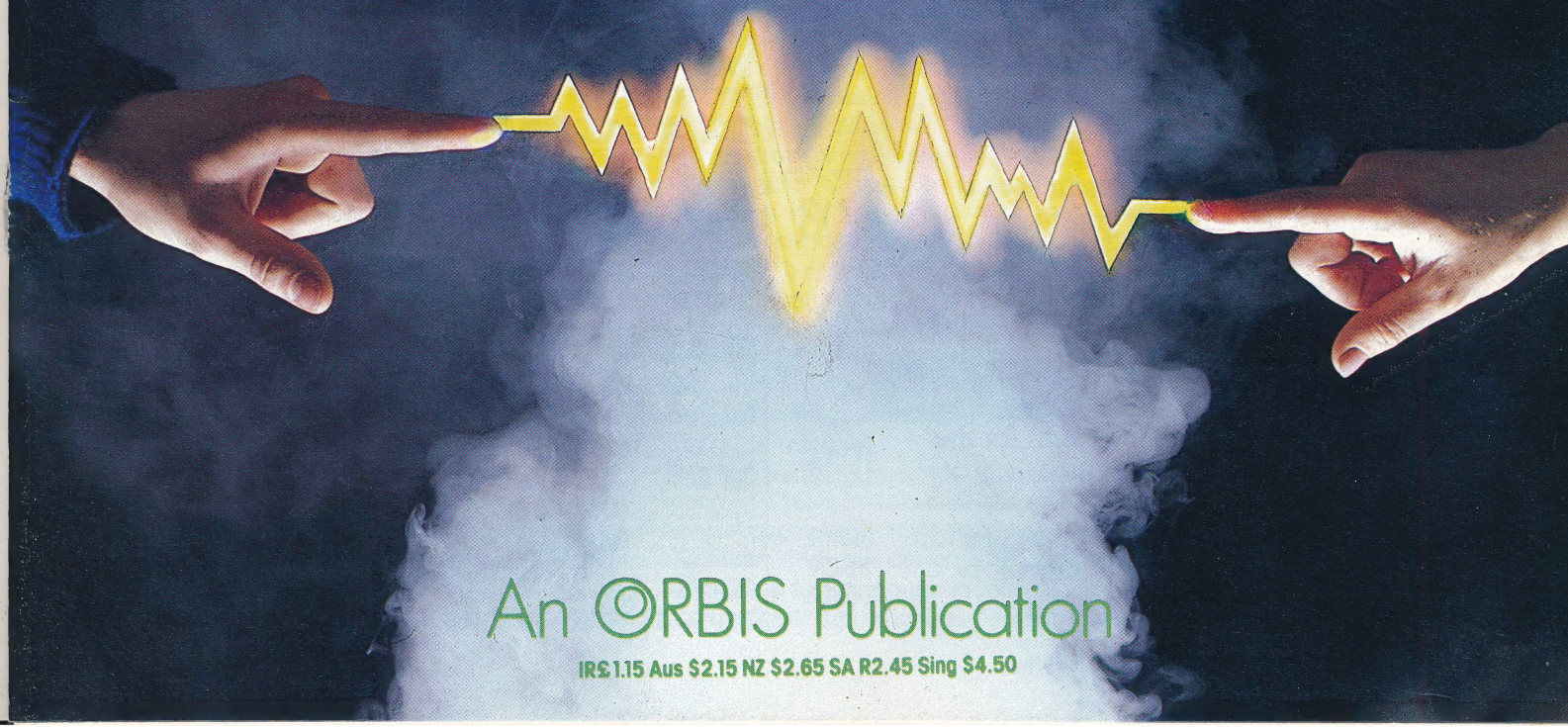
ISSN 0265-2919

90p 61

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO





An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

MIND OVER MATTER Computers controlled by thought are no longer the preserve of science-fiction writers. We look at the principles behind this recent technological development — 1201

HARDWARE

AMSTRAD SPIN-OFF The Amstrad DDI-1 disk drive, supplied with CP/M and Dr LOGO, upgrades the Amstrad CPC 464 to a powerful and economic system — 1209

SOFTWARE

THE MIGHTY PEN In our penultimate examination of database systems for home micros, we discuss the MicroPen package — 1207

CIPHER CHASE Impossible Mission, from Epyx, is a platform game with excellent graphics and a 'race against time' scenario — 1220

COMPUTER SCIENCE

DYNAMIC STRUCTURES Our tutorial course now considers PASCAL's use of file structures — 1204

JARGON

OUTPUT DEVICE TO PAGE PRINTER A weekly glossary of computing terms — 1215

PROGRAMMING PROJECTS

BATTEN DOWN THE HATCHES We develop the routines that deal with the remaining four major contingencies in our simulation game — 1212

MACHINE CODE

SPLITTING IMAGE We conclude our investigation of the VIC-II's graphics facilities by providing a program that allows the Commodore 64's screen to display high- and low-resolution graphics simultaneously — 1218

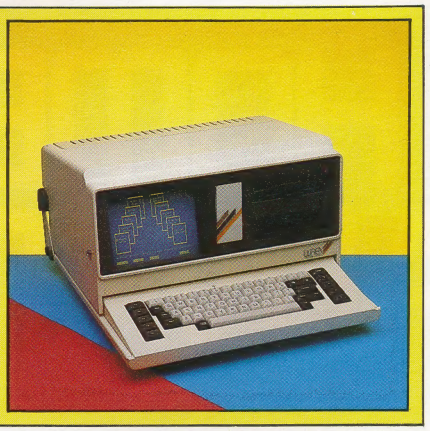
WORKSHOP

SINCLAIR'S SINEWS A program for the Spectrum to control our robot arm — 1216

REFERENCE CARD The first part of a full analysis of the Commodore 64's memory map — INSIDE BACK COVER

Next Week

- **In our Commodore 64 operating system series we turn our attention to the SID chip, which allows synthesiser quality music to be generated using sophisticated software.**
- **We discuss some mathematical methods for controlling our Workshop robot arm.**
- **There is a possibility, in our New World simulation game, that the crew will mutiny. We show how this contingency is programmed.**



QUIZ

1) What does GSR stand for, and what is the principle on which it operates?
2) Apart from both packages being bundled with the Amstrad disk drive, what do CP/M and Dr LOGO have in common?
3) How does the VIC-II chip obtain sufficient time from the processor to enable it to access data?
4) Why do we need to set up a 256-byte table in the Spectrum's memory in order to intercept Z80 interrupts?

Answers To Last Week's Quiz

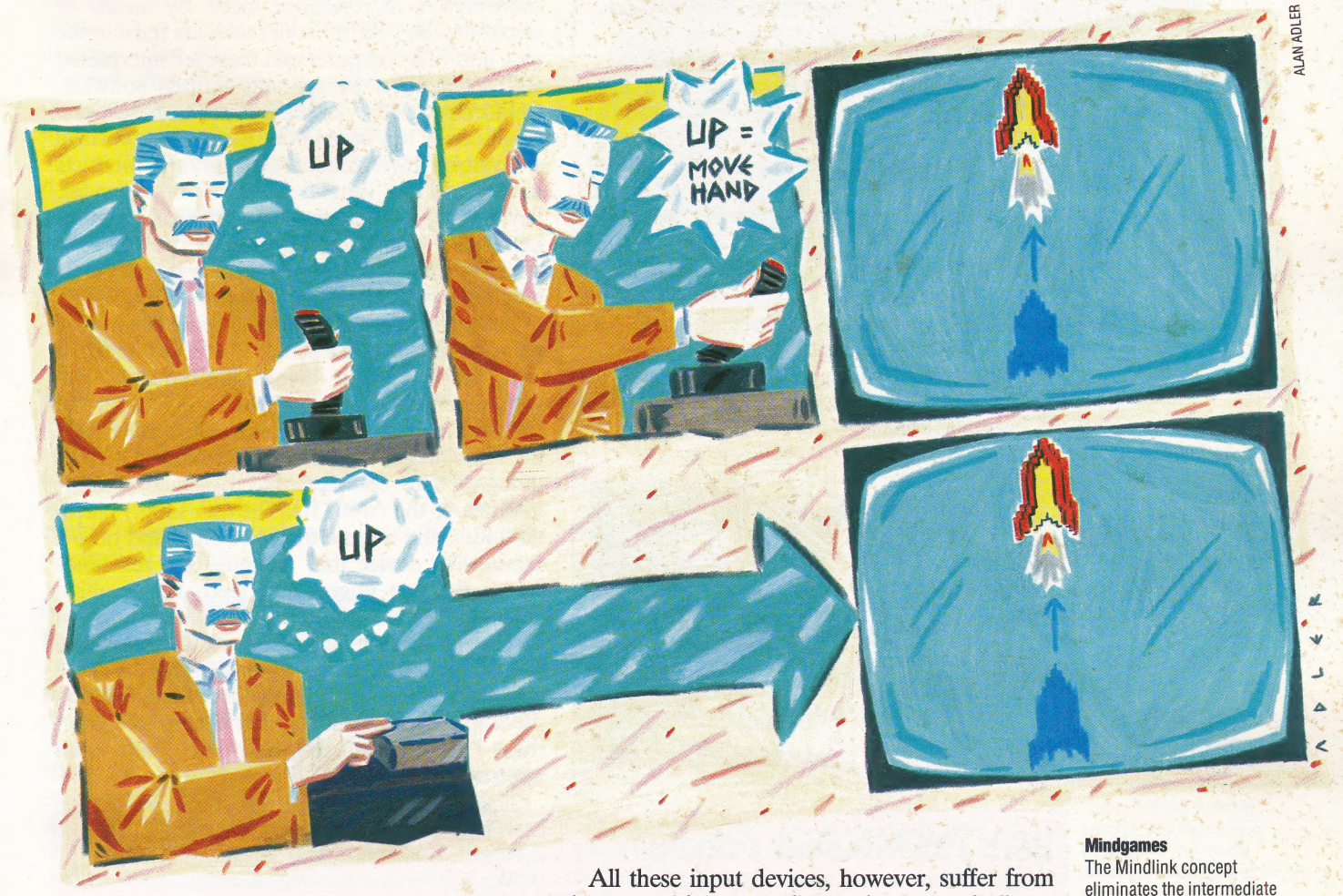
1) The Commodore 64's VIC chip uses processor time to support the screen, which interferes with the regular generation of interrupts to run the servo motors.
2) Because Prof provides no facilities for storing input data it cannot be used by the graphics input pad.
3) CD-ROM, standing for Compact Disc ROM, is a new technology whereby a computer is able to read data stored on a compact disc.
4) A standard optical disc can store around two gigabytes of information.

Editor Stephen Cooke; **Deputy Editor** Steve Colwill; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Steve Malone; **Sub Editor** Jon Kaye; **Contributors** Dr Andrew Bangham, Nick Walsh, Dr Antonia Jones, Martin Young, Surya, Steve Malone, David Mudd, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues 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 and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON–SMITH

MIND OVER MATTER



Mindgames
The Mindlink concept eliminates the intermediate mechanical stage of translating the player's impulses into signals meaningful to the computer. By allowing a more direct relationship between user and machine, Mindlink offers the possibility of friendlier software and faster input

The idea of controlling a computer by thought sounds like something out of a science-fiction movie, but not only is it feasible — it's already a fact. We study the phenomenon on which this technology is based and look at an input device that links the human nervous system to a computer.

From the days of the first computers the keyboard has been the primary input device, the means of getting information into a computer. It's probably the most efficient method of entering text and, to a lesser extent, figures — at least until reliable general-purpose voice-recognition systems are developed. But, good as the keyboard is for entering text, it's not necessarily the best device for entering other types of information. If you want to enter urgent directional data, as in many games, a joystick or trackball is preferable, and a mouse, light-pen or touch-sensitive screen is usually the best means of entering positional data, as when making menu selections. A bar-code reader is a convenient method of entering long serial or code numbers.

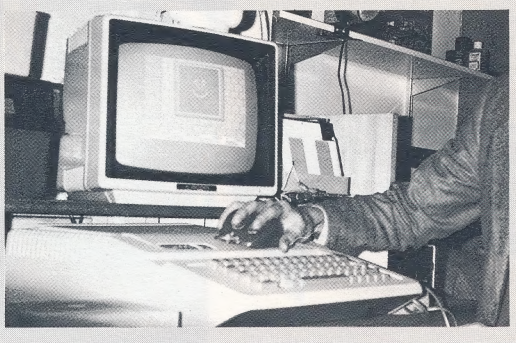
All these input devices, however, suffer from the same inherent flaw: they are indirect. Although a joystick is more convenient than a keyboard in game-playing (to move your spacecraft up, for example, you push the joystick handle forwards) it still represents an interruption, a step between what you want to happen in the computer and what actually happens. You think 'up', mentally translate this into 'joystick forward' and then physically push the joystick. For faster and more direct interaction between the user and the computer we need to eliminate this interim step: why not just think 'up' and have the computer respond directly to the thought?

Imagine being able to play a game of Defender just by thinking 'up', 'down', 'turn round', 'fire' and so on, or writing a letter by just thinking the words. The telepathic word processor may be some years away yet, but the thought-controlled Defender game is a reality today, made possible by a concept known as 'mindlink'.

In the mindlink concept, thought — or more accurately the physiological changes that result from changes in thought-patterns — is used to control an electronic device. This device serves as an interface to a computer in much the same way as a joystick or any other input device. Mindlink is

Mental Notes

A demonstration of a prototype GSR system running on an Apple computer is shown here. Similar systems have been developed for use with the IBM and Commodore PC's but development of effective software is likely to take some time. Roger Dilts, president of Behavioral Engineering and author of GSR software, is particularly interested in the possibility of combining Mindlink with NLP (Neuro-Linguistic Programming) — a branch of psychology that has been largely concerned with the study of learning. By using the GSR technique to monitor the user's mental state, software can keep a check on the degree of tension in the user and adjust the program flow accordingly. Roger Dilts believes that educational programs, for example, will be able to assess the student's emotional response to the material presented. If the presentation is too complex, the emotional tension caused will be picked up by the computer and enable the pace of the program to be changed or the subject matter to be simplified



based on what is known as the 'GSR phenomenon', whereby changes in a human's emotional state manifest themselves in the electrical conductivity of the skin. The user is connected to a resistance meter by means of electrodes. Signals from this meter are fed into the user port of a computer and these are interpreted and acted upon by specially written software. California-based company Behavioral Engineering has based both games and practical software on this technique, and among its products in the field is a simplified version of Defender. Several major computer companies are also known to have been working in this field for several years, notably Atari, but only recently have they met with any degree of success.

