eight on the earlier models (II and II+), each of which appears to the CPU as two sections of memory (one quite small, and the other a very useful 2 Kbytes).

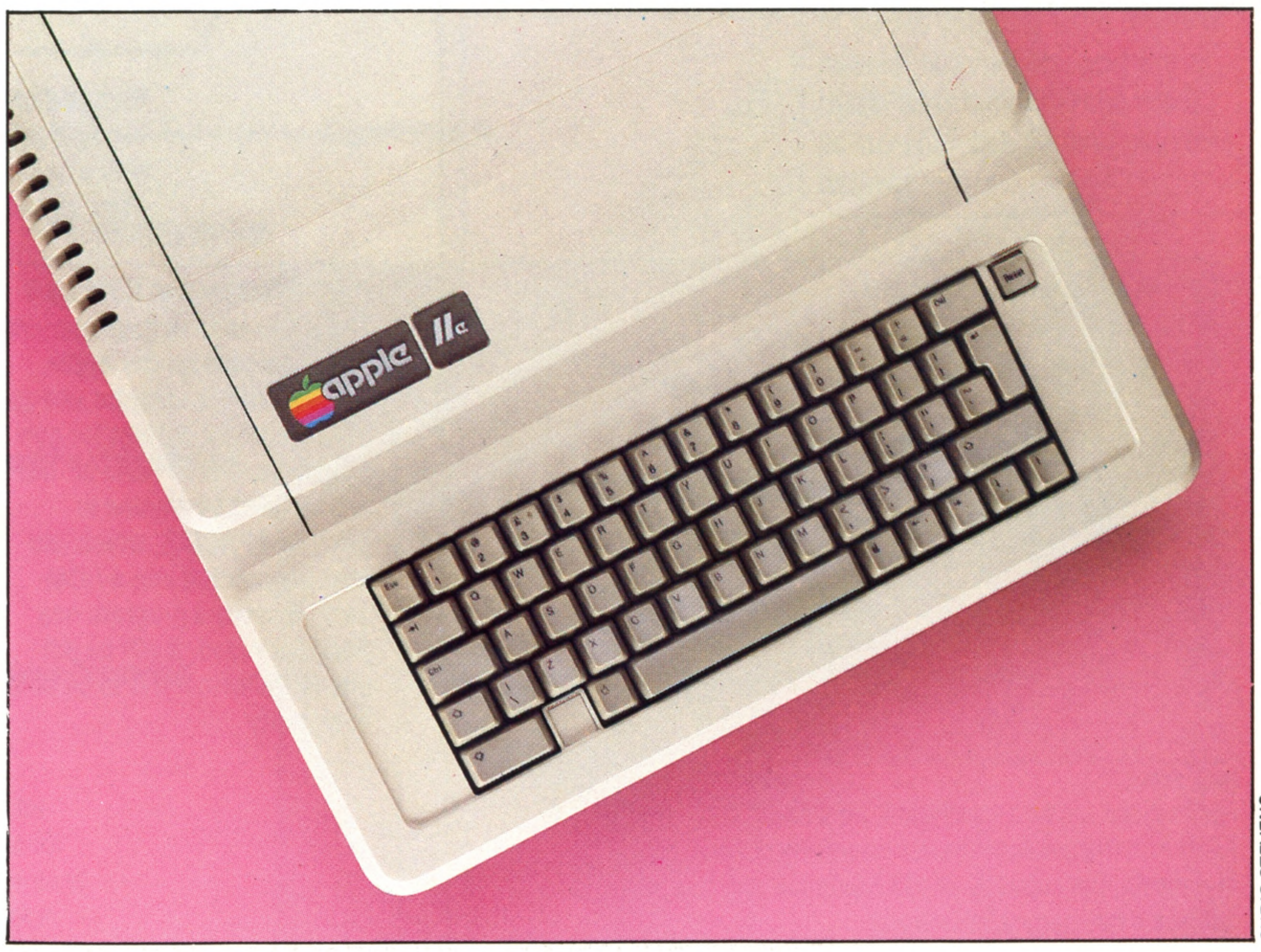
Consequently, a peripheral card that is plugged into the Apple can have 2,048 bytes of controlling program on board. This makes using such a card very simple, since there is no need to link special driver programs or re-write the operating system.

A wide selection of cards is now available for the Apple II, ranging from relatively simple input/output interfaces to very advanced cards, such as light pens and large RAM cards that can expand the memory to a Megabyte or more. However, because the 6502 can address only 64 Kbytes, the memory is arranged in 64 Kbyte 'banks', one of which is selected at a time. Complete computers with advanced 16- or 32-bit CPU chips can even be plugged in to run in parallel with the machine's 6502 processor, and can create a system that has as much (or more) computing power than a mini-computer.

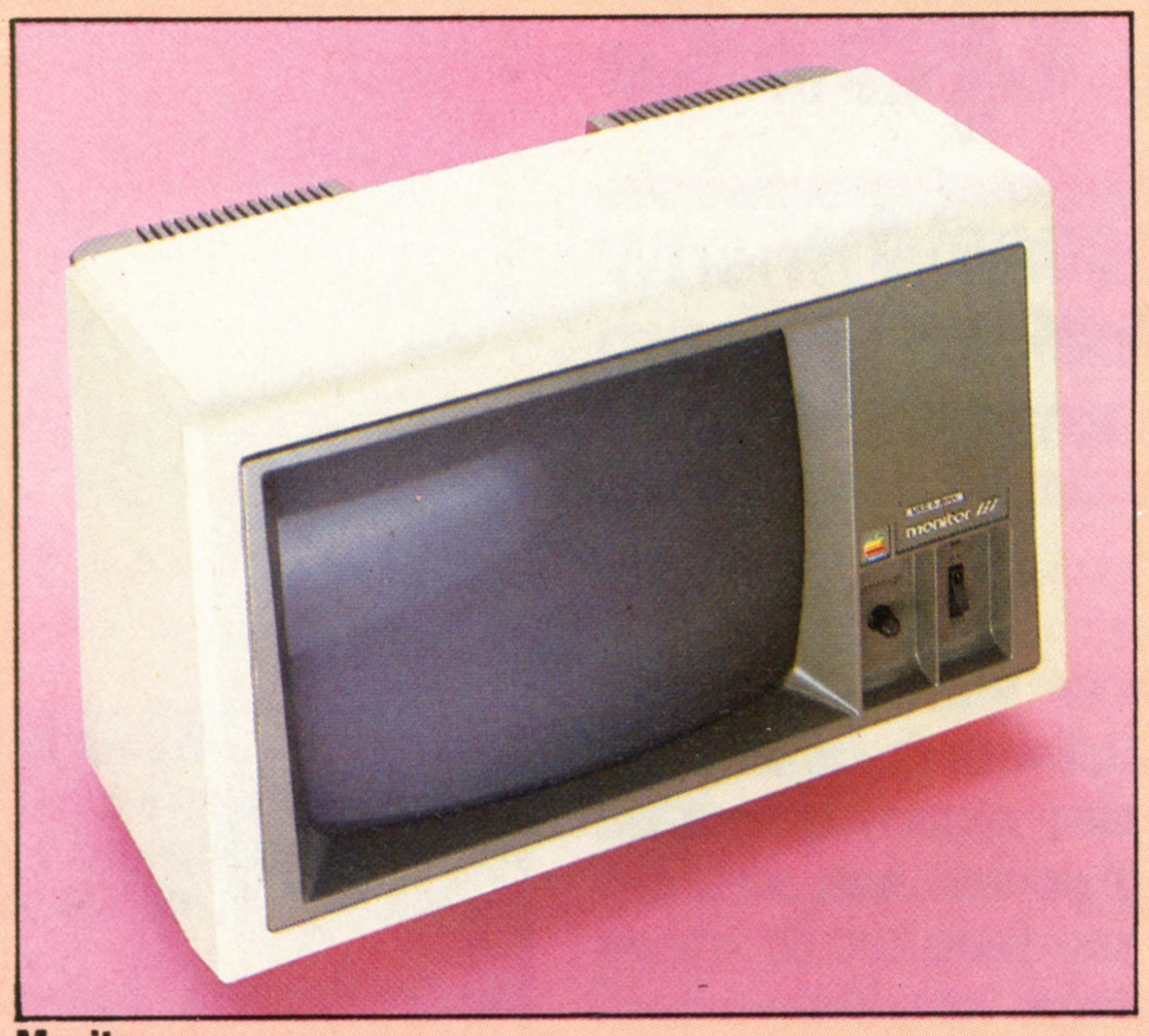
Naturally, a very large range of software has been developed to make use of this wealth of hardware, and now the library of Apple programs is easily the world's largest — even bigger than the list of programs that use the standard CP/M operating system. In fact, the CP/M library can be included on the Apple's list, because even though the Apple has a 6502 processor, it can run CP/M programs with the help of one of the most popular add-ons — a Z80 card. (CP/M will run only on a Z80 microprocessor). Once one of these has been plugged into an Apple slot, the machine will function as a perfectly normal CP/M computer.

Although quite small and unassuming in appearance, the Apple II is a machine that tends to be the subject of superlatives. It is the home computer with more different operating systems (11), languages (at least 27), and text editors (a dozen or more) than any other machine.

The story of the Apple II is far from completed; in 1984 an entirely new operating system for the machine was released. It was given the name ProDOS and incorporated many features that made the Apple II's performance surpass in some respects that of more expensive systems.