In the conventional Defender game you control a spacecraft orbiting a planet. The idea is to shoot alien ships without being shot yourself and without crashing into the planet below. In the Behavioral Engineering version, all you control is the ship's height: the difference is that you do it by thought! The company has designed a GSR interface to the Apple IIe. The user simply places the index and middle finger of one hand onto a mouse-like device that measures the resistance across the two fingers and feeds the resulting values into the computer. The software is designed so that an increase in current (a low resistance brought about by increased tension) results in the ship climbing while a decrease in resistance allows the ship to fall. The idea, of course, is to control these movements to align the ship with approaching targets.

GSR: The Mindlink Principle

The mindlink concept is based on a phenomenon known by three alternative names: the galvanic skin response (GSR), the psychogalvanic reflex (PGR) and the electrodermal reflex (EDR). We have used the term GSR throughout this article.

GSR refers to changes in the electrical conductivity of the skin that correspond to changes in the emotional state of the person. Experimentation has shown that the more tense a person is, the lower the electrical resistance of their skin. The best-known application of GSR is in polygraphs, the so-called 'lie-detectors'.

Although the GSR phenomenon has been observed in both humans and animals since the 19th century, very little is known about its cause. The original theory was that the sweat produced by excitement or anxiety simply acts as an electrolytic conductor, thus decreasing the resistance of the skin. More recent experiments have, however, cast doubt on this simplistic theory.

But, whatever the reason for the effect, GSR is known to be directly related to the degree of tension present in the sympathetic nervous system. This in turn is dependent on the central nervous system and thus on the brain — hence the possibility of controlling a computer by thought. Changes in the activity of the brain bring about changes in the state of the central nervous system; this produces a change in the state of the sympathetic nervous system that results in changes in skin resistance. And changing an electrical current is the basis of any form of input device



IAN DOBBIE

The term 'mindlink', believed to have been coined by Atari, is perhaps misleading, since the link is with the nervous system rather than the mind. This is, however, a grey area: if the two are interdependent, does it make sense to separate them? Most people find that they are able to exercise a fair degree of control after about 20 minutes, often without being aware of how they're doing it. Some people consciously tense and relax their body slightly, whereas others simply think 'up' or 'down' and let their nervous system do the rest.

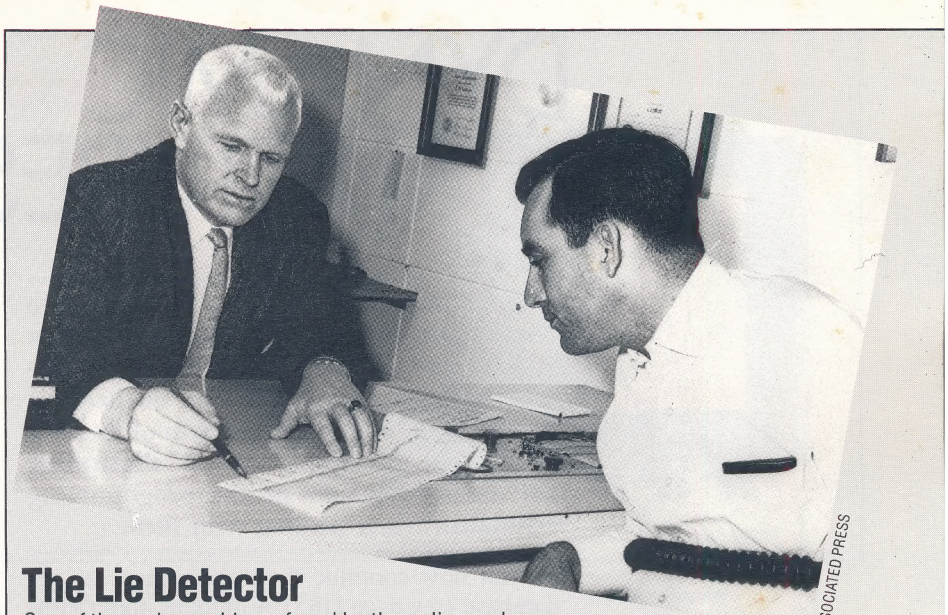
As well as games, mindlink has more practical applications. People who are completely paralysed still have the facility to consciously generate GSR effects, despite the fact that they have no control over their muscles. A GSR device has already been successfully linked to the Topo robot via an Apple computer, allowing a paralysed person to control the robot. Another interesting application is in space under zero gravity conditions. It can be extremely difficult to operate mechanical controls without gravity to counterbalance the force exerted. A GSR device could make an ideal substitute for many mechanical controls.

MOOD-SENSING SOFTWARE

Even more exotic applications include mood-sensing software. Educational software, particularly, could benefit from GSR feedback on the internal state of the user. The subject could wear a wristband containing the electrodes and allow the software to callibrate to their normal relaxed state. Then, if at any point in the program the GSR device registers a marked increase in tension, it can assume that the user is having difficulty and act accordingly. Even business software could detect stress in its user, and suggest a coffee break!

Just how rapidly GSR technology will develop depends partly on the introduction of more accurate and faster-responding devices and partly on our ability to refine our interpretation of the effects. The former problem is that a GSR response typically occurs two seconds after the event and takes between two and 10 seconds to fade. Present day devices ovecome this problem to an extent by measuring the rate of change of response rather than the strength of the response itself, but further work is necessary. The second problem centres on the crude nature of the deductions we're able to make from GSR feedback. We know that a rapid decrease in skin resistance indicates some kind of stress or tension, but we're not yet able to determine whether the tension is pleasurable or painful.

Despite the problems, however, GSR offers some exciting possibilities. And Atari is not alone in taking those possibilities seriously. Commodore is reputed to have offered Behavioral Engineering two million dollars for exclusive rights to its GSR device and software. Behavioral Engineering declined the offer.



ASSOCIATED PRESS

The Lie Detector

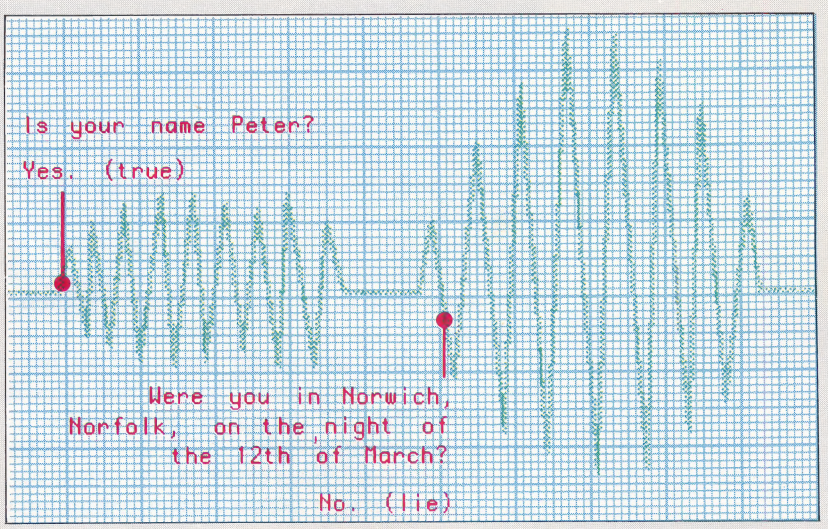
One of the major problems faced by the police and the judicial system in conducting an investigation and trial is knowing whether a suspect or witness is lying. Although a skilled observer will be able to detect minimal clues — changes in skin colouration and breathing, for instance — measurement of these factors has been neither defined not catalogued, nor are they admissible as evidence. For this reason, much research has been done on the development of an objective means of telling the difference between true and false statements. One of the results of this research was the polygraph, or lie-detector.

The polygraph is really nothing more than a resistance meter. The theory is that a subject will display significant increases in tension when telling a lie or when hearing keywords associated with the crime, and that this increased tension will show itself in diminished skin resistance.

The polygraph is the centre of much controversy. Much loved by investigative agencies, notably in the US, its critics argue that not enough is known about the GSR phenomenon for it to be interpreted reliably or consistently, and that different individuals will react in markedly different ways, regardless of guilt or innocence

Getting The Job
GSR is sometimes used to assess job applicants by using a lie detector to help assess a subject's response to questions during an interview. The American salesman in the picture is being shown the results of a test he has just taken

Little White Lies
A lie detector measures the changes in GSR provoked by the subject's emotional response to the question asked. In real life, however, a lie detector is extremely unlikely to give results as clear-cut as those shown here. Emotional responses vary according to the sensitivity of the subject to certain topics (for example, whether he finds the topic embarrassing) and are not simply dependent on the truth or falsehood of the replies



LIZ DIXON

DYNAMIC STRUCTURES

As we continue to create a 'database', we move from the use of functions within parameters to the development of an 'informal' algorithm. And since the heart of any database is the organisation of its files, we present several programs illustrating PASCAL's insistence on precise programming techniques when dealing with files.

We can now continue the development of our 'database' (see page 1185) by thinking about a few essential variables and an 'informal' algorithm.

```
VAR
   list  : RecordList;
   size  : Cardinal;
BEGIN
   size := 0; { active size of the list }
   { read the data into the list, updating size }
   { sort size items into the correct order }
   {print size items of the list }
END.
```

To translate this informal algorithm into something more tangible, we need to express the three main stages of processing as calls to procedures (with appropriate names) and to list any known data that each will require in the form of a parameter list. When this simple step is done, we can then write all the procedure blocks as a skeleton. For example, the last statement in the program will become:

```
Print (list, size)
```

The procedure will have all the information it needs if we pass it the list of data values and the number of elements, so the heading will not need any VARs in this case:

```
PROCEDURE Print (items : RecordList;
                 high : bounds);
```

We could go on to code the procedure in full, but it is best to leave it as a BEGIN . . . END 'stub' for now and deal with the other 'level 1' procedures. We must choose a name for each one, decide which data items need to be passed as parameters to which procedures, and determine whether any of these items need to be passed as 'VAR' (address) parameters.

All the data structuring methods we have discussed so far have been fixed in size. This is specified in the TYPE definitions and therefore is known at compile time. PASCAL provides advanced data structures which can vary their size (and even their type within certain constraints) during execution of a program. The size of an advanced structure is only limited by the physical memory or backing storage available, and the most familiar advanced structure is the sequential file.

Like an array, each element of a file may be of any type, simple or structured, with the exception that you can't have a file of files. Using our previous record type definition, we could add the following declarations:

```
TYPE
      FileType = FILE OF data;
VAR
      DataFile :FileType;
```

which would enable us to create and process a file containing an infinite number of components, each one of which was a record of a name, debt and any other fields we wish. Just as we say read (symbol), meaning read a single char from the file input, we can say: read (DataFile, item) or, write (DataFile, item) and handle a whole record structure as one data object. If these files are required only during the execution of the program, the only other requirement is to open them for reading from or writing to with the predefined procedures reset and rewrite, respectively.

Should you wish to make them permanent, which is more likely, then the file identifiers must be listed in the program header parameter list. In this case, for instance:

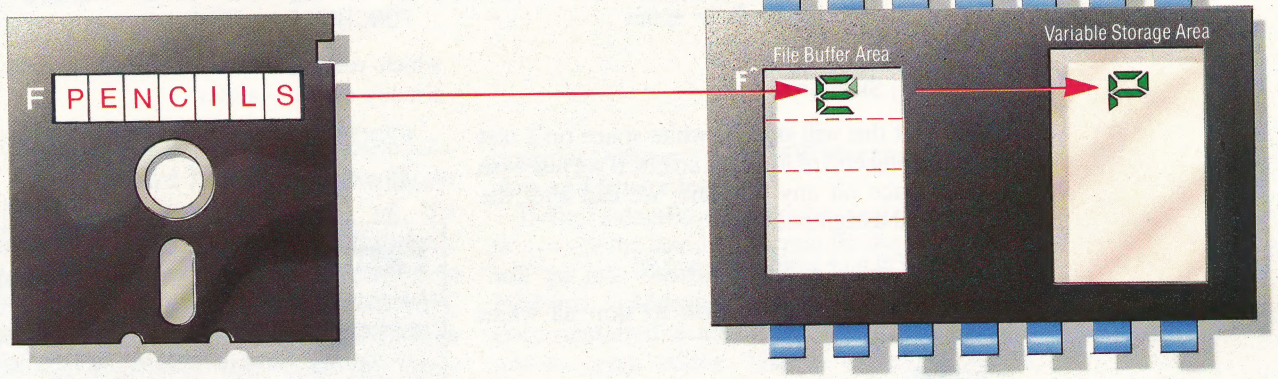
```
PROGRAM DataHandler (input, output, DataFile );
```

Every time we use read or write statements, we call PASCAL's predefined file I/O procedures. The only two files we know about so far, in fact, are the standard I/O devices on our computer. The file input is normally a keyboard and output will invariably be a VDU screen on microcomputer systems. By pretending that these devices are files, the handling of all I/O becomes extremely consistent in PASCAL. Remember that when we say write (N), omitting any file name, the value of N is printed in character form on the file output. Leaving a file name out in read or ReadLn statements defaults to the file input. Both these files are text files — that is to say, each 'record' of the file is a single char value.

TEXT FILES

Unlike other files, however, text files may have (as well as an end of file marker) special end of line markers embedded anywhere within them.These markers will vary with different operating systems, and may consist of a single control character, two characters (CR and LF for example) or possibly no characters at all, the length of each line being

Reading From A File



KEVIN JONES

stored instead.

In order to preserve portability, PASCAL provides the function EoLn (F) and the two predefined procedures ReadLn (F) and WriteLn (F) — all of which can only be used with files of type text. The EoF (F) function can, of course, be used with any file type. As the end of line marker may or may not be a single char, reading a char value when EoLn is true will return a Space character. Consequently, we should normally check with EoLn beforehand. As both input and output exist before and after any program is executed, they are permanently open, and do not need to be explicitly located or created.

With files other than input and output we must assign them to an external system name and then either open them for reading with a call to the reset procedure — i.e.reset (SomeFile)—or create a directory entry preparatory to writing to them by rewrite (AnotherFile). A general scheme for processing a text file is thus simply:

```
{ open the file }
WHILE { not reached the end of the source file } DO
   WHILE { not at the end of a line } DO
      { read a character }
      { process the character }
      { skip the end of line}
```

PUT AND GET

The procedures read and write are in fact implemented using file buffers and the I/O primitives put (for output) and get (for input from files). When a file is rewritten, no information is placed in the buffer until a write is performed. This actually consists of the two operations:

```
F^ := data;
put ( F )
```

Similarly, the statement read (F, data) can also be expressed:

```
data := F^;
get ( F )
```

Thus the I/O statements in our Copy procedure:

```
read ( source, character );
write ( destination, character )
```

could equally well have been coded as:

```
destination^ := source^;
put ( destination );
get ( source )
```

We would then not require the local char variable, character. A better understanding of the operation of ReadLn (F) may be obtained, perhaps, if we express it in terms of these primitives:

```
WHILE NOT EoLn ( F ) DO
   get ( F ); { throw away the rest of the line, if any }
get ( F ) { . . and skip the EoLn character(s) }
```

Thus after a ReadLn the file buffer will always contain the first item on the next line to be read. This could be a space if EoLn (F) is true or be undefined if EoF (F) is true. It is obviously illegal to attempt to read from a file when EoF is true, but it is also an error to perform any operation, including testing the file buffer. This means you need to be careful when dealing with files other than input and output. But by the same token, having to be aware of potential error conditions such as these will encourage the writing of extremely robust software.

Now that we know how to 'look ahead' at an incoming data stream, we can formulate the SkipBlanks procedure that we postulated in the previous instalment.

```
PROCEDURE SkipBlanks ( VAR F : text );
   CONST
      space =' ';
   VAR
      done : boolean;
   BEGIN
      done := EoF ( F );
      IF NOT done THEN
         done := input^ > space;
      WHILE NOT done DO
         BEGIN
```

In A File's Shadow

The process of opening a file F in PASCAL sets up an associated buffer area, F^, into which the first character from the file is read. When a read is performed, the character in the buffer area is assigned to a variable and the next character from the file is transferred to the buffer area. If the first few characters of F are PENCILS, then, on opening the file, P will be read into F^. Following the first read operation, P will be assigned to a PASCAL variable, and be replaced by E in the buffer, F^

ERRATUM

The procedure Line on page 1179 lacked a BEGIN statement. As always, this word must appear after the declaration part of the procedure (in this case, one VAR declaration) and before any executable statements

```
        get ( F );
        done := EoF ( F );

        IF NOT done THEN
           done := F^ > space
    END

END; { SkipBlanks }
```

Notice that this will skip all white space on a text file, including end of line characters. If we just wish to skip space on any one line, we can alter the conditional assignment to:

```
done := EoLn ( F ) OR ( F^ > space)
```

Should we subsequently wish to skip all white space:

```
REPEAT
   SkipBlanks ( F );

   IF NOT EoF ( F ) THEN
      TextFound := NOT EoLn ( F )

UNTIL TextFound OR EoF ( F )
```

With this last modification it becomes a simple matter to write interactive programs checking for null inputs. For instance:

```
REPEAT
   write ('Enter data :');
   SkipBlanks ( input )

UNTIL NOT EoLn ( input )
```

This will fail if you happen to enter the control character used by the system to indicate the end of file, but we could always use the previous scheme if we wish to make our program absolutely safe.

The procedure assign has come to be regarded as a 'standard extension' and may well be officially adopted soon, along with open and seek for random access files. Other essential extensions in common use are:

```
FUNCTION Fstat ( FileName )
```

which returns a boolean result — true if a file already exists — and a

```
PROCEDURE Rename ( OldName, NewName )
```

facility.

Pascal Flavours

As far as PASCAL is concerned, there is no restriction on how many files you can have open at any one time. However, because files require using the computer's operating system, we find one area in which there are divergences from its strict rules. For example, some versions of the language, such as Hisoft PASCAL, do not support files.

The operating system file naming conventions may therefore pose a problem. Many mainframe PASCALs will take the first 10 or so characters of any file identifier and connect it with a file of that name, so our permanent file example would create one called DATAFILE. On many systems, the naming conventions are such that this is not possible. Some examples would be: A CP/M-FIL.DAT, #4:APPLEFOR.MAT and O:P.BBCFILE. The 'standard extension' for connecting a PASCAL file identifier is the assign procedure, used before resetting or rewriting:

```
assign ( FileIdentifier, NameString )
```

So for our present purposes, assign (DataFile, D:OurFile.dat) would suffice

CopyText Program

PASCAL provides the identifier 'text' for the most common type of file, and this is used when we declare any text file variables in the program header parameter list. Here, we provide a text file copying facility. In order to make the program the most generally useful, we have formulated the whole of the copying process as a separate procedure called Copy, which can process any two text files.

Remember that non-standard compilers will require you to use reset (F1, 'Source') and so on in the main program. Notice that both files are passed as VAR parameters to the Copy procedure. This is because not only is the destination file updated, but the source file will also be read and so the status of the file will change. For this reason, file variables are always passed by address, never by value. Apart from any other consideration, a value parameter implies a local copy — a tall order with files that are always potentially bigger than the available memory! Passing of large structures such as arrays of records may also be made an exception for reasons of memory conservation

```
PROGRAM   CopyText  ( F1, F2 );

VAR
     F1,
     F2    : text;

{11111111111111111111111111111111}

PROCEDURE  Copy ( VAR  source,
                       destination  : text );
  VAR
     character       : char;
  BEGIN
    WHILE NOT EoF ( source ) DO
      BEGIN { copy a line : }
         WHILE NOT EoLn ( source ) DO
           BEGIN
              read ( source, character );
              write ( destination, character )
           END;
              { now copy the end of line : }
         ReadLn ( source );
         WriteLn ( destination )
      END

  END;    { Copy }
{11111111111111111111111111111111}

BEGIN     { CopyText - Main program }

  assign ( F1, 'Source' );
  reset ( F1 );      { locate and open for reading }
  assign ( F2, 'Destination' );
  rewrite ( F2 );    { create it }
  Copy ( F1, F2 )

END .
```

THE MIGHTY PEN

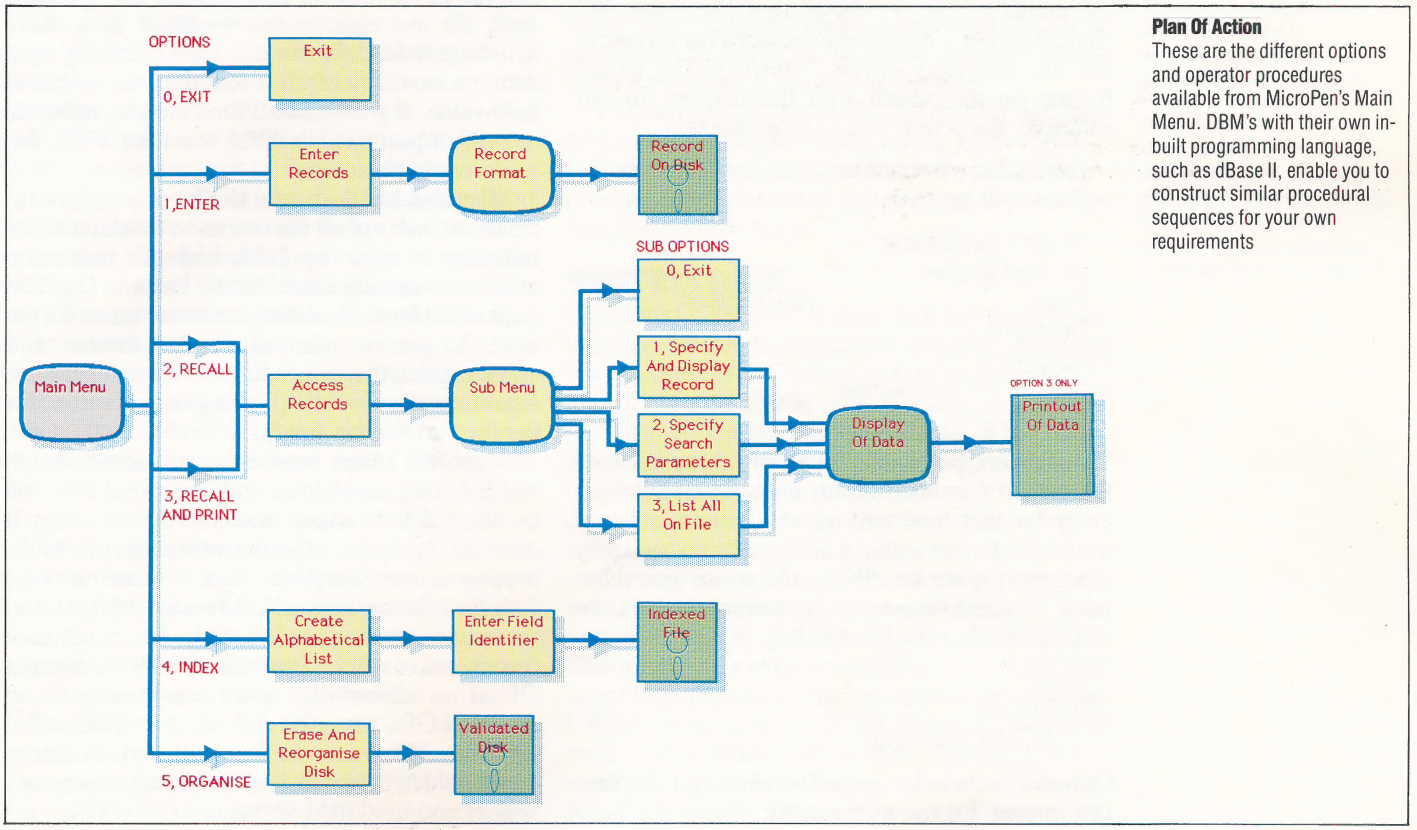
Using Archive, Card Box and dBase II, we have illustrated thus far many of the fundamental concepts of database management. We now turn our attention to MicroPen, designed for the Amstrad CPC 464, which is just one of many DBMs now available at relatively low cost for most home micro users.

Described as a 'database filing system for the CPC 464', MicroPen is published by the Amsoft division of Amstrad, but is the work of the software house, Intelligence Ireland. MicroPen forms part of a triptych of programs comprising a DBM, word processor and spreadsheet. The package has conceptual similarities to suites such as the Lotus 1–2–3 and Sord's PIPS. The DBM component includes a word processor called Penform. This is used to create the on-screen format of the records to be used within a DBM file. The DBM itself is called, simply, Pen.

The Amstrad version of MicroPen runs under CP/M-80, and therefore requires at least one DDI-1 disk drive. Although this might seem like an unfortunate extra expense, you will quickly realise that the speed and capacity of disks is essential — cassette-based DBMs can be painfully slow and frustrating.

Before a database file can be created, you must first create the layout or format for the records that will go into the file. This is done by running Penform. Although referred to in the documentation as a word processor, this is really just a screen editor. That is to say, it allows characters to be entered or modified on the screen, moving the cursor around using the cursor keys. Simple editing commands, such as [CTRL]Y to delete a line or [ESC]E to save to disk, augment the cursor keys. Penform is not a full word processor, as it cannot be used to create and edit full-length documents such as letters or articles.

There are a few restrictions on how much information can go into a MicroPen database file. A record (called a 'layout' in the documentation) may have up to 100 fields in it, but no record may contain more than 1,024 characters. Theoretically, a file may contain up to 32,750 records, but in practice such a large number of records is not feasible on a floppy disk-based system. The full complement of 32,750 records, each containing the maximum allowable 1,024 characters, would require more than 33 megabytes of disk storage just for the raw data alone (assuming that data-



Plan Of Action
These are the different options and operator procedures available from MicroPen's Main Menu. DBM's with their own in-built programming language, such as dBase II, enable you to construct similar procedural sequences for your own requirements

compression techniques are not used).

An unusual feature of the MicroPen DBM is that each field in a record has to be given a unique identifier on-screen. These identifiers are called 'field markers' and may be any printable character except the square bracket, which is unavailable because it has a special role as a field delimiter.

If we were using MicroPen as the DBM for our PARTS database, a record layout created by Penform might look like this:

```
Maker's Part Number:    [A                    ]
Our Part Number:        [B                    ]
Price:    [C    ]       No. In Stock: [D      ]
Description:    [E                            ]
Supplier:    [F                               ]
Supplier's Phone Number:    [G                ]
Who To Speak To:    [H                        ]
```

Having created a format such as this, it is saved to disk using a filename such as PARTS.INP (the .INP 'extension' to the filename is required by the Pen DBM program).

RUNNING PEN

When it is run, Pen displays a prompt on the screen that asks which database file is to be used. To use our inventory database, we would answer PARTS (the .INP extension to the filename is not required). Pen then displays a menu, known as the 'main menu', which provides five options:

```
Database: PARTS, per record 328, records on file 0.

0=Exit, 1=Enter, 2=Recall, 3=Recall&Print, 4=Index,
5=Organise
```

Option 1 is needed to enter records. The top line — the statistics line — shows that there are 382 characters per record, but no records (as yet) are in the file. Selecting option 1 displays the record format on the screen with instructions. In our example, the screen would look like this:

```
Press {ESC} when data entry completed for record 1
Press^C to clear field

Maker's Part Number:
Our Part Number:
Price:                          No. In Stock:
Description:
Supplier:
Supplier's Phone Number:
Who To Speak To:
```

The data is typed in the usual way. After the data for a field has been input, pressing ENTER ends entry for that field and moves the cursor to the next field. Errors within a field can be corrected by using the Delete key. When the entire record has been entered correctly, a sub-menu will be displayed using the Escape key:

```
0=Exit
1=Continue
2=Write Record to File
```

Option 2 writes the record to disk and displays the format for the next record.