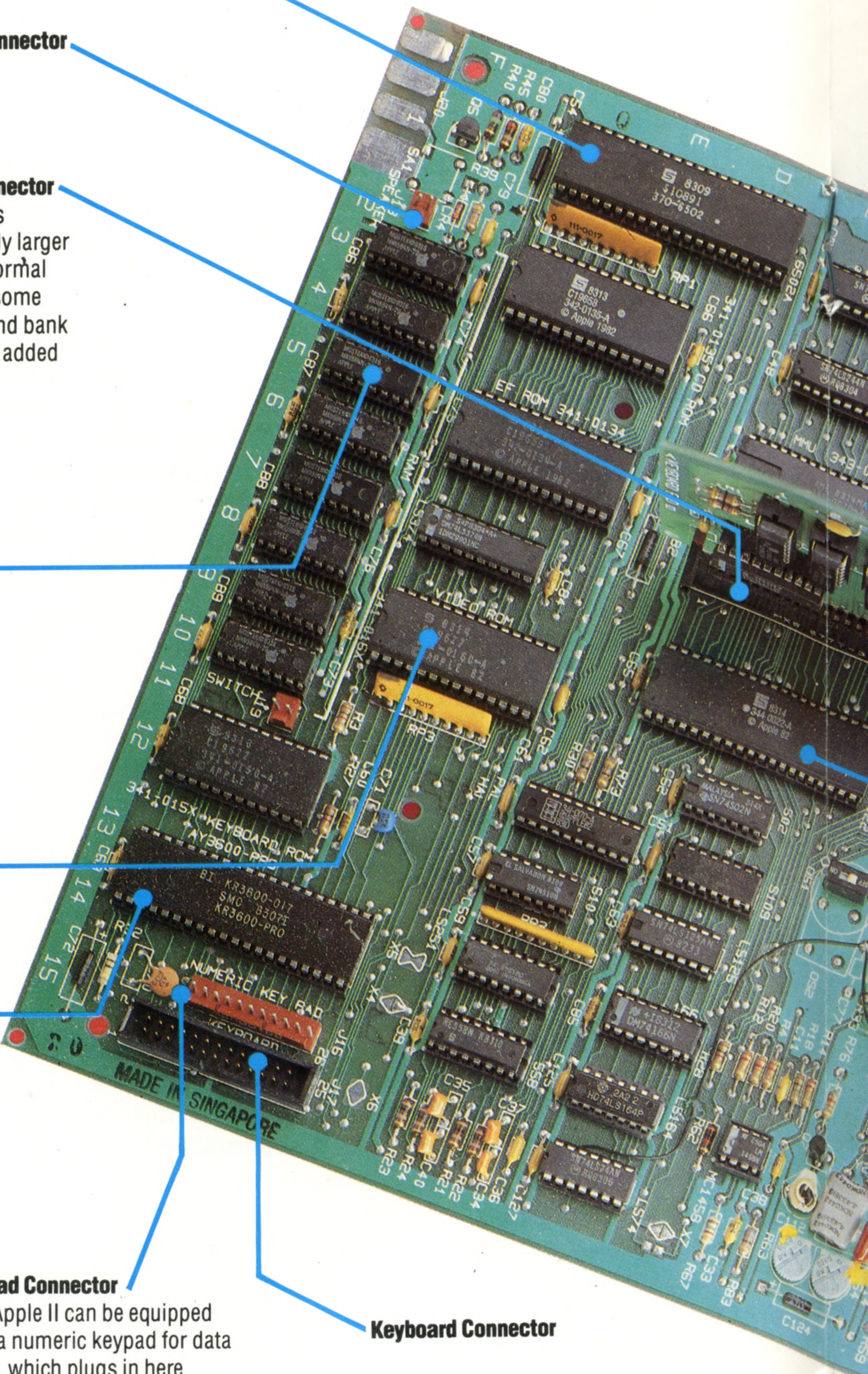

**Apple Keyboard**
The keyboard has been designed to the highest standards, with word processing in mind. The keys are properly sculpted, and the cross-section forms an arc for ease of use. The two keys marked with the Apple logo on either side of the space bar are control keys used in applications programs. At first sight, the Reset key might seem to be unfortunately close to the Delete key. However, to reset the computer, it is necessary to hold down CTRL and then press Reset

CHRIS STEVENS



CHRIS STEVENS

**Monitor**
The Apple IIe will work with any dedicated monitor though of course Apple's own unit blends best with the physical design. It can display up to 80 columns of text and is fitted with an anti-glare filter to reduce eye fatigue



**6502 CPU**

**Speaker Connector**

**Auxiliary Expansion Connector**
On the Apple IIe, slot 3 is duplicated by this slightly larger slot, which has all the normal bus-signals, as well as some extra to handle the second bank of 64 Kbytes that can be added on

**RAM**
The Apple IIe has a straight 64 Kbytes of RAM, held in eight 4164 chips, but the actual mapping is rather more complex, since the addresses of the peripheral chips are placed in the middle of the RAM

**Video ROM**
The characters printed on the screen are generated by this ROM, which is available in several different language-sets

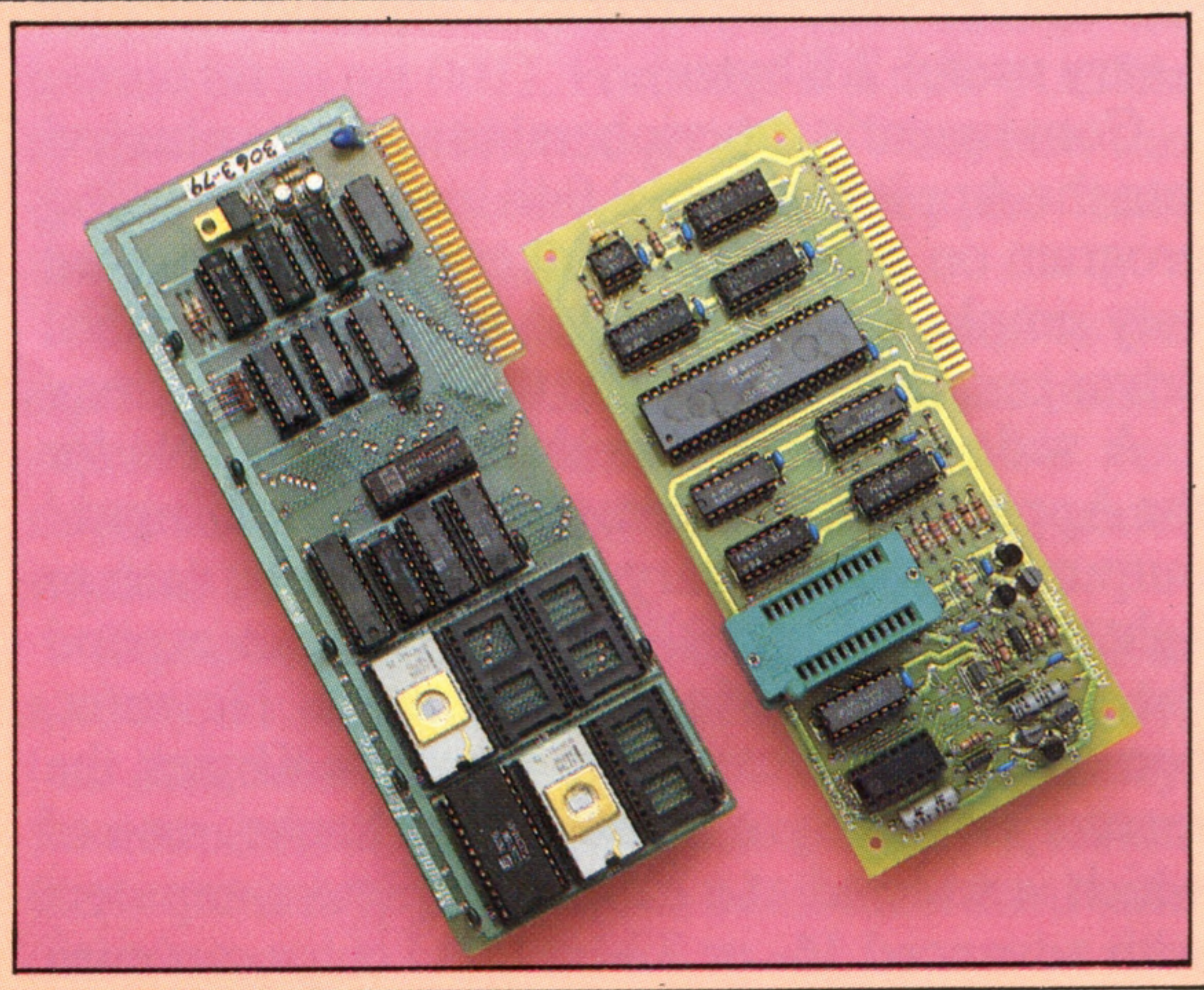
**Keyboard ROM**
The mapping of the keyboard is controlled by this ROM, which is changed to suit the needs of foreign countries, so that France can have AZERTY keyboards, for example

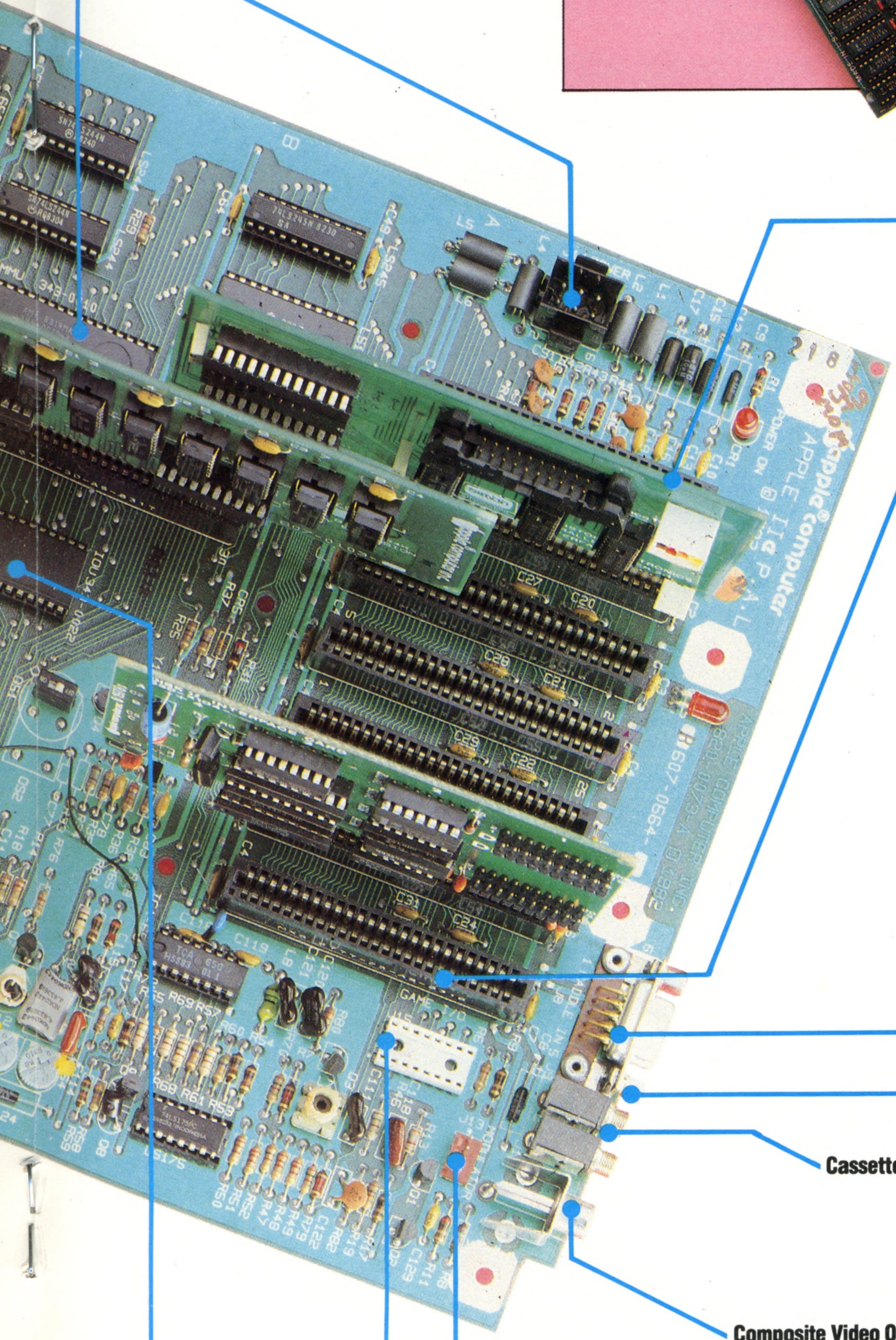
**Keypad Connector**
The Apple II can be equipped with a numeric keypad for data entry, which plugs in here

**Keyboard Connector**

**Disk Drive**
Apple disks are non-intelligent, so a controller card must be fitted into the main computer board. By contemporary standards, the capacity of 143 Kbytes is very low, but it is adequate for many applications

**EPROM Blower And ROM Card**
The wide range of development tools available for the Apple means that it is ideal for use as a development system for new programs. The EPROM blower can be used by anyone to produce EPROMS, which can then be inserted into a ROM card. When the program has been perfected, it will be put into ROM form if it is to be sold in quantity





**Memory Management Unit**
One of the two ULAs, which constitute the major difference between the Apple II and the Apple IIe. This one controls the 80-column screen as well as the second 64 Kbyte bank of RAM

**Power Connector**

**Slot 1**
This is normally occupied by a parallel printer interface

**Slot 7**
Special video signals are available in this slot only, so video-related cards such as light pens and colour modulators are often found here

**Game Port DIL Socket**
One of the most innovative features of the Apple was the game port, which gave a minimal but useful form of analogue input

**Cassette Input**

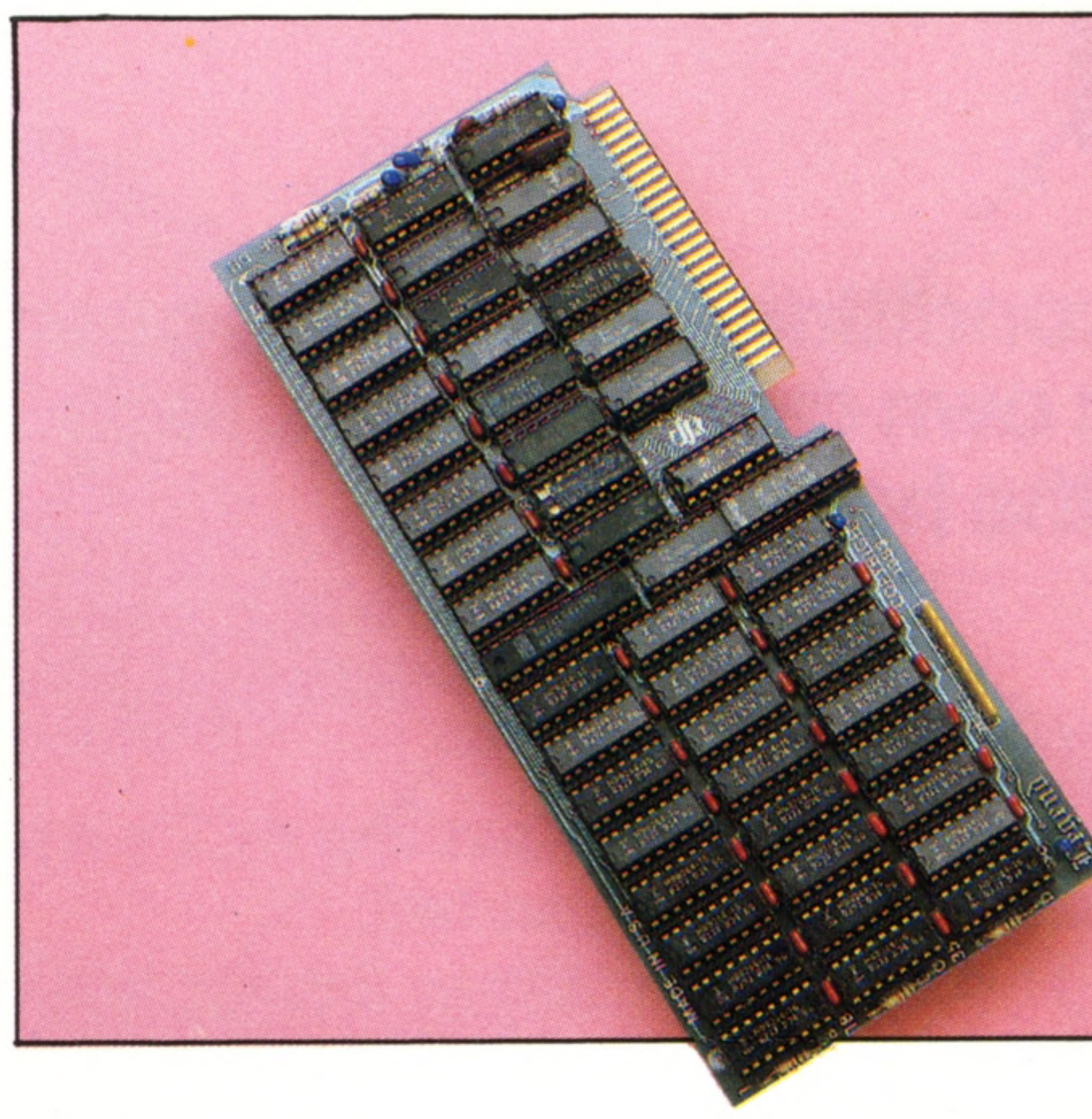
**Cassette Output**

**Composite Video Output**

**Auxiliary Video Connector**

**Input/Output Unit**
The ULA that handles the addressing of the auxiliary connector

**Game Port D-Connector**
The normal 16-pin socket for the game port is too fragile for everyday use, so the Apple IIe is fitted with a small D-socket in parallel



**256 Kbyte RAM Card**
The advanced address mapping of the Apple allows multiples of up to 2 Kbytes to be bank-switched, resulting in slightly more complicated access. However, this allows very large blocks of RAM to be used in the machine. This card carries 256 Kbytes, but can be expanded to 1 Mbyte!

## APPLE IIe

**PRICE**
£845 plus VAT

**SIZE**
460 × 385 × 115mm

**CLOCK SPEED**
1 MHz

**MEMORY**
16 Kbytes ROM

64 Kbytes RAM

Expandable to 128 Kbytes or more with bank-switching

**VIDEO DISPLAY**
24 lines of 40 characters, monochrome only. Low resolution graphics of 48 × 40 in 16 colours. High resolution graphics of 192 × 280 in 6 colours

**INTERFACES**
Cassette, composite video, 7 expansion slots, game port

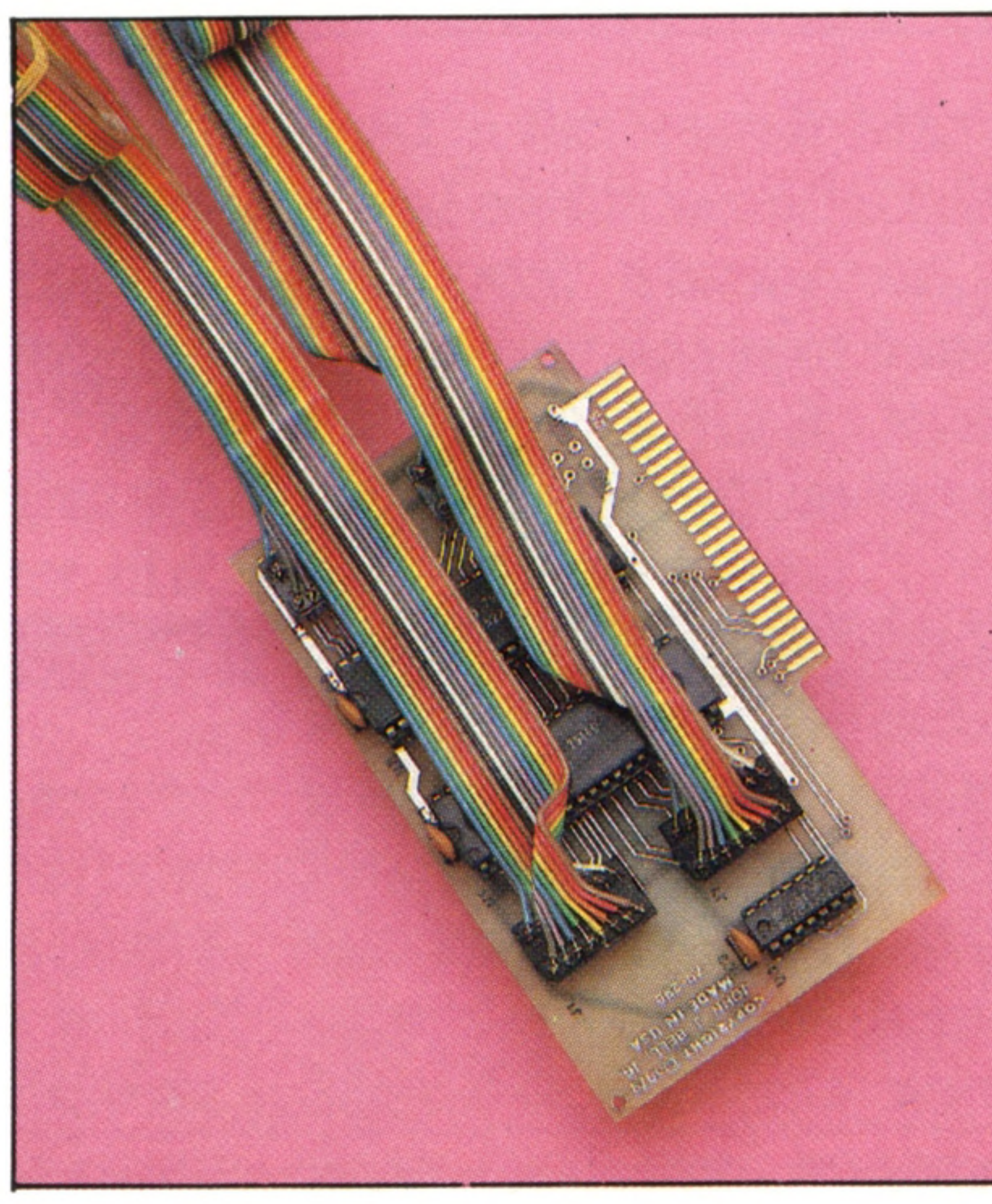
**LANGUAGES SUPPLIED**
Applesoft BASIC

**OTHER LANGUAGES AVAILABLE**
Most of the common alternative languages, and some rarities, are available

**COMES WITH**
Installation and BASIC manuals, TV lead

**KEYBOARD**
62 high-quality keys

**DOCUMENTATION**
The accompanying documentation is of a very high standard, though the advanced material needed for a full understanding of the machine has to be purchased separately and it is rather expensive. A huge range of books catering to all levels of interest is available, and the machine is probably the most comprehensively covered in this respect



**General-Purpose I/O Card**
Sometimes a card can have so many possible functions that a controller-ROM would restrict the user and become a liability. This card, which has two 6522 versatile interface adaptors, is such an example. It has 40 separately controllable I/O lines, two shift-registers which are used for converting data from parallel to serial form, and four 16-bit timers

CHRIS STEVENS

# Fully Loaded

**Hard disks need scrupulously clean conditions in which to operate. The Winchester disk, sealed inside its own casing, makes high capacity and fast access available to the home user**

Although home computers have assimilated many of the features of small business machines over recent years, there is one area of computer technology in which they they remain relatively unsophisticated: disk storage space. While a home computer user would be grateful to have a single floppy disk capable of holding 100 Kbytes of data, business machine requirements demand considerably more space. A pile of floppy disks, with all the business information scattered between them, is no solution to these large storage needs; and consequently, rigid disks capable of storing far greater amounts of data were developed. The pioneering work on these rigid disks began at IBM in the 1960's. Because the original disks contained a 30 Megabyte storage capacity on each drive, they were dubbed the '30/30', and hence, by analogy to the rifle, 'Winchester' disks.

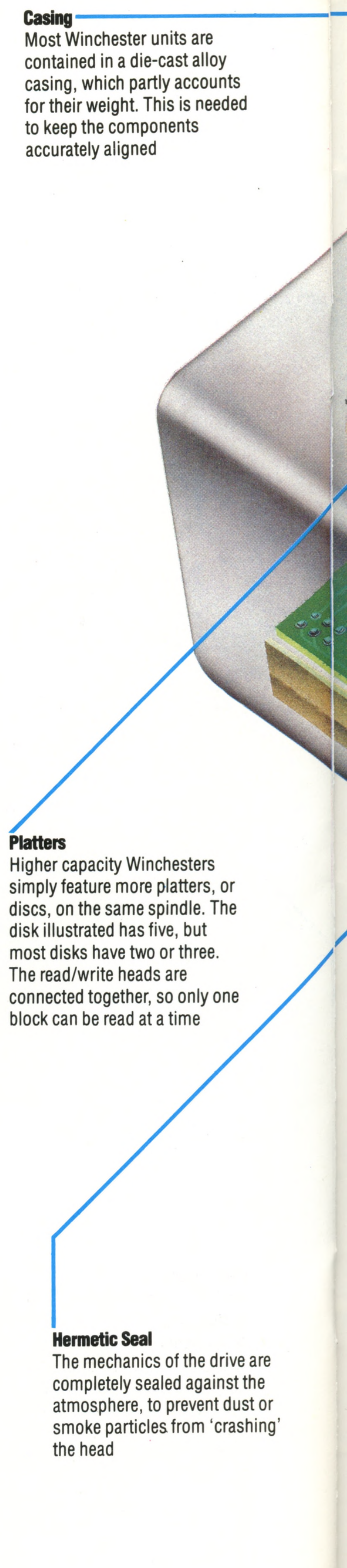
Winchester disks use rigid platters rather than the flexible pieces of plastic used in floppy disks. This allows the number of tracks on the disk to be increased from a typical working maximum of 96 tracks per inch (tpi) for floppy disks to several hundreds of tpi for Winchester disks. With increasing sophistication in disk technology it has become commonplace to fit five, 10 or even 20 Megabytes of storage into the same sized box as a 5¼" floppy disk drive.