The main menu option 2 allows records to be accessed, or recalled. This option brings up another sub-menu:

```
Recall by:    0=Exit
              1=Record number
              2=Search
              3=List all on file
```

Options 0 and 1 are fairly self-explanatory; option 1 allows you to specify and display a relevant record — it assumes, however, that you know the number of the record you want, although it is unlikely that you will remember this if there are many records in the file. Option 2 allows various search parameters to be specified so that a certain record can be located that conforms to the specification. The search sub-menu is:

```
Press Escape when search profile complete. To set
search mode:
^Q=Contains,^W=Not contains,^E=Equals,^R=Not
equals,^T=Greater,^Y=Less
```

You will already have noticed that there is a great inconsistency in the commands and menu options for the various parts of the package: sometimes the Escape key must be pressed, sometimes a CTRL - letter combination, sometimes a menu number. The CTRL - letter combinations seldom have real mnemonic values — ^E to find a record with matching fields during a search, ^Q to specify Contains and so on. Be that as it may, the 'search' options allow reasonable flexibility in specifying the records to be searched for.

Having given the parameters that define the record required, CTRL - letter combinations allow records to be located that contain the specified data; do not contain the specified data; have matching field data; do not have matching field data; or have a field value less than the specified field value. If you wanted, for example, to locate records of parts with a PRICE less than 37.50, this could easily be achieved.

MicroPen has the useful facility of being able to create an index of all the entries in a database file. Indexing is done by field, and the maximum number of records must also be known. The field is specified by its identifier, not by our name for the field. To get an index of Our Part Number, and assuming that there were 500 records in the file, we would specify B=#500 (B being the identifier for the Our Part Number field).

MicroPen offers reasonably advanced search and indexing capabilities and, at around £50, will be affordable to many home computer users. It does not, however, offer the advantage of a built-in programming language, such as those included with Archive and dBase II. Although dBase II is an expensive package designed for expensive computers, Archive comes 'free' with the Sinclair QL at an all-inclusive price comparable to an Amstrad CPC 464 plus disk drive and MicroPen software. These are the kinds of considerations you should bear in mind before buying a computer system and the DBM software to go with it.

AMSTRAD SPIN-OFF

The Amstrad computer's modular design and excellent BASIC ensured its popularity among less experienced users, but its lack of a fast storage capability has restricted its appeal to the lower end of the market. We examine Amstrad's DDI-1 disk drive package, which provides an economic upgrade for the machine.

The concept of the computer as another article of home electronics like a television or hi-fi, permanently set up in a corner of the room rather than having to be assembled and connected to other components each time it is to be used, obviously struck a chord with enthusiasts. There is also the bonus of avoiding battles over who uses the television.

The Amstrad is highly popular with home users, but it has had one major drawback: despite the fact that the company built a cassette drive into the machine, the computer lacked a fast storage capability. A disk drive was promised at the launch, prompting many people to buy the computer with the expectation that this would soon be available. However, shipments did not appear until the following year and the device is becoming widely available through retail outlets.

The disk drive package consists of the disk drive, an interface (allowing the machine to be connected to the floppy disk port at the rear of the computer), a system disk and a manual. The other end of the interface cable slides into the 34-way connector on the back of the disk drive.

The disk drive itself uses the Hitachi-standard 3in disks. This seems an odd choice, for it is Sony rather than Hitachi that appears to be winning the battle to capture the microfloppy market. The Sony 3½in format looks like becoming the standard as an increasing number of manufacturers — including Apple, Apricot and recently Acorn with its Electron — adopt Sony disks for their drives. Other than Amstrad, no major computer manufacturer seems to have opted for Hitachi disks.

Inside the drive, two clearly visible motors control the disk rotation and the single read/write head. At the back is the on-board power supply, shielded from the disk drive mechanism by a metal plate to guard against excess heat and stray magnetic fields. On the back of the casing above the interface port is an on/off switch.

The disks used by the Amstrad are similar in concept to their Sony counterparts, although they look very different. The Amstrad disks measure 100 by 80 by 4mm and, like the Sony disks, are cased in plastic and have a metal plate over the read/write window to protect the disk from fingermarks. This is slid back when placed in the drive. The Amstrad disks, however, have their shields on the inside of the metal casing, whereas Sony has placed them on the outside.



CHRIS STEVENS

Moving Up Market
The Amstrad DDI-1 disk drive enables users to upgrade their systems to the specifications required of a 'serious' microcomputer. With the addition of the three progams bundled with the disk drive — CP/M, AmsDOS and Dr LOGO — the Amstrad CPC 464 now has a much greater number of applications. The drive connects with the computer, via the interface cable that is provided, through the floppy disk interface port

The drive worked quickly and reliably and had no trouble finding files and loading them within a few seconds, but it seemed noisier than Sony's product. It was, however, quieter than the average 5¼in floppy disk drive.

SYSTEM DISK

Contained within the system disk are three basic utilities: AmsDOS, Amstrad's own disk operating system, and two utilities from Digital Research — the widely used CP/M disk operating system and Dr LOGO, a popular implementation of the turtle graphics and learning language becoming more widespread on micros and showing signs of displacing BASIC as the primary language on home computers.

Turning first to AmsDOS, this is perhaps the weakest of the three utilities provided. To be able to run AmsDOS, you first have to LOAD CP/M. This may seem a trifle strange, but the result is that AmsDOS appears to use no more memory than CP/M alone and so CP/M must be discarded once it has loaded AmsDOS. Unlike CP/M, which is a self-contained operating system, AmsDOS merely adds disk operating commands

to the bar symbol character | (Shift @); for example: |DRIVE and |DIR.

The method of manipulating disk files from AmsDOS is one of the most peculiar systems seen on any home micro. In order to, say, ERAse a file, you must first assign the filename to a string; only then can you delete the file by erasing the string. Thus, to erase the file INVOICE.$$$, the sequence of commands is:

```
A$="INVOICE.$$$"
|ERA,aA$
```

This method becomes even more complicated when RENaming a file, since two different files (old name and new name) have to be assigned to strings that can then be manipulated. Many users may find it easier to boot CP/M for such procedures when the commands can be carried out on a single line. However, CP/M has its own difficulties. Primarily, there are no facilities for running Amstrad BASIC under CP/M, so you will be forced to use AmsDOS if you want to program the disk drive system in BASIC.

The CP/M operating system on the system disk is the 2.2 version that has been applied on a vast number of eight-bit business machines in the last decade. Along with the CP/M itself, the usual 'transient' commands are provided with the system. Transient commands are those held permanently on disk and loaded into the computer's memory only when required; they are then discarded by the CP/M operating system. The trouble here is that to be able to use a large number of commands in CP/M (such as PIP, which transfers a file from one peripheral device to another) the operating system really requires two disk drives to be fitted to the computer — one to hold the systems disk so that the transient commands can be accessed quickly and simply, and another to hold the files.

Apart from the necessity in using the CP/M to continually swap the data and systems disk in and out of a single drive, many commands simply make no provision for the user having only a single drive available and expect to be able to transfer the files from one drive to another. To counter this, Amstrad has included extra transient commands for single drive transfer of files. FILECOPY provides prompts enabling you to exchange disks when transferring a file and, similarly, DISCCOPY copies an entire disk. Still, you need to buy two disk drives to take full advantage of the system.

The cynical view is that this is a clever ploy by Amstrad to force customers to buy two disk drives rather than one. However, a more charitable explanation is that the provision of CP/M has more to do with Amstrad's long term strategy. The philosophy behind the company's range of computers is not to produce leading-edge technology products, but to provide tried and tested equipment that can be used for a wide range of applications. Hence the company's decision to opt for the Z80 processor when most other manufacturers are scrambling to produce 16-bit machines. The use of CP/M fits in well with this since, although not the most user-friendly of disk operating systems, it is adaptable and there are thousands of programmers throughout the world with experience of writing software to be run under the system.

Despite the problems of using CP/M with only a single drive available, Amstrad is to be congratulated on providing a full implementation of such a powerful disk operating system on a home micro. Only a few years ago a computer with CP/M capability would have cost about £2,000, and now Amstrad is providing the same system for less than half that amount.

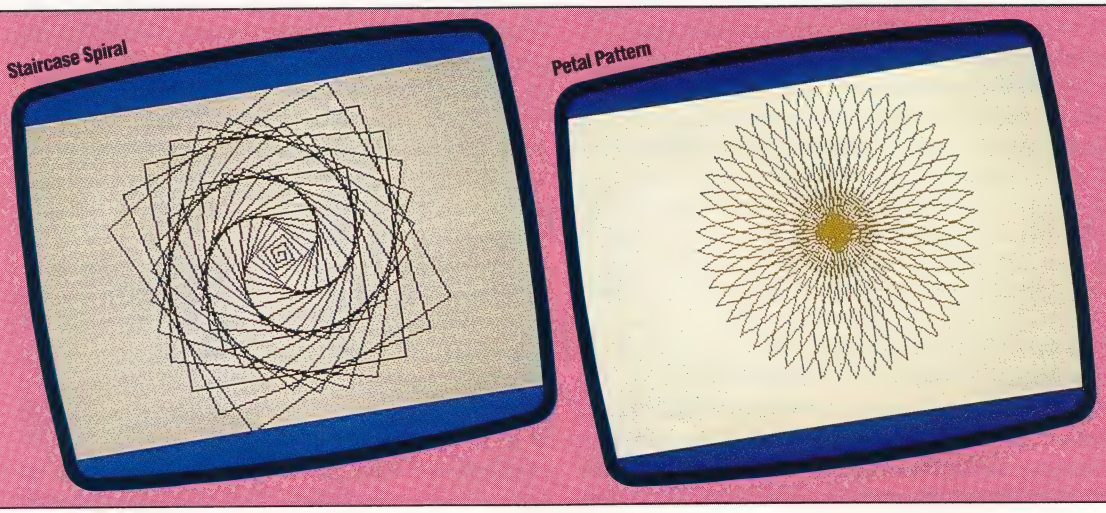
DIGITAL RESEARCH'S LOGO

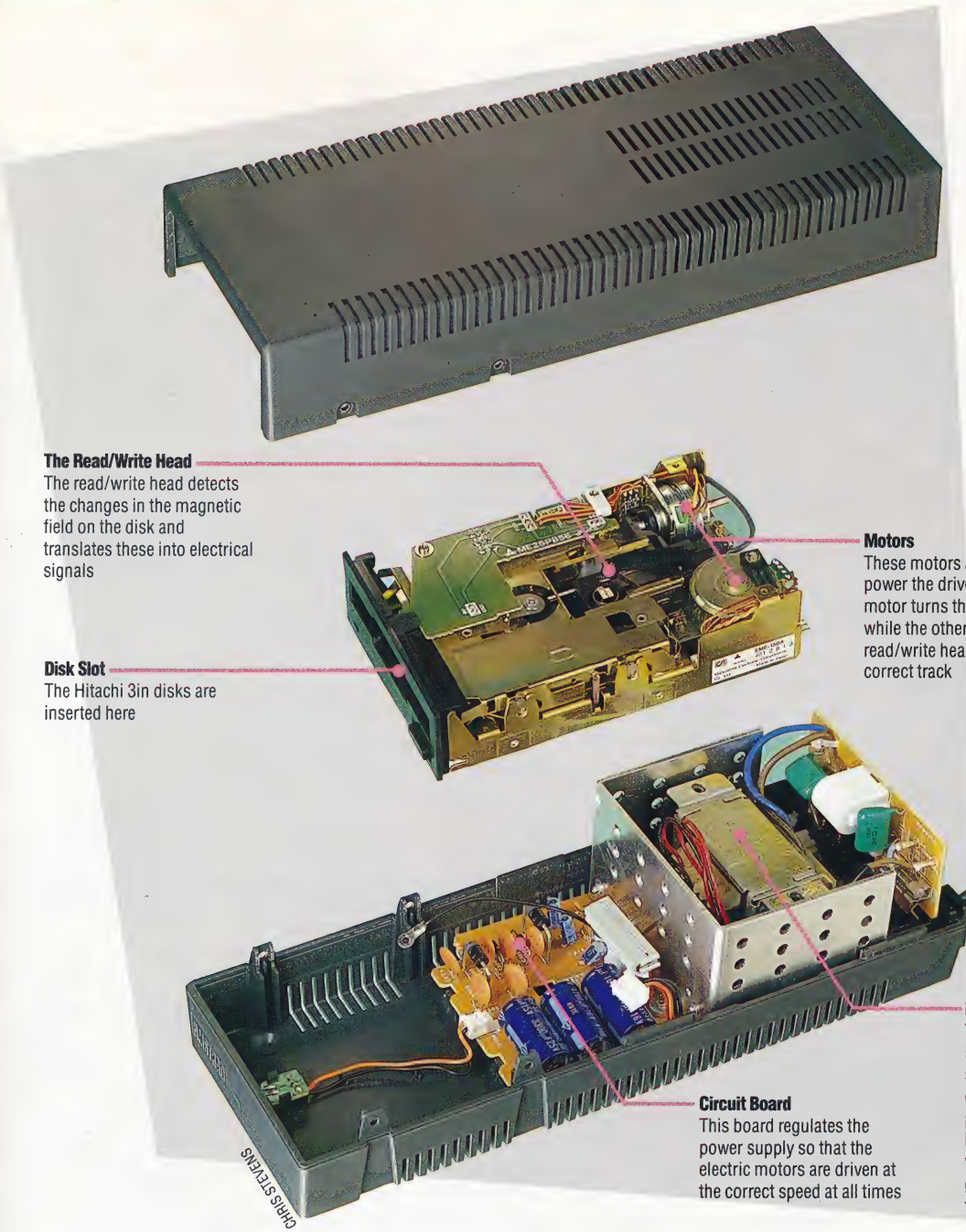
Dr LOGO is a version of the popular language used in many educational establishments. The language runs under CP/M, which therefore has to be loaded before you can load LOGO, and Dr LOGO can take advantage of CP/M's extended disk operating facilities. For example, CP/M allows the use of 'wildcards', which means that LOGO files can be called without a full name being typed in; so by typing a partial name, the language will call all the files fitting that partial description.