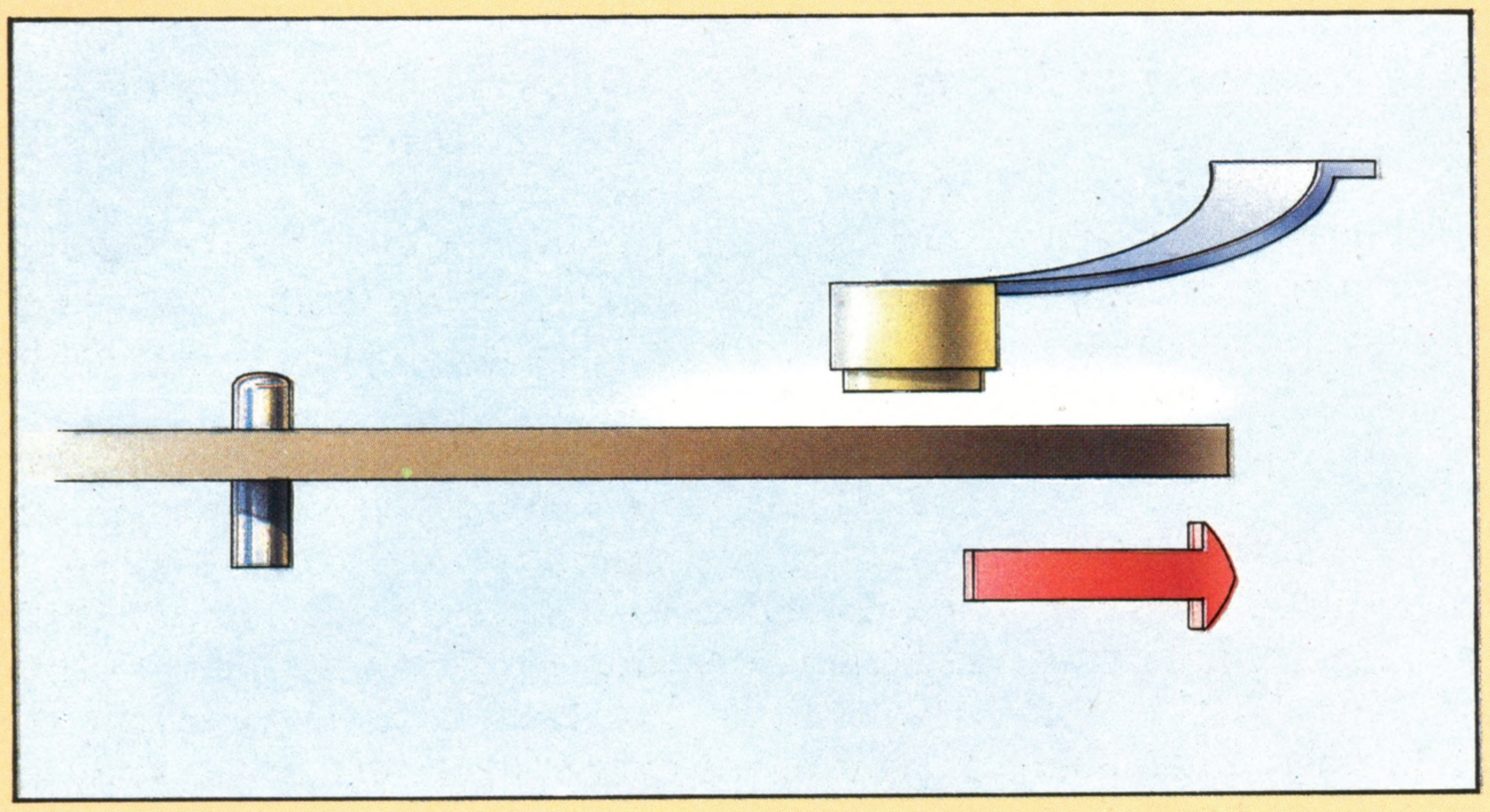
Home computer users are now beginning to benefit from the trend towards rigid disks with the availability of Sony 3½" and Hitachi 3" micro floppies. These are semi-rigid disks and are capable of storing as much information as any 5¼" floppy disk. The BBC Micro and the Oric-1 are among the first to have such devices available as add-ons, while computers like ACT's Apricot are fitted with them as standard.

This increase in storage density, however, has created other problems. The accuracy demanded by the head positioning mechanism, for example, required a completely new method of moving the head. The solution to this problem was found in the audio industry: Winchester heads are often driven into position by an electromagnetic coil of the type found in loudspeakers. Passing an electric current through a coil creates a magnetic field and this in turn causes a soft iron plug in the centre of the coil to move a precise distance. By attaching the head to the end of the plug (suitably isolated from any magnetic effects, of course) it can be moved across the disk surface very quickly and accurately.

The head 'flies' across the surface of the disk on a cushion of air, never actually touching it. This significantly reduces wear and tear on the disks, but means that they must be sealed in airtight boxes to prevent problems caused by dust and other foreign bodies. Generally, this means that the disk is fixed inside the drive and cannot be removed, though Winchester disks with removable cartridges are now starting to become available. These cartridges are usually self-sealing and open only to allow the head access to the disk surface when the cartridge is inserted into the drive. To prevent any dust getting in, the air pressure inside the cartridge is often kept higher than that outside, and all the air pumped into the unit is filtered first.

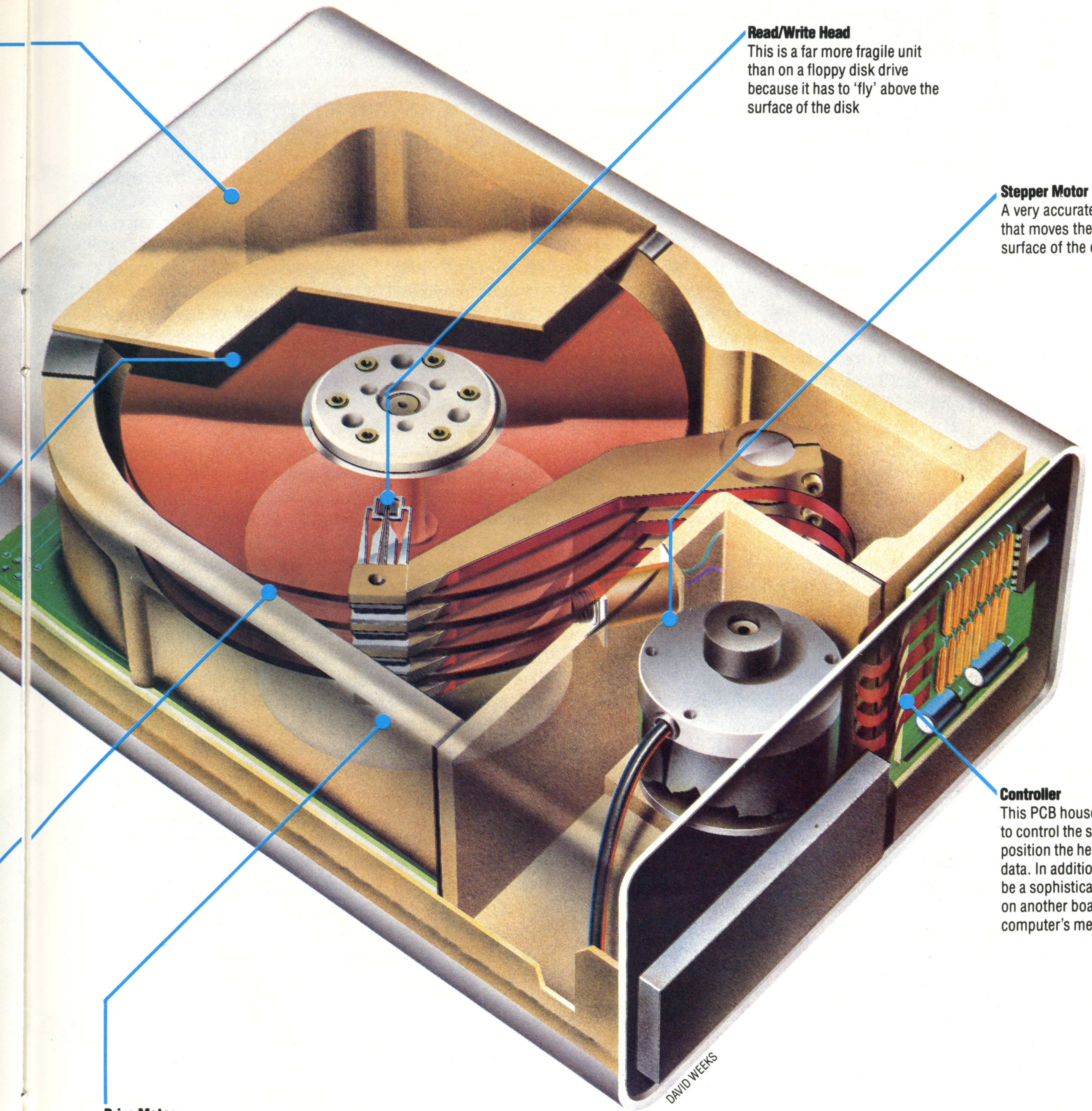
Another advantage of rigid disks is that multiple platters can be used. A 10 Megabyte Winchester disk is simply constructed from two


**Casing**
Most Winchester units are contained in a die-cast alloy casing, which partly accounts for their weight. This is needed to keep the components accurately aligned
**Platters**
Higher capacity Winchesters simply feature more platters, or discs, on the same spindle. The disk illustrated has five, but most disks have two or three. The read/write heads are connected together, so only one block can be read at a time
**Hermetic Seal**
The mechanics of the drive are completely sealed against the atmosphere, to prevent dust or smoke particles from 'crashing' the head



**Floating Heads**
Unlike a floppy disk drive, on which the head rubs against the recording surface, on a Winchester drive it 'flies' very close to it. The disk rotates so fast that the 'skin effect' creates a cushion of air that supports the head. If the head 'crashed' against the disk, it would probably remove the magnetic recording surface

KEVIN JONES



**Read/Write Head**
This is a far more fragile unit than on a floppy disk drive because it has to 'fly' above the surface of the disk

**Stepper Motor**
A very accurate electric motor that moves the head across the surface of the disk

**Controller**
This PCB houses the electronics to control the spin speed, position the head and read the data. In addition, there needs to be a sophisticated DOS, either on another board or in the computer's memory

**Drive Motor**
A DC (direct current) motor and a small generator are mounted on the same shaft that in turn spins the disk. The output from the generator is a measure of the speed of the motor and is fed back to a special controlling circuit. This is how the speed is determined so accurately

five Megabyte platters placed in the same box. The management of all this storage space is achieved by splitting it into a large number of sections. To maintain compatibility with existing software, which expects only to deal with floppy disks, these sections generally approximate to the capacity of a normal floppy. A Winchester disk looks, to the computer at least, like lots of separate floppy disk units.