LOGO allows use of the sound and graphics capabilities of the Amstrad and implementation

Turtle Art

These patterns were created using Amstrad's Dr LOGO programming language. The simple procedures we used to create them took just a few lines of code each. Although Dr LOGO permits you to alter the foreground and background colours from within the language, no provision is made to enable you to alter the shape of the turtle or to colour in any shapes you draw. However, it is possible to generate sound from within LOGO





AMSTRAD DDI-1
PRICE
£199 inc VAT
DIMENSIONS
280×105×75mm
CAPACITY
Systems disk: 169K; data disk: 178K
INTERFACES
Single 34-way parallel interface. The interface cable accommodates two disk drives
DOCUMENTATION
The manual provided is patchy and it is sometimes difficult to locate the information you require
STRENGTHS
The provision of CP/M and Dr LOGO upgrades the Amstrad computer to a very powerful machine
WEAKNESSES
AmsDOS is not very well implemented and users with a single drive may find that neither AmsDOS nor CP/M is entirely suited to their needs

will be familiar to those who followed our LOGO series (beginning on page 506). Provision is also made for the language to be operated from a joystick and fire buttons, so it is possible to write games in LOGO that can be easily interfaced to joystick control.

Although the implementation of Dr LOGO is very good and compares well with many other versions of the language, there is some doubt as to how well it fares under the hardware limitations. For example, a tessellation program (one that requires the same design to be repeated a number of times in order to fill the screen) that runs perfectly well on the Commodore 64, generated a 'not enough space on stack' message on the Amstrad, indicating that perhaps the language does not fit as comfortably as it could. Apart from this, Dr LOGO is fast and friendly and is an ideal introduction to the language.

As with all disk systems for home micros, the popularity of the system, at least initially, will stand or fall on software support. Amstrad has already made a number of utilities for the disk drive, most of which seem to be aimed at the more 'serious' home enthusiast. These packages include a database, a word processor and additional languages such as PASCAL.

In launching the disk drive for its machine, Amstrad has taken an apparently successful step towards making its computer a more up-market machine without imposing a price normally associated with such systems. The addition of a disk drive makes the computer even more competitive and illustrates Amstrad's intentions to broaden the scope of the machine beyond the fading games market. It seems a pity that some of the utilities were not made a little friendlier, as the majority of customers will be new to disk drives.

BATTEN DOWN THE HATCHES

This instalment of our programming projects simulation game deals with the codes for the four remaining possible major events. The ship may be attacked by pirates, it may be blown off-course by a storm, the rudder may break and the crew may spot an island and change course, hoping to replenish any provisions that have been depleted over past weeks.

The program is randomly directed to a major contingency subroutine by the subroutine at line 6500, which generates a random number and uses this in the ON X GOSUB statement at line 6510, as described in the previous module (see page 1192). To accommodate the number of possibilities in this module, it is necessary to add the first line numbers of the four extra contingencies to this line. Line 6510 should now read:

```
6510 ON X GOSUB 6530, 6700, 6800, 6900, 7000,
7050
```

The third line number calls a subroutine in which the ship is attacked by pirates. The program checks whether this contingency has already occurred by looking at the value of the event flag for this subroutine, M(3). If M(3) is 1, it returns to the main program; otherwise it continues, setting it to 1 in line 6818 to avoid repetition.

It is possible that all the crew may be dead, making an attack by pirates irrelevant. The program sets up a loop to check the values in the crew strength array, and counts the crew members with strengths of 0 or -999 by looking at the strength rating, TS(T,2), the strength/type array. Dead or non-existent crew members are counted in X and if X equals 16 there are none remaining and control will be returned to the main program.

If the contingency has not been previously implemented, and there are crew members on board, the pirates attack, killing some of them. The number killed will depend on whether the crew members have guns to defend themselves. A count of the dead is recorded in K, the minimum number being two. Line 6825 sets the value of K to this amount, and the program then looks in the appropriate element of the supplies array, OA(2), to see whether there are guns aboard. If the value of OA(2) is 0 or -999, there are no guns and K is set to 4. If there are guns in the ship, the variable S$ is set to IN SPITE OF YOUR GUNS. If K has been set to 4, S$ is changed to YOU HAVE NO GUNS.

Another loop is set up, between lines 6836 and 6845, to delete the appropriate number of crew members, and line 6835 sets X equal to 0 to keep a count of this figure. If the strength value of a particular element is 0 or -999, the loop moves on to consider the next element; if not, line 6840 sets the value to -999, and increments the value of X by 1. When the value of X becomes equal to the number of crew due to be killed, K, line 6842 sets the loop count, T, to 16 and the program drops out of the loop. If there are not K members of the crew left to kill, the program will drop out of the loop after searching through it 16 times. The number of crew killed is printed, to complete the sentence started at either line 6828 or 6830. The player is then asked to press any key to continue.

RUDDER DAMAGE

The next contingency concerns the rudder being damaged and needing to be repaired. If a mechanic has been hired, and is still alive, the rudder will be fixed quickly and the journey will continue after a short delay, but if there is no mechanic on board, the journey will, of course, take longer. The damaged rudder subroutine begins at line 6900 and is the fourth line number in the ON X GOSUB statement at 6510. Again, the subroutine checks whether this event has already happened by the usual method — seeing whether the value of M(4) has been set to 1 and, if not, setting it so.

The value of X is set to 4, signifying the number of extra weeks the journey will take if there is no mechanic available. The program then checks through the strength type array TS(,) for a live mechanic. A loop is set up between lines 6930 and 6938 that looks for the number 3 in the crew type array in TS(,) representing a mechanic, and a value for the mechanic's strength not equal to 0 or -999, signifying that the mechanic is still alive. If all these conditions are upheld, then the value of X is reset from 4 to 1. The broken rudder will be fixed quickly if a mechanic is available, and the length of the journey will be increased by only one week.

The player is told either: ALTHOUGH YOU HAVE A MECHANIC, or, if X is equal to 4, YOU HAVE NO MECHANIC AND. The extra length of the journey is then printed. This value is added onto the total journey length, EW, at line 6965 and pressing any key at this time will continue the game. The variable X, therefore, has two purposes: it first indicates the number of weeks to be added to the voyage, but also acts as a flag, indicating whether a mechanic is available or not.

A storm, handled by the subroutine at 7000, represents the fifth possible major event. It blows the ship off-course and increases the length of the journey. If there is a navigator in the crew, the ship will quickly be set on the correct course. If not, the

IAN McKINNELL

journey will take longer. This subroutine is similar in structure to the 'broken rudder' subroutine, and, in fact, uses part of its coding.

The program checks whether the storm has already occurred and, if not, sets the value of M(5) to 1. The extra journey time is recorded in the value of X, which is set to 2, and if there is no navigator aboard, the journey will take two extra weeks. A loop is set up from 7030 to 7038, to scan the crew array for navigator, searching for a value of 4 in the type array, and a value greater than 0 in the strength array. If these requirements are met, the value of X is set to 1, and T is set to 16, causing the program to drop out of the loop.

S$ is set to ALTHOUGH YOU HAVE A NAVIGATOR or YOU HAVE NO NAVIGATOR AND, depending on the value of X. The program then goes to line 6950, in the broken rudder subroutine, in order to re-use the section of code, which prints the extra journey time, increments EW, the journey length, with X, the extra time, and returns to the main program.

SIGHTING AN ISLAND

The final major event, the sighting of an island, occurs at the subroutine starting at line 7050. This is the last line number in the ON X GOSUB statement at line 6510. The island is not on the main route, and so visiting it will add time to the journey. However, the ship may be able to replenish some of its provisions by changing course and landing. Thus the player must make a choice between changing direction for the island and continuing on-course. If the ship visits the island, the result of the expedition for food and water may be strongly influenced by any previous encounters with the albatross. If the player has shot the albatross the diversion will not be successful!

The program checks whether the subroutine has been implemented previously, in the usual way. The player is then informed that the charts indicate an island where provisions may be re-stocked, that the diversion will add time to the journey and is then asked whether the visit should be made. Line 7082 waits for a reply and looks at the first character of the input to determine whether the answer is yes or no. If the player does not want to make the detour, control is returned to the main program by line 7086. If the answer is YES or Y, the player reaches the island at line 7100.

The program then checks whether the player has shot the albatross, at line 7106. If the albatross was not killed, B$, which was set at N in line 48, will be unchanged and the program will go to line 7110. If the player shot the bird, however, B$ will have been changed to Y, by line 6162 of the albatross subroutine. If the bird was killed, the island will be barren and the water poisoned, no extra provisions will be obtained and the player will be reminded of the shooting. The program is sent to line 7130, which increments the journey length.

If the bird was not shot, a loop, from 1 to 4 for each provision type, is set up at line 7110 to collect the extra provisions; the amount of supplies is selected at random. Line 7112 generates a random number, and if less than .25, goes to the next provision. (There is a one-in-four chance of not obtaining each provision type.) Line 7115 generates a random number between 5 and 14, stored in X. This amount is printed by 7120, followed by the units of that particular provision, whether in kilos or barrels. If a provision had been washed overboard during the current week, the amount in the provision array would have been set to -999. If so, line 7122 resets it to 0 to enable the extra provisions to be recorded. The extra provisions, X, are added to the existing provisions by line 7125. The journey will be increased by one or two weeks. Line 7135 randomly decides what the increase will be, the player is informed, and the extra time is added to the journey length variable, EW, at line 7140.

In the next instalment we consider the factors relating to a possible mutiny.



Module Nine: Other Major Events

Initialise Flags

```
48 A$="N":B$="N"
```

Pirates Contingency Routine

```
6800 REM PIRATES
6805 IFM(3)=1THENRETURN
6810 X=0
6812 FORT=1TO16
6814 IFTS(T,2)=0ORTS(T,2)=-999THENX=X+1
6815 NEXT
6816 IFX=16 THEN RETURN
6818 M(3)=1
6820 PRINTCHR$(147)
6822 S$="  PIRATES ATTACK THE SHIP!*":GOSUB9100
6824 PRINT:GOSUB9200
6825 K=2
6826 IFOA(2)=0OROA(2)=-999THENK=4
6828 S$="IN SPITE OF YOUR GUNS*"
6830 IFK=4THENS$="YOU HAVE NO GUNS*"
6832 GOSUB9100
6835 X=0
6836 FORT=1TO16
6838 IFTS(T,2)=0ORTS(T,2)=-999THEN6845
6840 X=X+1:TS(T,2)=-999
6842 IFX=KTHENT=16
6845 NEXT
6850 PRINTX;
6855 S$="OF THE CREW IS KILLED*"
6856 IFX>1THENS$="OF THE CREW ARE KILLED*"
6860 GOSUB9100
6865 PRINT:GOSUB9200
6890 S$=K$:GOSUB9100
6895 GETI$:IFI$=""THEN6895
6899 RETURN
```

Rudder Contingency Routine

```
6900 REM RUDDER
6905 IFM(4)=1THENRETURN
6910 PRINTCHR$(147)
6915 M(4)=1
6920 S$="TROUBLE WITH THE RUDDER!*":GOSUB9100
6925 PRINT:GOSUB9200
6928 X=4
6930 FORT=1TO16
6935 IFTS(T,1)=3ANDTS(T,2)<>0ANDTS(T,2)<>-999
THENX=1:T=16
6938 NEXT
6940 S$="ALTHOUGH YOU HAVE A MECHANIC*"
6945 IFX=4THENS$="YOU HAVE NO MECHANIC AND*"
6950 GOSUB9100
6955 S$="YOUR JOURNEY WILL TAKE*":GOSUB9100
6960 PRINTX;"WEEKS LONGER"
6965 EW=EW+X
6967 PRINT:GOSUB9200
6969 S$=K$:GOSUB9100
6970 GETI$:IFI$=""THEN6970
6975 RETURN
```

Storm Contingency Routine

```
7000 REM STORM
7005 IFM(5)=1THENRETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="YOU ARE BLOWN OFF COURSE*":GOSUB9100
7022 S$="IN A STORM!*":GOSUB9100
7025 PRINT:GOSUB9200
7028 X=2
7030 FORT=1TO16
7035 IFTS(T,1)=4ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=1:T=16
7038 NEXT
7040 S$="ALTHOUGH YOU HAVE A NAVIGATOR*"
7045 IFX=2THENS$="YOU HAVE NO NAVIGATOR AND*"
7049 GOTO6950
```

Island Contingency Routine

```
7050 REM ISLAND
7055 IFM(6)=1THENRETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="YOUR CHARTS SHOW AN ISLAND*":GOSUB9100
7071 S$="WHERE YOU MAY BE ABLE TO*":GOSUB9100
7072 S$="RE-STOCK YOUR PROVISIONS*":GOSUB9100
7073 S$="BUT IF YOU GO THERE*":GOSUB9100
7074 S$="IT WILL ADD TIME TO YOUR JOURNEY*":GOSUB9
100
7075 PRINT:GOSUB9200
7080 S$="DO YOU WANT TO GO THERE*":GOSUB9100
7082 INPUTI$:I$=LEFT$(I$,1)
7084 IFI$<>"Y"ANDI$<>"N"THEN7082
7086 PRINT:GOSUB9200
7090 IFI$="N"THEN7145
7100 S$="YOU REACH THE ISLAND*":GOSUB9100
7105 S$="AND OBTAIN:*":GOSUB9100
7106 IFB$="N"THEN7110
7107 PRINT:GOSUB9200
7108 PRINT"NOTHING!":GOSUB9200
7109 S$="(REMEMBER THE ALBATROSS!)":GOSUB9100:GOTO
7130
7110 FORT=1TO4
7112 IFRND(1)<.25THEN7129
7115 X=INT(RND(1)*10)+5
7120 PRINTX;U$(T);"S OF";P$(T)
7122 IFPA(T)=-999THENPA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="BUT THE JOURNEY WILL NOW TAKE*":GOSUB9100

7135 X=INT(RND(1)*2)+1
7139 PRINTX;:S$="WEEKS LONGER*":GOSUB9100
7140 EW=EW+X
7145 PRINT:GOSUB9200
7150 S$=K$:GOSUB9100
7155 GETI$:IFI$=""THEN7155
7159 RETURN
```

Basic Flavours

Spectrum:
Make the following changes:

```
6512 IF X=3 THEN GOSUB 6800
6513 IF X=4 THEN GOSUB 6900
6514 IF X=5 THEN GOSUB 7000
6515 IF X=6 THEN GOSUB 7050

6820  CLS
6895  LET I$=INKEY$: IF I$=""THEN GO TO 6895
6910  CLS
6970  LET I$=INKEY$: IF I$=""THEN GO TO 6970
7010  CLS
7060  CLS
7082  INPUT I$: LET I$=I$(1 TO 1)
7155  LET I$=INKEY$: IF I$=""THEN GO TO 7155
```

BBC Micro:
Make the following changes:

```
6820  CLS
6895  I$=GET$
6910  CLS
6970  I$=GET$
7010  CLS
7060  CLS
7155  I$=GET$
```

OUTPUT DEVICE

This is any piece of machinery that allows a computer to interface with the 'outside world'—without *output devices*, the processed information within the computer could not be accessed. Their most common applications are to enable users to understand the results of the machine's calculations, although this is not their sole function. There are a large number of output devices available for the microcomputer at present, the most common of which are cathode-ray tubes — both monitors (see page 1120) and television screens — and printers. However, advancing technology and the increasing use of computers in a wide number of new applications means that other devices are continually being developed. For example, computer-controlled robots are becoming increasingly common in industrial environments. Looking further into the future, many experts are predicting the next major development in output devices to be speech synthesis, which will allow a computer to talk directly to the user.