When a Winchester disk is formatted (that is, the tracks and sectors are first marked out) the DOS (see page 324) must be capable of skipping over the 'bad sectors', which have faults in the magnetic recording surface. The only comparable operation on a home computer is the formatting used in the Sinclair Microdrive. On a floppy disk, a bad sector would result in the whole disk being scrapped; whereas on a Winchester disk, the formatting program merely notes that the sector is unusable and blocks it off from further use. After all, with five million bytes of space who's going to miss a few hundred?

In keeping with the advances being made in the rest of the computer industry, the Winchester disk is getting smaller in size, and there is now a 5 Megabyte micro-Winchester disk. With such rapid developments in disk storage units, a home computer for less than £500 that features 10 Megabytes of data storage isn't as far away as might have been imagined.

# Assembly Lines

**We can now bring together the sub-programs that will process our computerised address book, and examine ways of making the program more user-friendly**

Although many details of the address book program have yet to be finalised, the overall structure should now be becoming clear. At this point in the development of a program of any size it is as well to draw a block diagram of the program and to think through the flow of activities in the program.

This is also the point at which the program writer should think about the 'human interface' and 'user image' aspects of the program. These important concepts and practices are often not given the attention they deserve, even in commercial software.

'Human interface', simply defined, means the 'ergonomics' of the software, or how easy it is to use. 'User image' is concerned with how the user perceives the program being used. We shall examine these concepts with regard to our program as developed so far, and determine to what extent we will be able to implement them.

The table shows the main blocks of the program that have been considered so far. As a convention, and simply to keep things tidy, we have used names with six characters for procedures or subroutines, seven characters (including $) for string arrays, four characters for simple numeric variables and five characters (including $) for simple string variables that are global. Local variables (in loops, for example) will usually be single letters.

**MAIN PROGRAM BLOCKS**

| | | |
|---|---|---|
| | CREARR | (creates arrays and initialises variables) |
| INITIL | RDINFL | (reads in file from tape or disk) |
| | SETFLG | (sets flags and modifies variables) |
| GREETS | | (prints greeting message) |
| CHOOSE | CHMENU | (prints options menu) |
| | INCHOI | (assigns option to CHCE) |
| | FNDREC | (locates and prints a record) |
| | FNDNMS | (locates names from partial input) |
| | FNDTWN | (locates records for specific town) |
| | FNDINT | (locates names from initials) |
| EXECUT | LSTREC | (lists all records) |
| | ADDREC | (adds a new record) |
| | MODREC | (modifies existing record) |
| | DELREC | (deletes a record) |
| | EXPROG | (saves file and exits program) |

Each of the major program blocks in the second column needs to be further broken down into sub-units, and the sub-units will need to be further refined until we have enough detail to write the actual code in BASIC. The processes involved in this form of 'stepwise refinement' have been illustrated for many of the blocks in earlier parts of the Basic Programming course.

Assuming that all or most of the program modules have been worked out, coded into BASIC, and individually tested, how can they be linked together to form a complete program? The best way to tackle this problem is to save each module on tape or disk, giving it the same filename used in the program development notes. Thus ADDREC can be written and tested as far as possible, and then saved under the filename ADDREC. Normally, when a program is loaded from tape or disk, we use the LOAD command, followed by the filename, as in LOAD "ADDREC". This has the effect, however, of clearing everything in memory, so if we load ADDREC and then subsequently load MODREC, the whole of the ADDREC program will disappear.

Fortunately, there is a partial solution. The MERGE command loads a program from tape or disk without erasing any program already in memory. But there is one important proviso. If any of the program line numbers in the MERGEd program are the same as line numbers in the program already in memory, the new line numbers will overwrite the old line numbers and cause chaos. Versions of BASIC with the RENUM command can get round this by renumbering the lines in a program module before they are saved, so that when they are merged there will be no conflict.

Unfortunately, many versions of BASIC on home computers do not have the RENUM command, and therefore careful planning of the line numbers from the beginning will be necessary. When a full chart of all the major program modules has been worked out (as we have partially done in the table), starting line numbers for each block can be assigned. Parts of the program likely to have extensive modifications or changes, such as the main program or file-handling parts of the program, should have increments of 50 or even 100 in order to leave plenty of room for additions. Program modules less liable to modification, such as the GREETS routine, can have line number increments of 10. Putting lots of blank REMs in the program not only makes the program easier to read, but also allows additional statements or calls to subroutines to be added later. If your BASIC doesn't feature MERGE either, then you will have to type in the various modules as they are written, and save them together.

The program blocks in the table have been merged as a 'trial run' in the listing printed here, to illustrate the pitfalls of the 'try it and see' approach encouraged by languages such as BASIC. Our program would not run properly because the flow of control through it has not yet been thought through carefully enough. There is no point in typing in the whole of this program just to find out that it will not work, but if you have saved all the routines from earlier parts of the course, and if your BASIC has the RENUM command, you can try renumbering and then MERGEing to produce a similar listing.

The first block in the main program is INITIL, which is supposed to initialise variables, dimension arrays, read in files, assign the data to the arrays, set flags and so on.

The *INITIL* subroutine is broken down into *CREARR* (to create the arrays), *RDINFL* (to read in files and assign the data to the appropriate arrays) and *SETFLG* (to set flags etc.).

When all this has been done, the program moves on to *GREETS*, a subroutine to print a greeting message on the screen. The last part of this routine waits for the user to press the space bar for the program to continue.

The program then goes on to *CHOOSE*. This comprises two parts: the first presents a menu of the options offered by the address book program; the second accepts the choice input from the keyboard and assigns the (numeric) value to a variable called CHOI.

The value of this variable is used by the next program block, EXECUT, to select one of nine further program blocks. All of these, except EXPROG, will need to return to *CHOOSE* after they have been executed so that the user has the opportunity of selecting another option. This will not be required if 9 (EXPROG) has been selected because this option is supposed to terminate the operation of the program.

The chief problem with this program as it stands is that the control flow is not correct. INITIL insists that we read in a file from mass storage whether a file exists or not. If the program is being run for the first time, no records will have been entered and there will be no data files on the tape or disk. Any attempt to open and read a non-existent file will result in an error message and the program will not work.

What is required is to have the *RDINFL* routine called only by one of the EXECUT modules, and then only once each time the program is run. This suggests that there should be an INFL flag, originally set to 0, that will be set to 1 once the file has been read in. If it has been set to 1, it will inhibit further attempts to read in. ADDREC will then always search through the arrays to locate the first empty element and will write the information there. This record will almost certainly not be in the proper sort sequence, so there should be a RMOD flag which will be set to 1 when executing The RMOD flag should also be set to 1 if MODREC or DELREC are executed. You can try writing the relevant code to achieve this, or if you simply want to run the program change line 1310 to RETURN.

Adding a record, deleting one or modifying one all mean that the sequence of records is likely to be out of order, so any module (FNDREC, for example) should first check RMOD to see if any changes have been made. If they have, we could either insist on a sort before a search is made, or put up with an inefficient search through a pile. EXPROG will automatically check RMOD and call the sort routine if it is set (to 1) before saving the data in the file on tape or disk.

The 'human interface' aspects of the program, which we mentioned earlier, can be broken down into the following categories:

User interface
User image
Error recovery
Security
Adaptability

User interface refers to the way the user of the program communicates with the program. We have opted for the use of menus throughout (rather than commands). Many people prefer commands, but the important point is that, whatever form of intercommunication is used, it should be consistent. Similar commands should not do different things in different parts of the program. If they do, the user has to read each menu carefully before each choice is made and 'reflexes' cannot be built up.

As our program stands, it is poor in this respect: the greeting message is terminated by hitting the space bar; the options menu is terminated automatically by hitting any of the number keys from 1 to 9; and the data entry in ADDREC is terminated (for each field) by hitting the RETURN key. This kind of inconsistency may be acceptable in a 'home-brew' program, but should be considered unacceptable in commercial software.

User image refers to the way the user perceives the operation of the program. It is considerably influenced by the quality of the user interface. Most of the operations going on inside the computer are completely hidden from the computer operator. The only way the operator can form an idea of what's going on in the program is from the visual input he receives from the screen in response to the inputs from the keyboard. The user image we would want from our address book program would be that of an actual, physical address book. Similarly, the user image desirable from a word processing program would be that of a piece of 'paper' (on the screen) upon which we type. In this case, ideally, bold type would appear bold on the screen, underlined type would be underlined, and justified type (type with a straight right margin) would be justified on the screen.

A perfect user image is seldom possible — no real address book expects you to 'PRESS 1' to find a name. Nevertheless, a good user image involves well designed screen layouts and a consistent pattern to the operations. Prompts should always appear in the same position on the screen (some well known word processors, for example, display some prompts on the top line of the screen and some on the bottom line, apparently at random). A program with a good user image will also inform the user at any time where he is in the program. If you are in the ADDREC mode, there should be a message always visible to tell you this. If you have just entered a field (to add to a new record), there should be a message to say, for example, HIT RETURN IF ENTRY IS CORRECT, ELSE HIT ESCAPE (which brings us to the important subject of error recovery and reporting, which we will come to later).

Ideally, all formatting should appear on the screen, so that, for example, the record displayed on the screen will be of the same format as a record printed out by the printer. Many commercially available pieces of software incorporate 'help menus' which will tell you what to do next if you're not sure.

The user image of a program is best when it is concrete — a piece of typing paper or an index card — rather than abstract with 'sub-files', 'buffers' and so on. Many commercial database programs suffer in this respect; the user has constantly to keep in mind that certain pieces of information are in sub-files or temporary, hidden fields. These factors tend to make the use of such a program more of an intellectual burden.

Error recovery is also an important subject. What happens, for example, if you have just entered someone's name but realise that you made a typing error? Will you have to go ahead and then call MODREC to correct it, or will the program give you the option to 'quit' before you go any further? Most versions of BASIC will report errors in the entering of a program, either when the erroneous line is entered, or when the program is run. However, this is not part of the 'user interface'. BASIC does include a number of messages that re-prompt the user for a correct entry if an incorrect one is made (for example, the REDO prompt if an unsuitable entry is made in response to an INPUT statement).

Handling errors has two facets — error reporting and error recovery. One well known word processing program, for example, has good error reporting but poor error recovery; if you create a long document and try to save it on a disk that is already nearly full, the program gives you the helpful message DISK SPACE EXHAUSTED. It does not, unfortunately, allow the user to recover from this error — a new disk cannot be formatted without first destroying the text that you may have spent hours typing in!

Any operation performed by the user that could result in the loss of data (MODREC, for example) should always be queried before execution. Messages such as THIS WILL DESTROY THE RECORD. ARE YOU SURE? (Y/N) should always be provided. In a word processor, a similar message would be: THE 'SAVE' COMMAND WILL NOT KEEP A BACKUP OF THE OLD DOCUMENT. IS THIS OK? (Y/N).

Error handling (trapping and reporting) should be considered in the design of a program wherever there is a possibility of wrong data input, wrong menu choice, wrong commands and whenever data is to be modified or saved, especially if the save involves writing over old data.

You must pay attention to security — what happens to the program or data if there is a fatal error (such as a power failure). The program designer should consider how much data it is possible to lose and devise methods of recovering as much as possible or making the remaining data usable. One rather sophisticated word processor incorporates a program called RECOVER so that if there is a catastrophic error (a power failure, for example, or switching off the computer before saving the document), almost nothing is lost. Such advanced programming techniques, sadly, are beyond the scope of this course. The point is, though, to make your programs as secure as possible by anticipating all possible fatal errors beforehand (that can be reasonably dealt with), and writing routines designed to cope with them.

Adaptability, the ease with which the program can be customised, is also important. We have touched on this topic a number of times already. At the simplest level, always leave plenty of room between line numbers (in BASIC) and incorporate plenty of empty REMs that can be filled later with statements and GOSUBs if necessary. When creating arrays, at least one redundant array should be built in to allow for future expansion. It is a cardinal rule of program writing that future requirements cannot be anticipated. The only thing certain is that a good program can always be made better, and making it better is likely to mean writing more code.