OVERFLOW

This occurs when a resulting number after a function has been performed exceeds the memory space that has been set aside to store it. *Overflow* also refers to the amount by which the number has exceeded this space. In computing, this event will usually generate an error or the overflow number will be ignored. In either case, it is obvious that overflow should be avoided wherever possible.

PACKING

Packing is a process in which data is compressed into as small a memory space as possible in order to conserve memory. There are probably as many ways of achieving this as there are programmers, but the various systems fall into a few main categories. The most commonly used method is to store several bytes of data into a single computer word (two or four bytes). Other methods include 'keyword tokenisation', Boolean assignment of variables or 'array packing'. Often, however, packing will cause a subsequent loss in the speed of program execution. For example, in array packing (a feature built into the PASCAL programming language), the computer will require time to extract the necessary data from the pack.

PAGE

When we wish to find a reference in a book, rather than looking through a whole volume from beginning to end, we will most likely look in the index and discover on which *page* the reference is. In searching through a computer's memory, the same concept is used, and the procedure is referred to as 'paging' the memory.

In eight-bit computers a page consists of 256 bytes of information (the maximum number that can be addressed by a single byte). The machine's memory is divided into a number of pages, so when we wish to access data held at a particular location, we will instruct the computer to look at a particular page rather than having it start at the beginning of memory and look through all the addresses in order to find the correct one. From there it has only that one page's 256 bytes to search through.

The concept of 'paging' becomes very important when addressing bytes of memory in Assembly language. For simple calculations and other 'scratchpad' information, it is usual to use 'zero page', as this requires a single-byte address, and therefore requires less time to access the required data. However, for information above zero page, two bytes of information are required to obtain the address; one to specify the page and one to specify the location on that page.

PAGE PRINTER

A *page printer* is a device that prints pages of information in a single action, rather like a conventional printing press. This is in contrast to a line printer, which prints individual characters one after another. Obviously, this method of printing is considerably faster, and because of the way page printers act, their mechanism is somewhat different from the more traditional printers. To begin with, it is the paper that moves rather than the print head, and because of the speed at which print has to be transferred to the page, the normal impact printing methods are unsuitable. Hence, most page printers use electrostatic (see page 528) or laser (see page 948) technology to print characters.

Over To You
Output devices are generally peripherals that display computer data in ways that can be easily understood. The printer/plotters shown here are typical output devices. Digital data from the computer is decoded within the device and then translated into pen movements to draw pictures and print words on paper



CHRIS STEVENS

SINCLAIR'S SINEWS

In the previous two sections of Workshop we presented machine code and BASIC programs to control movement sequences of the Workshop robot arm for the BBC Micro and Commodore 64. We now look at a similiar program for the Spectrum that allows the arm to be controlled via the interface built earlier in the course.

The software required to control the robot arm must be based in machine code, as the digital servo motors used require regular pulses every one-fiftieth of a second. Such speeds are effectively beyond the reach of a BASIC program. Like the Commodore and BBC Micro versions, this machine code is executed on an interrupt basis, but it differs significantly because the Spectrum uses a Z80 processor rather than the 65XX series used by the other two machines.

The best way to handle interrupts in Z80 machine code is to use IM 2 mode interrupts. In simple terms, when this interrupt mode is set and an interrupt is generated to update the screen display (fortunately, around every one-fiftieth of a second), the processor gets the start address of the interrupt code from two sources. The hi-byte of the address is held in the I register and the lo-byte is taken from the state of the data bus when the interrupt occurred.

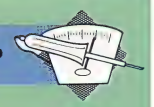
This system is designed for communication with peripherals attached to the data bus that can control the value held on the bus. However, in our application, we do not know the state of the data bus and so we have to accommodate all 256 possibilities. The program therefore sets up a page of memory to hold the real start address of our program. Filling a page with #FBs and pointing to the page by setting the I register causes an IM 2 interrupt to start executing code at location #FBFB. (This technique will be more fully described in a forthcoming Machine Code series on the Spectrum operating system.)

The rest of the machine code uses the same principles of operation as the 65XX versions described previously, outputting a stream of pulses to I/O port 31, the port mapped into by our interface. The BASIC control program also follows along similar lines to those developed for the BBC Micro and Commodore 64, allowing sequences to be programmed from the keyboard and replayed later.

Spectrum Arm Sequence Programmer

BASIC Loader

```
 10 REM ********************
 15 REM ********************
 17 REM **                  **
 20 REM **  spectrum arm **
 25 REM **   controller   **
 30 REM **                  **
 40 REM ********************
 50 REM ********************
 60:
 70 CLEAR 30000
 80 LET sa=64017: REM mc start wedge
 82 LET np=64008: REM newpos start address
 84 LET dl=64016: REM delay factor address
 86 LET en=64148: REM wedge off
 88 LET sm=64151: REM smooth move start addr
 90 LOAD "ARM.HEX"CODE sa
100 GO SUB 1000: REM set up
110 CLS
120 PRINT AT 6,2;"would you like to;"
130 PRINT AT 9,2;"1..program new sequence"
140 PRINT AT 10,2;"2..add moves to a program"
150 PRINT AT 11,2;"3..load a file"
160 PRINT AT 12,2;"4..leave program"
170 LET g$=INKEY$: IF g$="" THEN  GO TO 170
180 IF g$="1" THEN  LET c=1: LET lm=0
190 IF g$="1" OR g$="2" THEN  GO SUB 2000: GO SUB
3000: GO SUB 4000: GO SUB 5000
200 IF g$="3" THEN  GO SUB 6000
210 IF g$="4" THEN  CLS : RANDOMIZE USR en: STOP
220 GO TO 110
230:
1000 REM **** set up ****
1010 LET c=1: LET ns=4: REM no of servos
1020 LET lm=0: LET oc=0: LET df=10: REM delay fact
or
1030  LET mc=100: REM max count
1040 DIM r(mc,ns): REM key positions array
1050 DIM l(1): REM lastmax data array
1060 FOR i=0 TO 3: POKE np+i,0: NEXT i
1070 RANDOMIZE USR sa
1090 RETURN
1099:
2000 REM **** inform ****
2010 CLS
2020 PRINT AT 3,2;"please use"
2030 PRINT AT 4,2;"n/m...for 1/r"
2040 PRINT AT 5,2;"p/l...for 1st arm u/d"
2050 PRINT AT 6,2;"a/z...for 2nd arm u/d"
2060 PRINT AT 7,2;"x/c...for grab open/close"
2070 PRINT AT 8,2;"s.....save position"
2080 PRINT AT 9,2;"q.....return to menu"
2090 PRINT AT 10,2;"r.....move to saved position"
2100 PRINT AT 11,2;"v/b...next and back count"
2110 PRINT AT 12,2;"e.....set new count"
2120 PRINT AT 13,2;"i/d...to inc/dec speed"
2130 PRINT AT 1,2;"count=";c;" ";
2135 FOR i=1 TO 4: PRINT r(c,i);" ";: NEXT i: PRIN
T
2140 RETURN
2190:
3000 REM **** program arm ****
3010 POKE dl,1
3020 LET dx=5
3030 LET a$=INKEY$: IF a$="" THEN  GO TO 3030
3035 IF a$="n" THEN  LET p=PEEK (np)+dx: IF p<256
THEN  POKE np,p
3040 IF a$="m" THEN  LET p=PEEK (np)-dx: IF p>0 TH
EN  POKE np,p
3050 IF a$="p" THEN  LET p=PEEK (np+1)+dx: IF p<25
6 THEN  POKE np+1,p
3060 IF a$="l" THEN  LET p=PEEK (np+1)-dx: IF p>0
THEN  POKE np+1,p
3065 IF a$="a" THEN  LET p=PEEK (np+2)+dx: IF p<25
6 THEN  POKE np+2,p
3070 IF a$="z" THEN  LET p=PEEK (np+2)-dx: IF p>0
THEN  POKE np+2,p
3080 IF a$="x" THEN  LET p=PEEK (np+3)+3*dx: IF p<
256 THEN  POKE np+3,p
3090 IF a$="c" THEN  LET p=PEEK (np+3)-3*dx: IF p>
0 THEN  POKE np+3,p
3100 IF a$="r" THEN  FOR i=1 TO 4: POKE np+i-1,r(c
,i): NEXT i
3110 IF a$="v" THEN  LET c=c+1: IF c>mc THEN  LET
c=1
3120 IF a$="b" THEN  LET c=c-1: IF c<1 THEN  LET c
=mc
3130 IF a$="e" THEN  PRINT AT 2,20;"count value":
INPUT c
3140 IF a$="s" THEN  FOR i=1 TO 4: LET r(c,i)=PEEK
 (np+i-1): NEXT i: LET c=c+1
3145 IF a$="i" THEN  LET dx=dx+1
3147 IF a$="d" THEN  LET dx=dx-1: IF dx<1 THEN  LE
T dx=1
3150 IF c>lm THEN  LET lm=c
3160 IF oc<>c AND c>1 THEN  PRINT AT 1,2;"count=";
c-1;" ";: FOR i=1 TO 4: PRINT r(c-1,i);" ";: NEXT
i: PRINT
3170 LET oc=c
3190 IF a$<>"q" THEN  GO TO 3030
3200 RETURN
3900:
4000 REM **** replay sequence ****
4010 CLS
4020 PRINT AT 12,2;"replay sequence y/n,r repeats"
```



```
4030 LET a$=INKEY$: IF a$<>"y" AND a$<>"n" AND a$<
>"r" THEN  GO TO 4030
4040 IF a$="n" THEN  RETURN
4050 IF a$="r" THEN  PRINT AT 14,1;"delay factor 1
-255";: INPUT df
4060 IF df<1 OR df>255 THEN  GO TO 4050
4070 POKE dl,df: REM set delay register
4080 FOR i=1 TO lm
4090 PRINT AT 1,2;"no in sequence=";i;" ";
4100 FOR s=1 TO 4
4110 POKE np+s-1,r(i,s)
4120 NEXT s: NEXT i
4140 GO TO 4010
4990:
5000 REM **** save a file ****
5010 CLS
5020 PRINT AT 12,2;"save sequence (y/n)?"
5030 LET g$=INKEY$: IF g$<>"y" AND g$<>"n" THEN  G
O TO 5030
5040 IF g$="n" THEN  RETURN
5050 LET l(1)=lm
5060 INPUT "filename";f$
5065 PRINT "press play and record"
5067 RANDOMIZE USR en: REM wedge off
5070 SAVE f$+".lm" DATA l()
5080 SAVE f$+".r" DATA r()
5085 RANDOMIZE USR sa: REM wedge back on
5090 RETURN
5900:
6000 REM **** load a file ****
6010 CLS
6020 PRINT AT 12,2;"load a file (y/n)"
6030 LET g$=INKEY$: IF g$<>"y" AND g$<>"n" THEN  G
O TO 6030
6040 IF g$="n" THEN  RETURN
6045 INPUT "filename";f$
6050 FOR i=1 TO mc
6060 FOR j=1 TO ns
6070 LET r(i,j)=0
6080 NEXT j: NEXT i
6090 RANDOMIZE USR en: REM wedge off
6100 LOAD f$+".lm" DATA l()
6110 LET lm=l(1): LET c=lm: LET oc=0
6120 LOAD f$+".r" DATA r()
6125 RANDOMIZE USR sa: REM wedge back on
6130 RETURN
```