## Basic Flavours

ZX81
SPECTRUM

See the program on page 318 for the Spectrum version of lines 1300 to 1370, and the Basic Flavours box for the ZX81 version. On the Spectrum, lines 3750 to 3780 may cause problems because keys repeat if held down for more than a fraction of a second. The Spectrum handbook recommends that this type of INKEY$ loop contain an extra line to avoid this:

```
3755 IF INKEY$ <>"" THEN GOTO 3755
```

Sinclair BASIC supports VAL(A$), but will crash the program if the first character of A$ is non-numeric; in this program the problem can be avoided by:

```
3790 LET CHOI=CODE A$-48
```

but this is not a complete solution — it works only when A$ is a single character (as it must be in the program). Sinclair BASIC does not support ON... GOSUB, but it does allow you to write GOSUB (numeric expression) as well as simply GOSUB (line number); line 4010, therefore, may be replaced by

4010 GOSUB (290+CHOI*20)

Sinclair BASIC does not support RENUMBER

**MERGE**

This is not supported by the Oric-1, the Commodore 64 and Vic-20, the Lynx, the Dragon 32, and the BBC Micro. Often, however, there will be a way of simulating the MERGE command that a user group or specialist publication may reveal. The Lynx has the command APPEND, which allows you to LOAD one program onto the end of the program in memory: this can replace MERGE provided that the first line number of the progam on tape is higher than the last line number of the program in memory.

**BBC**

The string variable name MODFLD$ is not acceptable in BBC BASIC because it begins with the keyword MOD. This name should be changed, therefore, to MDFLD$.

**INKEY$**

See 'Basic Flavours' page 175.

**OPEN CLOSE**

See 'Basic Flavours' page 319.

**PRINT CHR$(12)**

Replace this by CLS, except on the Commodore 64 and Vic-20 — on these machines type PRINT"shiftkey+CLRkey", which should cause a reverse-field heart or capital S to appear between the quotes.

**ON... GOSUB**

In line 4010, ON...GOSUB will cause control to pass to non-existent lines — 310, 330, 350, etc. — which will cause the program to crash. These lines will later be expanded to contain the menu execution subroutines; in the meantime, enter these lines to your program as 'dummies', like this:

```
310 RETURN
330 RETURN
```

and so on.

**STR$**

The Lynx does not support this command, so replace line 9080 by:

```
9075 N=SIZE
9077 GOSUB 9500
9080 NDXFLD$(SIZE)=N$
```

and insert this subroutine:

```
9500 REM To convert N into N$ where N is an
integer
9510 LET N$=" "
9520 IF N<0 THEN LET N$=""
9530 N=ABS(N)
9540 X=10
9550 I=1
9560 FOR K=1 TO 8
9570 I=I*X
9580 R=K
9590 If I>N THEN K=8
9600 NEXT K
9610 FOR K=1 TO R
9620 I=I/X
9630 Z=INT(N/I)
9640 LET N=N-Z*I
9650 N$=N$+CHR$(48+INT(Z))
9660 NEXT K
9670 RETURN
```

```
10 REM  'MAINPG'
20 REM  *INITIL*
30 GOSUB 1000
40 REM  *GREETS*
50 GOSUB 3000
60 REM  *CHOOSE*
70 GOSUB 3500
80 REM  *EXECUT*
90 GOSUB 4000
100 END
1000 REM  *INIT* SUBROUTINE
1010 GOSUB 1100: REM  *CREARR* (CREATE ARRAYS) SUBROUTINE
1020 GOSUB 1300: REM  *RDINFL* (READ IN FILES) SUBROUTINE
1030 GOSUB 1320: REM  *SETFLG* (SET FLAGS) SUBROUTINE
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM *CREARR* (CREATE ARRAYS) SUBROUTINE
1110 DIM NAMFLD$(50)
1120 DIM MODFLD$(50)
1130 DIM TWNFLD$(50)
1140 DIM CNTFLD$(50)
1150 DIM TELFLD$(50)
1160 DIM NDXFLD$(50)
1170 REM
1180 REM
1190 REM
1200 REM
1210 LET SIZE = 0
1220 LET RMOD = 0
1230 LET SVED = 0
1240 LET CURR = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 RETURN
1300 REM *RDINFL* SUBROUTINE -- SEE 'FLAVOURS' BOX
1310 ON ERROR GOTO 1370
1320 OPEN "I",#1,"ADBK.DAT"
1330 FOR L = 1 TO 50
1340 INPUT #1 NAMFLD$(L),STRFLD$(L),TWNFLD$(L),CNTFLD$(L),TELFLD$(L)
1350 NEXT L
1360 CLOSE #1
1370 RETURN
1380 REM  DUMMY *SETFLG* ROUTINE
1390 RETURN
3000 REM  *GREET* SUBROUTINE
3010 PRINT
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT TAB(12);"*WELCOME TO THE*"
3060 PRINT TAB(9);"*HOME COMPUTER COURSE*"
3070 PRINT TAB(6);"*COMPUTERISED ADDRESS BOOK*"
3080 PRINT
3090 PRINT TAB(5);"(PRESS SPACE-BAR TO CONTINUE)"
3100 FOR L = 1 TO 1
3110 IF INKEY$ <> " " THEN L = 0
3120 NEXT L
3130 PRINT CHR$(12)
3140 RETURN
3500 REM  *CHOOSE* SUBROUTINE
3510 REM
3520 REM  'CHMENU'
3530 PRINT CHR$(12)
3540 PRINT "SELECT ONE OF THE FOLLOWING
3550 PRINT
3560 PRINT
3570 PRINT
3580 PRINT "1. FIND RECORD (FROM NAME)"
3590 PRINT "2. FIND NAMES (FROM INCOMPLETE NAME)"
3600 PRINT "3. FIND RECORDS (FROM TOWN)"
3610 PRINT "4. FIND RECORD (FROM INITIAL)"
3620 PRINT "5. LIST ALL RECORDS"
3630 PRINT "6. ADD NEW RECORD"
3640 PRINT "7. CHANGE RECORD"
3650 PRINT "8. DELETE RECORD"
3660 PRINT "9. EXIT & SAVE"
3670 PRINT
3680 PRINT
3690 REM  'INCHOI'
3700 REM
3710 LET L = 0
3720 LET I = 0
3730 FOR L = 1 TO 1
3740 PRINT "ENTER CHOICE (1 - 9)"
3750 FOR I = 1 TO 1
3760 LET A$ = INKEY$
3770 IF A$ = "" THEN I = 0
3780 NEXT I
3790 LET CHOI = VAL(A$)
3800 IF CHOI <1 THEN L = 0
3810 IF CHOI >9 THEN L = 0
3820 NEXT L
3830 RETURN
4000 REM  *EXECUT* SUBROUTINE -- SEE 'FLAVOURS' BOX
4010 ON CHOI GOSUB 310,330,350,370,390,410,430,450,470
4020 RETURN
9000 REM *ADDREC* SUBROUTINE
9010 PRINT CHR$(12)
9020 INPUT "ENTER NAME";NAMFLD$(SIZE)
9030 INPUT "ENTER STREET";STRFLD$(SIZE)
9040 INPUT "ENTER TOWN";TWNFLD$(SIZE)
9050 INPUT "ENTER COUNTY";CNTFLD$(SIZE)
9060 INPUT "ENTER TELEPHONE NUMBER";TELFLD$(SIZE)
9070 LET RMOD = 1
9080 LET NDXFLD$(SIZE) = STR$(SIZE)
9090 GOSUB *MODNAM*
9100 RETURN
```

TONY LODGE

# Sound Qualities

## Sound generation on the BBC Model B

Exceptional sound facilities and comprehensive BASIC commands to control them place the BBC Micro among the best computers available for the home user who is particularly interested in sound. Three independent square wave oscillators, eight types of noise and four independent ADSR (Attack, Decay, Sustain and Release) and pitch envelopes are supplied as standard. This means that music sequences can be constructed consisting of up to three voices in harmony and special commands ensure that notes selected to form a chord are played at exactly the same time.

### Sound Creation

In its simplest form, sound can be created by the use of the SOUND command:

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

In this case:

C=Channel or oscillator number (0-3)
V=Volume
P=Pitch of note
D=Duration or length of note

Any one of three oscillators (1, 2 and 3) may be selected to play a note and 0 selects noise. Volume takes a value between 0 (off) and −15 (loud). Pitch is defined in steps of a quarter of a semitone between 0 (A# at 116.5Hz) and 253 (D at 4698.64Hz) giving a range of five and a half octaves. Middle C is set by the value of 53. If the noise channel has been specified, there are eight types available, which set up the following pitch numbers:

| Number | Noise Type |
|---|---|
| 0 | High pitch 'tremolo' |
| 1 | Medium pitch 'tremolo' |
| 2 | Low pitch 'tremolo' |
| 3 | 'Tremolo'. Pitch varies with the pitch of channel 1 |
| 4 | High pitch |
| 5 | Medium pitch |
| 6 | Low pitch |
| 7 | Pitch varies with the pitch of channel 1 |

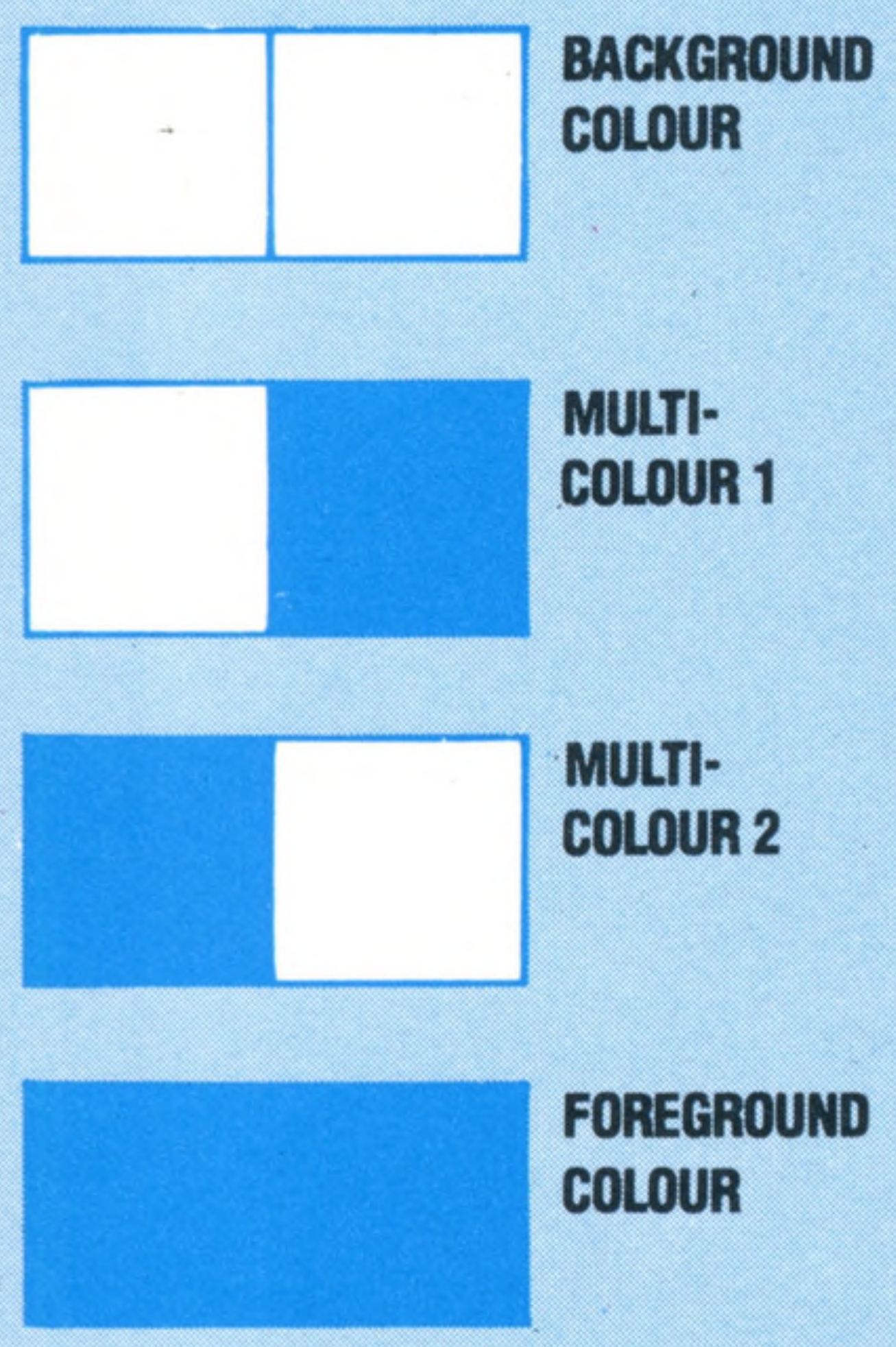
Finally, duration of note is controlled in steps of 20th of a second between 1 and 255, which give a maximum note length of 12.75 seconds. A command to play A above middle C on channel 1 at a medium volume of −7 for half a second is constructed as:

```
SOUND 1, -7, 89, 10
```

If the computer receives a second SOUND command before the first is completed the note is placed in the channel queue.

# Light Construction

## Commodore 64 graphics capabilities



The Commodore 64 computer has a great number of possibilities for the graphics programmer. The main drawback is that, in common with the Vic-20, there is only an extremely limited set of BASIC commands provided as standard. POKEing and PEEKing inside the computer's memory allow the programmer access to all the machine's features but some of the procedures required can be quite confusing. Again, as with the Vic-20, there are several ways out of this maze. These take the form of a number of independently produced software packages that make the creation of sprites or user-defined characters much simpler. In addition to these, Commodore produce their own plug-in cartridge, called 'Simon's BASIC'.

The Commodore 64 has the standard upper and lower case character sets available in normal or inverse display modes. Also available are the special graphics characters that were first developed for the PET and which are also used on the Vic-20. The screen display consists of 40 columns and 25 rows and characters are positioned on it either by PRINTing them to the screen in any chosen colour or by POKEing the relevant codes into the screen and colour memories detailed in the user manual. The special Commodore graphics characters form an extremely flexible medium in which to put together attractive displays in low resolution. More than 60 special characters may be used in this way. Again, each of these special characters may be displayed in normal or inverse mode, giving the user well over 120 options from which to choose when designing a figure.

If the exact character cannot be found then it is possible to define new characters for use in the display. This is, unfortunately, not very easy to do. To define a character the programmer must first copy all the normal characters that will be required from ROM memory to RAM, before POKEing the necessary numbers into the series of locations that are to hold the new character.

In addition to the simple functions outlined, the capabilities of the SOUND command can be extended by changing its format to:

```
SOUND &HSFC,V,P,D
```

In this case the ampersand sign & instructs the computer to treat HSFC as a four digit hexadecimal number (see page 179). However, to understand how to use the modified command it is only necessary to analyse the function of each digit. Only three of these are significant as C is merely the usual channel number described in SOUND.

H can either be unspecified (0) or set at 1. H = 1 allows the previous note on the same channel to continue to completion. Otherwise, if a note has been constructed with a long release phase (dying away slowly) the computer will mistakenly assume the note is complete while it is still sounding and start the next note abruptly in the middle of it. H set at 1 forces the computer to wait for completion. If the H parameter is used the remaining parts of the SOUND & command, apart from the channel number, are ignored.

S allows the user to specify a number of notes to be played at the same time on different channels, creating a chord. S = 0 leaves the command in its normal state. S is set to 1 if it is required that a note on one other channel is played at the same time. If set to 2, notes on both the other channels will play. In effect, when the computer encounters an S value it holds the relevant note back until it can account for the associated one or two other notes in the chord that are indicated by the same S number on the remaining channels. It then plays all the specified notes together.

F is set to either 0 or 1. Zero has no effect but 1 causes the computer to discard all notes that are waiting in the channel queue to be played, stops the current note playing and plays the note contained in its own command immediately.

SOUND & is best illustrated with an example. This program plays the first line of 'Happy birthday to you' in the key G# (below middle C) as:

```
G#G#A#G#C#D#
```

It also includes a three part major chord for the final D#

```
10 SOUND 1,-7,37,10:REM *1ST G#*
20 SOUND 1,-7,37,10:REM *2ND SHORT G#*
30 FOR I=1 TO 3: READ N
40 SOUND 1,-7, N,10: NEXT I: REM *A#,G#,C#*
50 SOUND &201, -7,37,15: REM *G#*
60 SOUND &202, -7,53,15: REM *C*
70 SOUND &203,-7,65,15: REM *D#*
80 DATA 45,37,57
90 END
```

The sound capabilities given by the command ENVELOPE will be discussed later in the course.

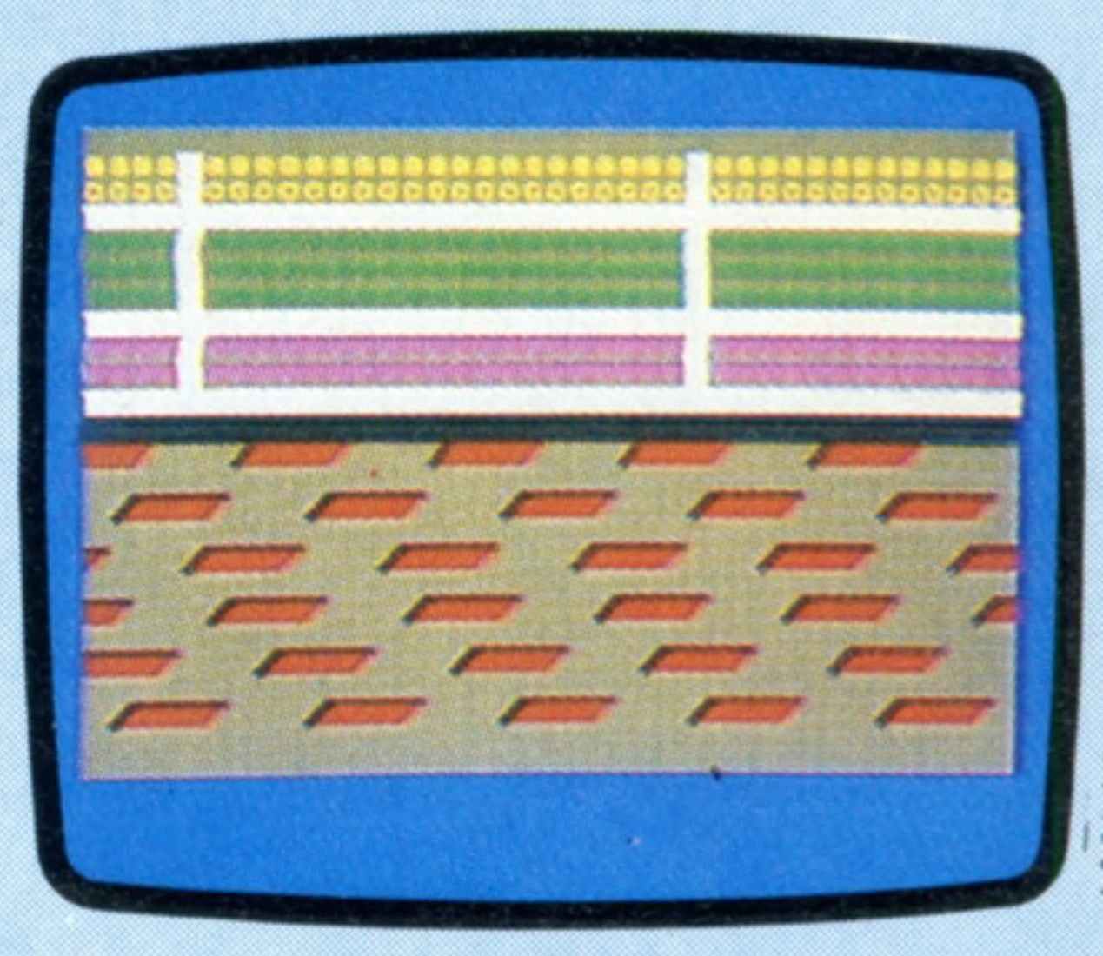
Each character cell is made up of eight by eight rows of pixel dots that are represented in binary form as groups of 1s and 0s. Normally 1 is interpreted as a pixel set to the foreground or character colour, and 0 interpreted as a pixel set to the background or screen colour. There is a facility on the Commodore 64 to allow up to four colours to be represented within any character cell. This is known as multi-colour mode. When the computer is switched into multi-colour mode it uses two bits to represent the colour of each pixel.

There are four possible combinations for any pair of bits and this fact is used to represent four different colours inside the character cell. Each character cell on the screen can be set either to be interpreted in the normal way or as a multi-colour character, but in the latter case the choice of colours available is reduced from 16 to 8. Further drawbacks are that the two multi-colours must be the same for all the characters on the screen, and that when in multi-colour mode the horizontal resolution is reduced by a factor of two.

There are no high resolution graphics commands in Commodore BASIC. However, it is possible to create high resolution displays by using the technique known as 'bit-mapping'. The Commodore 64's screen display is made up of 64,000 pixels. Bit-mapping works by allowing the programmer to turn each pixel on or off individually. This is a fairly complex procedure for the average user to undertake and the use of BASIC in this way makes for an extremely slow build-up of the high resolution picture. In practice, there are two alternatives open to those who wish to exploit high resolution graphics: the first is to purchase the Simon's BASIC cartridge from Commodore and the second is to learn to program in machine code. Standard high resolution mode on the Commodore 64 divides the screen into 200 rows of 320 pixels. Multi-colour mode is also available in high resolution, allowing up to four different colours within any eight by eight block.

Here is a short program that uses the Commodore character set to create a supermarket scene. Later in the Sound and Light course we will discuss sprites and Simon's BASIC in more detail.