Assembly Listing

```
                1000 ;SPECTRUM ARM CONTROLLER
                1010
001F            1020 PORT:  EQU  31
F900            1030        ORG  #F900
F900            1040 MOTTAB DEFS 256
FA00            1050 ANGTAB DEFS 8
FA08            1060 NEWPOS DEFS 8
FA10            1070 DELAY: DEFS 1
                1080
                1090 ; SET  UP VECTOR TABLE
FA11 2100FC     1100 INIT:  LD   HL,#FC00
FA14 01FB00     1110        LD   BC,#00FB
FA17 71         1120 LOOP1: LD   (HL),C
FA18 23         1130        INC  HL
FA19 10FC       1140        DJNZ LOOP1
FA1B 71         1150        LD   (HL),C
FA1C 3EFC       1170        LD   A,#FC
FA1E ED47       1180        LD   I,A
                1190 ;INIT TABLES TO FF
                1200 ;
FA20 2100F9     1210        LD   HL,MOTTAB
FA23 3EFF       1220        LD   A,#FF
FA25 06FF       1230        LD   B,#FF
FA27 77         1240 LOOP2: LD   (HL),A
FA28 2C         1250        INC  L
FA29 10FC       1260        DJNZ LOOP2
FA2B 2100FA     1270        LD   HL,ANGTAB
FA2E 0610       1280        LD   B,16
FA30 77         1290 LOOP2A LD   (HL),A
FA31 2C         1300        INC  L
FA32 10FC       1305        DJNZ LOOP2A
FA34 ED5E       1310        IM   2
FA36 C9         1320        RET
                1380 ; INTERRUPT HANDLER
                1390
FA37 F3         1410 HANDLE DI
FA38 F5         1420        PUSH AF
FA39 C5         1430        PUSH BC
FA3A D5         1440        PUSH DE
FA3B E5         1450        PUSH HL
FA3C FF         1460        RST  #38 :NML ROUTINE
FA3D F3         1470        DI
FA3E CD47FA     1480        CALL EVENT :OUR ROUTINE
FA41 E1         1490        POP  HL
FA42 D1         1500        POP  DE
FA43 C1         1510        POP  BC
FA44 F1         1520        POP  AF
FA45 FB         1530        EI
FA46 C9         1540        RET
                1550
                1560 ;
                1570 ;++++ EVENT ROUTINE ++++
                1580 :
FA47 0608       1590 EVENT: LD   B,8
FA49 3E7F       1600        LD   A,#7F
FA4B 16F0       1610        LD   D,#F0
FA4D 2158FA     1620        LD   HL,FIX+2
FA50 DD2100FA   1630        LD   IX,ANGTAB
FA54 3607       1635        LD   (HL),7
FA56 DD5E07     1640 FIX:   LD   E,(IX+7)
FA59 12         1650        LD   (DE),A
FA5A 0F         1660        RRCA
FA5B 35         1670        DEC  (HL)
FA5C 10F8       1680        DJNZ FIX
                1690 :
                1740 ;PREP MOTTAB FOR PORTSEND
                1750 :
FA5E 0600       1760        LD   B,0
FA60 3EFF       1770        LD   A,#FF
FA62 2100F9     1780        LD   HL,MOTTAB
FA65 A6         1790 LOOP3: AND  (HL)
FA66 77         1800        LD   (HL),A
FA67 2C         1810        INC  L
FA68 10FB       1820        DJNZ LOOP3
                1830 ;
                1840 ;++++ START PULSES ++++
                1850 :
FA6A 3EFF       1860        LD   A,#FF
FA6C D31F       1870        OUT  (PORT),A
                1880 :
                1885 ;++++ CALL DELAY ++++
FA6E D5         1887        PUSH DE
FA6F 1E04       1889        LD   E,4
FA71 16FF       1890        LD   D,#FF
FA73 CD8BFA     1892        CALL OLOOP
FA76 D1         1894        POP  DE
                1920 ;
                1930 ;++++ SEND MOTTAB TO PORT ++++
                1940 :
FA77 0E1F       1950        LD   C,PORT
FA79 06FF       1960        LD   B,#FF
FA7B 2E00       1970        LD   L,00
FA7D EDB3       1980        OTIR
                1990 :
                2000 ;++++ RESTORE MOTTAB TO FF +++
+
FA7F 3EFF       2010        LD   A,#FF
FA81 0600       2020        LD   B,0
FA83 2100F9     2030        LD   HL,MOTTAB
FA86 77         2040 LOOP4: LD   (HL),A
FA87 2C         2050        INC  L
FA88 10FC       2060        DJNZ LOOP4
FA8A C9         2070        RET
                2080 ;++++ DELAY LOOP ++++
                2090 ;
FA8B 1D         2100 OLOOP: DEC  E
FA8C C8         2110        RET  Z
FA8D 15         2120 ILOOP: DEC  D
FA8E CA8BFA     2130        JP   Z, OLOOP
FA91 C38DFA     2140        JP   ILOOP
                2200 ;
                2210 ;++++   RESTORE IM 1 ++++
                2220
FA94 ED56       2230 REST:  IM   1
FA96 C9         2240        RET
                2250 ;
                2340 ;++++ SMOOTH MOVER ++++
                2350 :
FA97 0E04       2360 START: LD   C,4
FA99 0604       2370        LD   B,4   ;SET UP COUN
TERS
FA9B 2103FA     2380        LD   HL,ANGTAB+3
FA9E 110BFA     2390        LD   DE,NEWPOS+3
FAA1 7E         2400 NEXMOT LD   A,(HL)
FAA2 EB         2410        EX   DE,HL
FAA3 BE         2420        CP   (HL)
FAA4 EB         2430        EX   DE,HL
FAA5 280A       2440        JR   Z,MOVED ;SAME
FAA7 3804       2450        JR   C,ADD   ;BORROW
FAA9 35         2460        DEC  (HL)
FAAA C3B2FA     2470        JP   NEXT
FAAD 34         2480 ADD:   INC  (HL)
FAAE C3B2FA     2490        JP   NEXT
FAB1 0D         2500 MOVED: DEC  C
FAB2 2B         2510 NEXT:  DEC  HL
FAB3 1B         2520        DEC  DE
FAB4 10EB       2530        DJNZ NEXMOT
FAB6 F5         2540        PUSH AF
FAB7 D5         2550        PUSH DE
FAB8 ED5B10FA   2560        LD   DE,(DELAY)
FABC 16FF       2570        LD   D,#FF
FABE CD8BFA     2580        CALL OLOOP :CALL DELAY
FAC1 D1         2590        POP  DE
FAC2 F1         2600        POP  AF
FAC3 20D2       2610        JR   NZ,START
FAC5 C9         2620        RET
                3000 ;++++ SET UP HANDLER ++++
                3010 ;++++ JUMP ADDRESS   ++++
                3020 ;
FBFB            3030        ORG  #FBFB
FBFB F3         3040        DI
FBFC C337FA     3050        JP   HANDLE

Pass 2 errors: 00

ADD     FAAD    ANGTAB FA00
DELAY   FA10    EVENT  FA47
FIX     FA56    HANDLE FA37
ILOOP   FA8D    INIT   FA11
LOOP1   FA17    LOOP2  FA27
LOOP2A  FA30    LOOP3  FA65
LOOP4   FA86    MOTTAB F900
MOVED   FAB1    NEWPOS FA08
NEXMOT  FAA1    NEXT   FAB2
OLOOP   FA8B    PORT   001F
REST    FA94    START  FA97
```

SPLITTING IMAGE

The second part of our investigation of Commodore 64 graphics considers how the VIC-II chip borrows time from the main 6510 processor to create the video display. We provide a program that allows high-resolution graphics to be displayed simultaneously on a split screen — a technique often used in adventure games to produce pictures and words simultaneously .

To display video data, the VIC-II chip must read from memory. This is accomplished by allowing the VIC-II access to some (but not all) of the lines on the address bus, and all the lines on the data bus. The process by which the VIC-II reads ROM or RAM shared with the 6510 is called *direct memory access* (DMA). Ideally, the VIC-II would be using the address or data lines at times when the 6510 does not need them, in which case the action of the two processors would be 'transparent' to one another. Unfortunately, the 6510 is a rather simple processor with very few internal registers and no 6510 instruction takes more than seven clock cycles (most take only three or four).

Since the VIC-II must read rather a lot of data, especially when displaying several sprites, there is not enough time available for it to operate in a transparent fashion. Because of this, the VIC-II has a special control line, called the *bus available* (BA) line, which it can use to send a 'hold off' signal to the 6510. This enables the VIC-II to reserve as much time as it needs to read the video data.

When the BA line is set low it indicates that the video chip will require some time from the 6510. The other chip is then given just enough time to finish its current instruction before VIC-II pulls down the *address enable control* (AEC) line and ruthlessly disables the 6510's address bus drivers. This is rather like zapping the 6510 with a phaser gun set to stun — it is quite unaware that it is put out of the game.

The amount of time stolen from the 6510 by the VIC-II chip is quite significant. For example, the character pointer addresses must be fetched after every eighth raster line displayed on the screen (since characters have eight rows of pixels), and each line requires 40 consecutive accesses to fetch the video memory pointers (there are 40 characters per screen row). In the PAL system (used in Britain), the video chip refreshes every other line on the 625 raster lines on the television screen roughly 25 times a second; the American NTSC system uses 524 lines, and refreshes these 30 times a second. This adds up to a significant amount of time. In fact, a little arithmetic shows that the net effect is to 'slow up' the 6510 by as much as 15 to 20 per cent.

The DMAs have no apparent effect on the internal working of the machine. However, where real time I/O is concerned, these unpredictable 'absences' of the 6510 may be a problem — for example, in the operation of the tape cassette. In such cases it may be necessary to blank the screen (using POKE 53265,11) during the I/O, and re-enable it (using POKE 53265,27) after the I/O is finished. Whether or not the VIC-II's use of direct memory access is a problem during I/O will depend on the external device and the method used to communicate with it. In many instances, such as a disk access, screen blanking will not be necessary.

SPLITTING THE SCREEN

An interesting feature of the Commodore 64 is that, by very careful programming, one can have the visible screen split between high- and low-resolution modes. This makes it possible, for example, to display a high-resolution graph or picture while at the same time printing text or engaging in some user interaction in the low-resolution portion of the screen. Characters cannot be easily printed directly on the high-resolution screen.

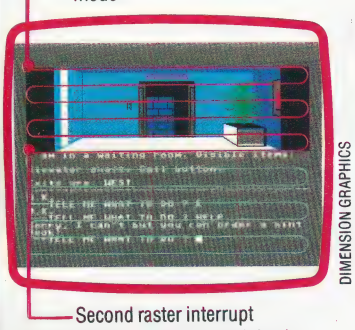
The Splitscreen program that we provide here demonstrates several of the points discussed. A small high-resolution screen is set up behind the BASIC interpreter ROM and displayed continuously in the top third of the visible screen. At the same time, the lower two-thirds remain in normal low resolution mode.

On page 1138 we illustrated the use of RAM vectors on the Commodore 64. We showed how changing the value of a two-byte pointer allowed us to divert an operating system routine along some code of our own. To get a split screen we shall divert the IRQ routine, but in a subtle way.

There are several ways that an IRQ to the 6510 can be triggered. The usual way is that one of the CIA timers sets a bit in the interrupt flag register (IFR), at address 53273 ($D019), every one-sixtieth of a second. This causes the Commodore 64 to execute the IRQ service routine, which — among other things — scans the matrix of keyboard switches to see if a key is being pressed. Bits in the IFR are also set to one by the VIC-II chip if certain events occur, such as when two sprites collide or if the raster count reaches a pre-set value. There is another register, the *interrupt enable register* (IER), at address 53274 ($D01A), which acts as an enabling switch to the 6510 IRQ line. If corresponding bits in the IER are set by the programmer, then bits set in the IFR will trigger a 6510 IRQ.

The general idea of the Splitscreen program is as follows:

1) When a raster interrupt occurs at the top of the screen, set the bit map mode, and then set up a



DIMENSION GRAPHICS

Both Sides Of The Story
Many commercial adventure games written for the Commodore 64 use raster interrupt techniques to produce high resolution pictures and text simultaneously. Spiderman by Scott Adams, shown here, uses the top third of the screen to display a location in the game; the remaining two-thirds are used for a description of the scene in the normal text display mode. The TV raster scan is programmed to generate two interrupts each time the screen is scanned: the first at the top of the screen and the other about one-third of the way down. An interrupt routine is called each time a raster interrupt occurs causing the display to flip between the high resolution and text modes.

raster interrupt half way down the screen. Return from interrupt (RTI).
2) On being interrupted half way down the screen, switch back to low-resolution mode and then set up the next raster interrupt to occur at the top of the screen (followed by another return).

Note one rather odd feature of this arrangement: part of the normal screen memory corresponding to the top third of the screen is being used for colour information for the high-resolution screen — the remaining two-thirds being used for character data in the normal way.

However, there is one snag with this program. The raster interrupts will occur about every one-fiftieth of a second and must be serviced immediately, as otherwise the raster beam will have moved on, displaying the wrong data. Now, in the normal course of events, the IRQ service routine will be triggered every one-sixtieth of a second. The service routine is quite long and, should it occur just before the raster interrupt, there will be a delay before getting around to our Splitscreen raster wedge. This must be avoided at all costs, or else the screen will flicker at the boundary of the high- and low-resolution modes.

The solution to this problem is to force the IRQ service routine to occur immediately after the raster wedge code. This is done by turning off the usual one-sixtieth of a second timer, thereby disabling the usual IRQ trigger, and jumping to the IRQ service routine after executing the raster wedge. This forces the two pieces of code to be executed in synchronous relation to one another. Of course, it also means that the keyboard will be scanned marginally less frequently — this has the useful side-effect of making BASIC run slightly faster, but otherwise does no harm.

Finally, we have put the high-resolution screen behind the BASIC interpreter ROM. This makes plotting to it a bit tricky in BASIC, since the language can't read the RAM area behind the BASIC ROM, and we cannot perform the necessary AND or OR operations to set individual pixels. However, a POKE will pass through the BASIC ROM to the RAM behind, so we can do simple plotting from BASIC. In pure machine code programs, however, we don't need the interpreter ROM — it is just eight Kbytes of wasted space — so we blank it out. A few simple amendments to the source listing will allow you to relocate the high-resolution screen so that it can be used more easily from BASIC.