**Step One**
This static supermarket scene was created using low resolution graphics. In a future instalment of the Sound And Light course we'll add a moving shopper with a trolley, using sprite graphics



IAN McKINNELL

```
3000 REM ** SUPERMARKET **
3010 PRINT"[illegible]"
3020 POKE53280,6:POKE53281,12
3030 PRINT"[illegible]●●●●●[illegible] [illegible]●●●●●●●●●●●●●●●●●●●●●●●●[illegible] [illegible]●●●●●●●●●●●●●●"
3040 PRINT"[illegible]OOOO[illegible] [illegible]OOOOOOOOOOOOOOOOOOOOOOOO[illegible] [illegible]OOOOOOOOOOOOO"
3050 PRINT"[illegible]                                        [illegible]"
3060 PRINT"[illegible]♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
3070 PRINT"[illegible]♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
3080 PRINT"[illegible]♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦[illegible] [illegible]♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
3090 PRINT"[illegible]                                        [illegible]"
3100 PRINT"[illegible]♥♥♥♥[illegible] [illegible]♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥[illegible] [illegible]♥♥♥♥♥♥♥♥♥♥♥♥♥♥"
3110 PRINT"[illegible]♥♥♥♥[illegible] [illegible]♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥[illegible] [illegible]♥♥♥♥♥♥♥♥♥♥♥♥♥♥"
3120 PRINT"[illegible]                                        [illegible]"
3130 PRINT"[illegible]                                        [illegible]"
3140 PRINT"[illegible]  ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤"
3150 PRINT
3160 PRINT" [illegible]◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤"
3170 PRINT
3180 PRINT"◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤  "
3190 PRINT
3200 PRINT"[illegible] ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤ '
3210 PRINT
3220 PRINT"[illegible]  ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤"
3230 PRINT
3240 PRINT" ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤   ◤"
3250 FORI=1063TO2023STEP40
3260 POKEI,160:POKE54272+I,6:NEXT
3270 GOTO3270
```

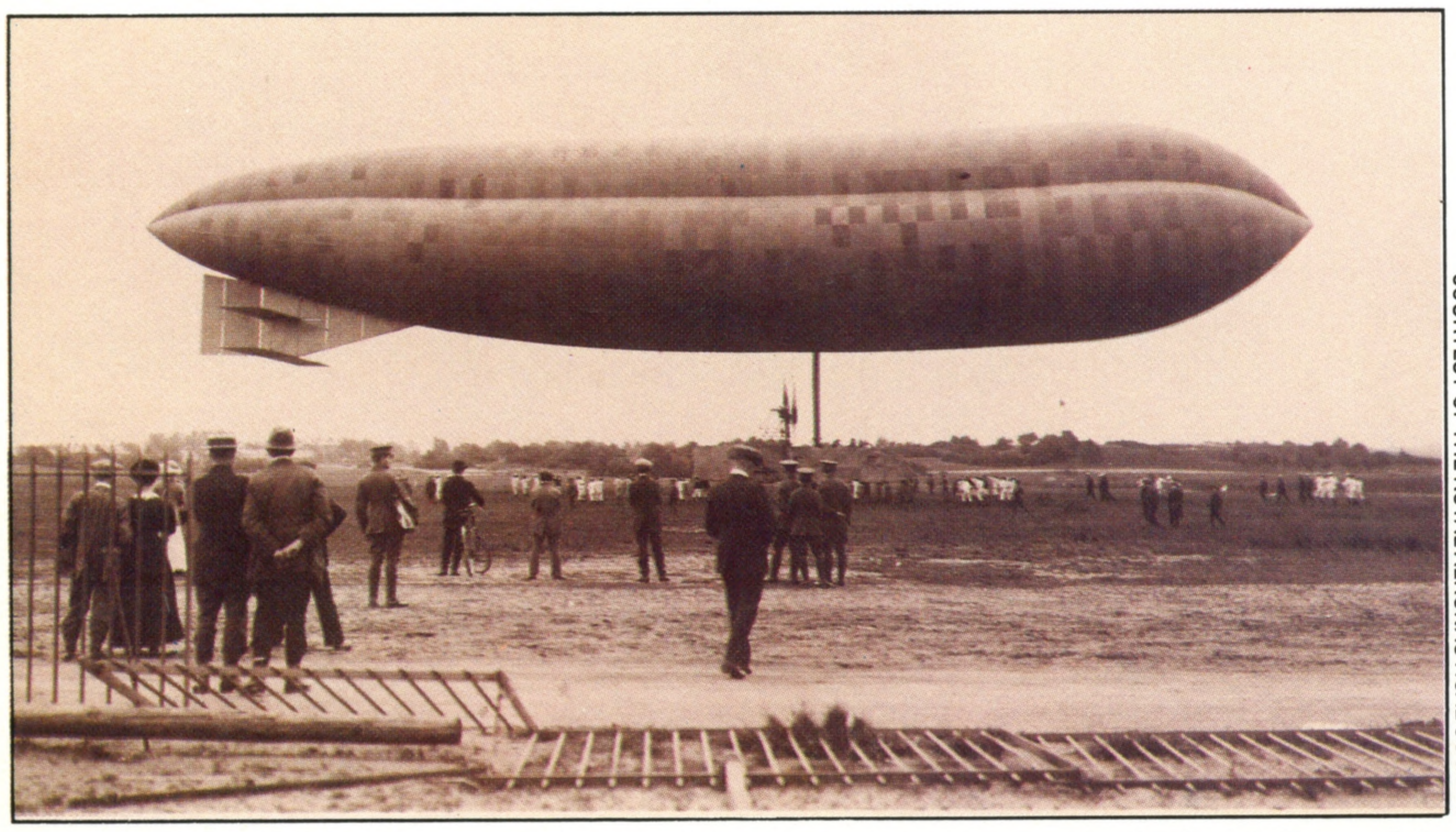
# Leonardo Torres

## His interests ranged from airships to cable cars, but he made a number of important contributions to the development of computing

Leonardo Torres y Quevedo, the scientist who first applied floating-point arithmetic to computers, was born in the Spanish town of Santa Cruz on 28 December 1852. He was educated at the Institute of Bilbao and the School of Engineers in Madrid, before embarking on a career as an engineer and inventor.

His powers of invention reached their height in the later part of his life. He designed the Niagara Transport Bridge and cable car, which are still in use today at Niagara Falls, and a semi-rigid airship that was manufactured during the First World War. But fundamentally, Torres was a child of his times and his main interest was in electromechanical devices. In 1906, he demonstrated a radio-controlled model boat in the harbour of Bilbao before the King of Spain, and in 1911 he invented the first automatic chess player. The machine used electromagnets beneath the table to move the pieces, and was programmed to win a simple game against a human opponent.

**Floating Points**
Like many of his contemporaries, Torres was a true polymath. His interests ranged from the design and construction of mechanical devices such as the airship shown here, through electromechanical calculating machines, into the realm of pure mathematics


COURTESY OF THE ROYAL AERONAUTICAL SOCIETY

Torres' interest in automata derived from his experience of the assembly lines in the industrial plants of early 20th-century Europe; and all through his life he sought to separate the types of problem that required human thought from those that could be done automatically.

In 1914, he published a paper showing the feasibility of building Babbage's Analytical Engine (see page 220) using electromechanical techniques, and it was in this paper that he first suggested the use of floating-point arithmetic in any future computer. In 1920, he constructed an electromechanical calculator that used an adapted typewriter for entering the numbers and printed the results automatically. The typewriter was connected by telephone wires to the calculator, and Torres foresaw the possibility of many terminals attached to a central calculator (or processing unit).

He was honoured by the French Academy of Sciences for his work, and later became President of his country's own Academy of Sciences. Torres died in Madrid on 18 December 1936.

## Floating-Point Arithmetic

A cash register displays the totals in pounds and pence (£12.25, for example), and in such a machine there is only a need for two places after the decimal point. But in a computer, greater accuracy is often required and the number of decimal places is allowed to vary or 'float' according to the needs of the problem. This is known as 'floating-point arithmetic'.

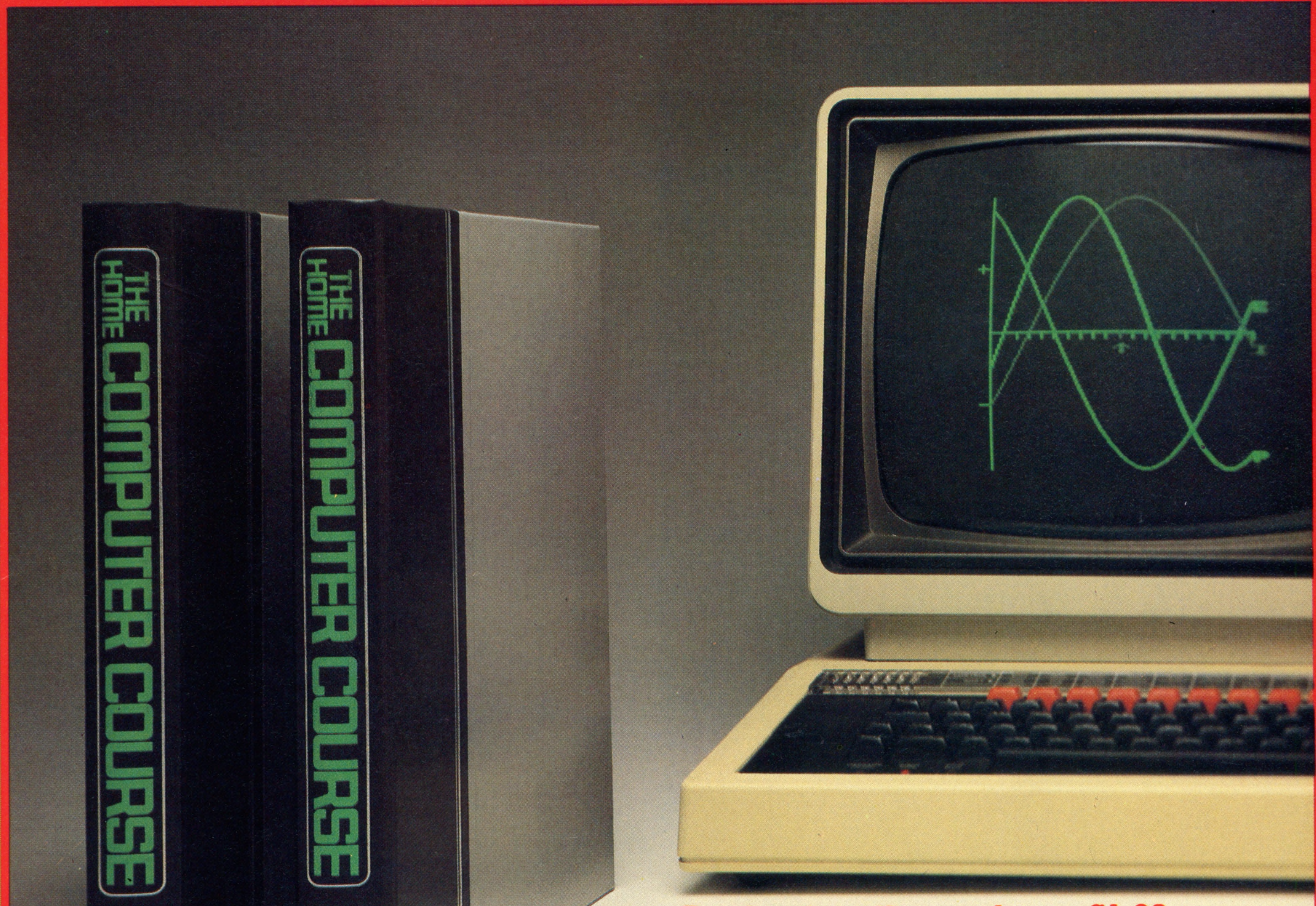
Any number can be written in a variety of ways. For example, 0.8752 metres can be expressed as 875.2 millimetres, or 0.8752 x 1000 millimetres, or simply $0.8752 \times 10^3$. This last method lends itself to an economical method of encoding for a computer. If a computer has only allocated six digit spaces to represent each number (and for the sake of clarity the decimal system is used instead of binary), then the above number can be stored as 875203: where the last two digits on the right-hand side are called the 'index' and represent the power of 10 (in this case 3) and the leading four digits are called the 'mantissa'. To give another example for this computer: the number 418302 represents $0.4183 \times 10^2$, or 41.83.

The mantissa and index are usually 'normalised' to remove any leading zeros from the mantissa. For example, the number 41.83 could be written as 004104, but would be normalised to 418302 — thus including more significant digits in the mantissa.

The index/mantissa form of floating-point arithmetic has the advantage that a wide range of numbers can be represented. For the computer suggested above, which allocates only two digits for the index, a number as large as $0.9999 \times 10^{99}$ can be dealt with — or a number so small that there are 98 zeros after the decimal point before any non-zero digit is encountered.

However, the accuracy of this system remains limited to the digits allocated to the mantissa. Consequently, some numbers can be only approximately represented, and great care and ingenuity must be put into the techniques of arithmetic programming to stop errors arising. This is the reason why (1/3)*3 will give 0.9999999 on some computers, rather than the true answer of 1.

# 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

# 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