Commodore 64 Splitscreen Program

BASIC Loader

```
900 REM**************************
902 REM**     SPLITSCREEN 64     **
903 REM**      BASIC LOADER      **
910 REM** NOTE-IN THIS VERSION **
920 REM** HIRES UNDER BASIC SO **
930 REM**   CAN'T USE PEEK IN   **
940 REM**     PLOT SUBROUTINE   **
950 REM**************************
960 :
1000 REM SPLIT SCREEN LOAD AND TEST**
1010 DATA76,50,192,49,234,173,25,208
1020 DATA240,37,169,255,160,21,141,25
1030 DATA208,173,0,221,73,2,141,0,221
1040 DATA173,17,208,73,32,141,17,208,41
1050 DATA32,240,4,169,121,160,255,141
1060 DATA18,208,140,24,208,108,3,192
1070 DATA173,14,220,41,254,141,14,220
1080 DATA173,20,3,141,3,192,173,21,3
1090 DATA141,4,192,169,5,141,20,3,169
1100 DATA192,141,21,3,173,17,208,41,95
1110 DATA141,17,208,169,0,141,18,208
1120 DATA169,255,141,25,208,169,1,141
1130 DATA26,208,165,1,41,254,133,1,169
1140 DATA0,133,247,169,160,133,248,160
1150 DATA0,162,28,169,0,145,247,200,208
1160 DATA251,230,248,202,208,246,169
1170 DATA188,133,248,160,0,162,4,169
1180 DATA183,145,247,200,208,251,230
1190 DATA248,202,208,246,165,1,9,1,133
1200 DATA1,96,255
1210 DATA20633:REM*CHECKSUM*
1220 CC=0:FORI=0TO160
1230 READX:POKE49152+I,X:REM INSERT CODE
1240 CC=CC+X:NEXT
1250 READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
1260 REM***TEST SPLIT SCREEN***
1270 SYS49152:REM SPLIT SCREEN
1280 CM = 40960:REM START HIRES
1290 REM** PLOT VERT  AXES **
1300 X=160:FORY=0TO70
1310 GOSUB1400:REM COMPUTE ADDRESS
1320 POKEM+R,2↑(7-C):REM PLOT POINT
1330 NEXT
1340 REM** PLOT HORIZ AXES **
1350 Y=37:FORX=0TO319
1360 GOSUB1400:REM COMPUTE ADDRESS
1370 POKEM+R,255:REM PLOT SEGMENT
1380 NEXT
1390 END
1400 REM** COMPUTE S/R **
1410 U=INT(Y/8):V=INT(X/8)
1420 R=YAND7:C=XAND7
1430 M=CM + (40*U+V)*8
1440 RETURN
```

Assembly Listing

```
                ;++++++++++++++++++++++++++
                ;++++++++++++++++++++++++++
                ;++                      ++
                ;++   SPLITSCREEN 64     ++
                ;++                      ++
                ;++++++++++++++++++++++++++
                ;++++++++++++++++++++++++++

                ;
                 *= $C000
                 JMP START
                ;
                 MEM1 = $F7
                 MEM2 = $F9
                 MEM3 = $FB
                 SCN = $FD
                 CINV = $314     ;IRQ VECTOR
                 TEMPIRQ *=*+2
                ;
NEWIRQ          =*               ;NEW IRQ WEDGE START
                ;
                 LDA $D019       ;LOOK AT THE INTERRUPT STATUS
                 BEQ NOTVIC      ;WAS THE IRQ FROM VIC?          DDD
                 LDA #$FF
                 LDY #$15
                 STA $D019       ;CLEAR INT LATCH.
                 LDA $DD00
                 EOR #$02
                 STA $DD00       ;CHANGE BANKS
                 LDA $D011
                 EOR #$20        ;CHANGE BIT MAP MODE BIT
                 STA $D011
                 AND #$20        ;TEST BMM BIT
                 BEQ LSCN        ;LOWER PART OF SCREEN
                 LDA #$79        ;SET RASTER TO LINE 22          DDE
                 LDY #$FF
LSCN
                 STA $D012       ;SET NEW RASTER COMPARE
                 STY $D018
NOTVIC
                 JMP (TEMPIRQ)
                ;
START           =*               ;INITIALISE IRQ WEDGE
                ;
                 LDA $DC0E
                 AND #$FE        ;DISABLE SCAN TIMER A
                 STA $DC0E       ;CIA #1
                 LDA CINV
                 STA TEMPIRQ     ;
                 LDA CINV+1      ;CHANGE IRQ VECTOR
                 STA TEMPIRQ+1   ;
                 LDA #<NEWIRQ
                 STA CINV
                 LDA #>NEWIRQ
                 STA CINV+1
                 LDA $D011
                 AND #$5F        ;RESET BMM MODE
                 STA $D011
                 LDA #$00
                 STA $D012
                 LDA #$FF
                 STA $D019       ;RESET IRQ LATCHES
                 LDA #$01
                 STA $D01A       ;ENABLE RASTER COMPARE
                 LDA $01
                 AND #$FE
                 STA $01         ;BANK OUT BASIC ROM
                 LDA #$00
                 STA MEM1        ;SET UP 0 PAGE PTRS TO POINT
                 LDA #$A0        ;TO START OF HIRES AREA BEHIND
                 STA MEM1+1      ;BASIC ROM
                 LDY #$00
                 LDX #$1C
                 LDA #$00        ;CLEAR HIRES
                 STA (MEM1),Y                                    DDD
                 INY
                 BNE DDD
                 INC MEM1+1
                 DEX
                 BNE DDD
                 LDA #$BC
                 STA MEM1+1
                 LDY #$00
                 LDX #$04
                 LDA #$87        ;SET COLOUR
                 STA (MEM1),Y                                    DDE
                 INY
                 BNE DDE
                 INC MEM1+1
                 DEX
                 BNE DDE
                 LDA $01
                 ORA #$01
                 STA $01         ;RESTORE BASIC ROM
                 RTS             ;BACK TO BASIC
```

CIPHER CHASE

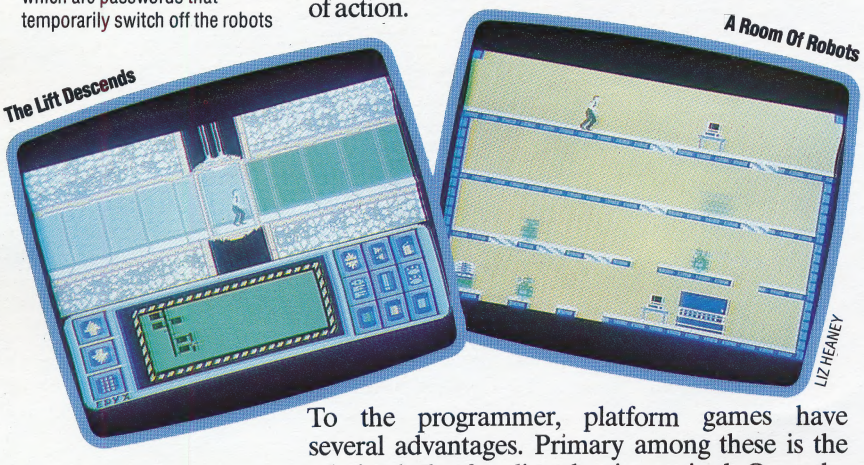
In a race against time, you find yourself penetrating the depths of the evil Elvin's fortress, collecting passwords to a valued code. Impossible Mission puts you in this unenviable but highly enjoyable position with excellent graphics and speech synthesis in a new 'platform' game from Epyx.

Impossible Mission: For the Commodore 64: cassette £8.95, disk £11.95
Publishers: CBS Software, 3-5 Rathbone Place, London W1
Author: Dennis Caswell
Joystick: Required
Format: Cassette or disk

One of the most popular types of games program is the platform game. In this type, the player (as the hero) must obtain articles that are positioned on platforms within the various screens. The hero usually enters the screen in one of the bottom corners and must reach the platforms by a series of lifts, ladders and trampolines (or whatever else the programmer has provided). Often, vital articles such as important treasure or keys will be placed at a particularly awkward position leading to considerable thought concerning how best to get there.

Games such as these are further complicated by aliens or other dangerous graphics figures that have to be avoided, since touching them usually proves fatal. These graphics are either static, or else they move along preset patterns. Futhermore, many of these sprites have a 'fire' capability. Depending on the intentions of the programmer, movements of these enemies can be more or less intelligent. Many will move back and forth along a single path whereas others may be programmed to 'home in' on the hero, cutting down your freedom of action.

Nothing's Impossible
The object of Impossible Mission is to travel via the lifts and corridors to the various rooms within the evil scientist's lair. Once you have entered the rooms you must, while avoiding the robots, search among the pieces of furniture for parts of the password, which will eventually enable you to enter Professor Atombender's laboratory. Occasionally you may also turn up 'snoozers', which are passwords that temporarily switch off the robots



To the programmer, platform games have several advantages. Primary among these is the relative lack of coding that is required. Once the programmer has defined the graphic design of the hero, guardians, treasure, platforms, ladders and lifts, all the different 'rooms' can be programmed merely by repositioning the various elements. This obviously saves a great deal of space, which can be used to enhance the detail of the game or increase the number of rooms. Additional space can be saved because each of the platforms does not have to be stored separately. The programmer has only to store the graphic design and listings, and the length of each of the platforms can be stored separately.

Impossible Mission takes full advantage of the platform game's ability to implement code in this way. The object of the game is to find the various pieces of a code — hidden in pieces of furniture in 32 rooms — which enables the player to break into the stronghold of the evil scientist, Elvin. The pieces of furniture are guarded by robots and occasionally by a large black ball. Also contained in some of the pieces of furniture are special passwords that, once obtained, enable the player to reset lifts within rooms and to temporarily switch off the robots so that they can be passed by in safety. These passwords are entered via the computer terminals placed in each room. The player moves from room to room via a lift and through a series of corridors. At the bottom left-hand corner of the screen is a map showing which rooms have been entered so far and the player's current position.

Passwords can also be obtained from code rooms. To receive a password, the room emits a number of notes of varying pitch, and the player has to place them in ascending order. The more passwords one attempts to obtain, the larger the number of notes that are played. In this respect, Impossible Mission is a standard platform game. Unlike most platform games, the player does not have a set number of 'lives' but, rather, each time the agent is 'killed' the player incurs a time penalty.

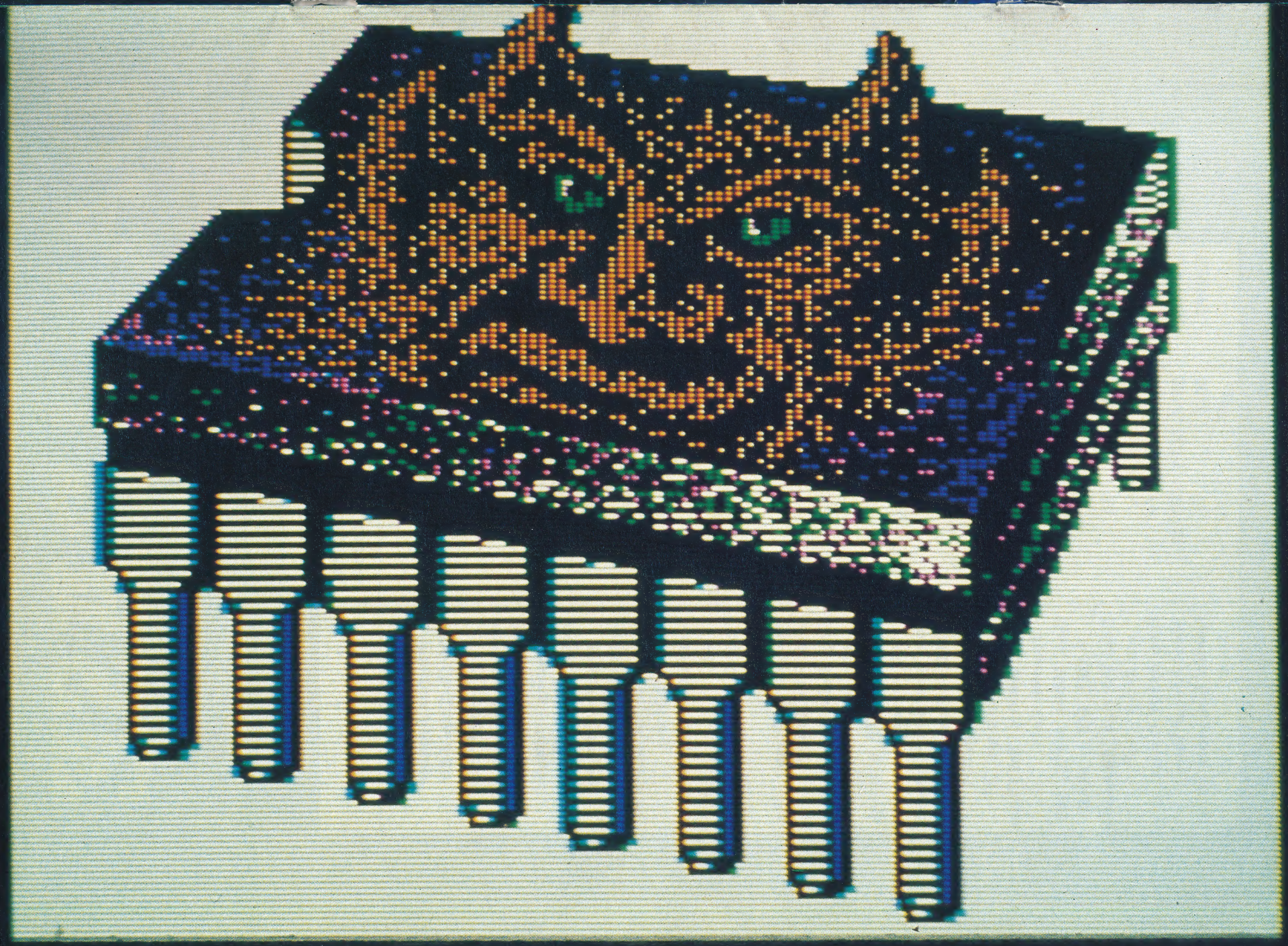
The memory space saved by the programmers is put to excellent use here. The most notable feature of this is the speech synthesis, which is notorious for eating up large quantities of memory — a couple of short spoken phrases of about 10 words each can take up to three or four Kbytes of RAM. This example is among the best and clearest so far heard on any game. The graphics are also very good, and although the rooms are not replete with them, the individual details of the agent, the furniture and the robots are very finely drawn. These various elements add up to a very well crafted and enjoyable game.

DATABASE

Here, courtesy of Commodore Business Machines, we publish the first instalment of a detailed analysis of the Commodore 64's memory map.

COMMODORE 64 MEMORY MAP

LABEL	HEX ADDRESS	DECIMAL LOCATION	DESCRIPTION
D6510	0000	0	6510 On-Chip Data-Direction Register
R6510	0001	1	6510 On-Chip 8-Bit Input/Output Register
	0002	2	Unused
ADRAY1	0003–0004	3–4	Jump Vector: Convert Floating—Integer
ADRAY2	0005–0006	5–6	Jump Vector: Convert Integer—Floating
CHARAC	0007	7	Search Character
ENDCHR	0008	8	Flag: Scan for Quote at End of String
TRMPOS	0009	9	Screen Column From Last TAB
VERCK	000A	10	Flag: 0 = Load, 1 = Verify
COUNT	000B	11	Input Buffer Pointer / No. of Subscripts
DIMFLG	000C	12	Flag: Default Array DIMension
VALTYP	000D	13	Data Type: $FF = String, $00 = Numeric
INTFLG	000E	14	Data Type: $80 = Integer, $00 = Floating
GARBFL	000F	15	Flag: DATA scan/LIST quote/Garbage Coll
SUBFLG	0010	16	Flag: Subscript Ref / User Function Call
INPFLG	0011	17	Flag: $00 = INPUT, $40 = GET, $98 = READ
TANSGN	0012	18	Flag: TAN sign / Comparison Result
	0013	19	Flag: INPUT Prompt
LINNUM	0014–0015	20–21	Temp: Integer Value
TEMPPT	0016	22	Pointer: Temporary String Stack



© 1983 JONES, L.