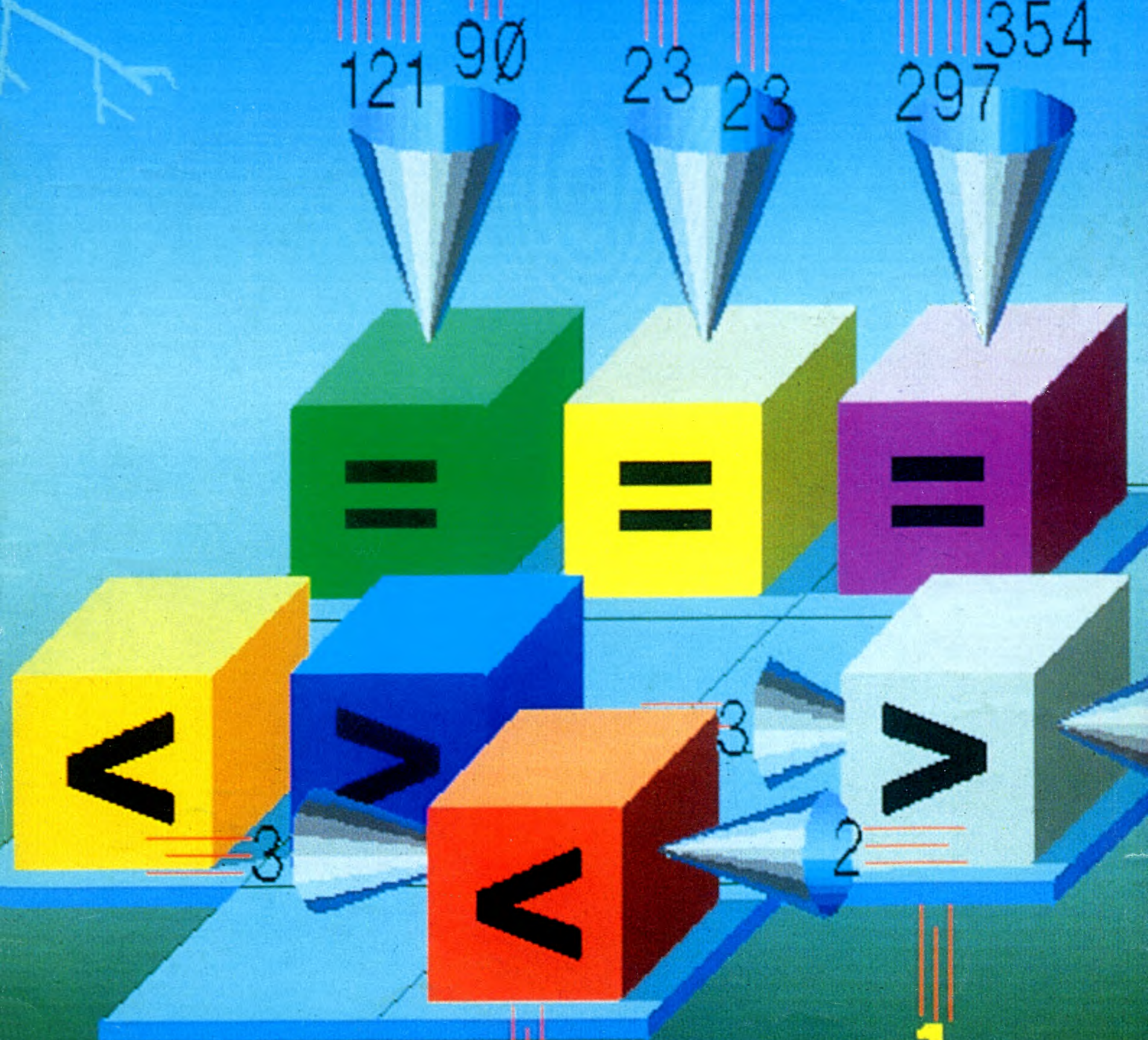
A MARSHALL CAVENDISH **49** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

INPUT

Vol. 4 **No 49**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Digital Arts. Pages 1525, 1526, 1528, 1530, Phil Dobson. Page 1530, Renumber routine, Commodore Computing International. Pages 1532, 1535, 1536, Digital Arts. Pages 1537, 1538, 1540, 1542, Gary Wing. Pages 1545, 1548, Graeme Harris/Chris Lyon. Pages 1553, 1555, 1556, Kevin O'Keefe. Page 1554, Rick Blakely, Stepper Motor, Impex, Richmond, Surrey.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.

There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

SPECTRUM/ COMMODORE TOOLKIT

- ADDING EXTRA COMMANDS
- RENUMBER ROUTINE
- AUTO LINE NUMBERING
- BLOCK DELETE
- OTHER COMMANDS

Make your life easier with this new set of programming tools. This machine code utility offers many routines to help you sort out your BASIC programs

Although all the computers covered in *INPUT* use the same BASIC language, you'll be well aware of the variations among the different dialects. In fact, it is very rare to find even a short program that will run on more than one computer. Sometimes it is just a matter of variations in the way the commands are used but often you'll find that many commands implemented on one computer do not exist at all on another. Many of the commands missed out are those which, while not exactly essential to programming, do make the programmer's life easier—including facilities such as renumber or auto line numbering or hex to decimal conversions, and so forth.

The toolkit program given here for the Spectrum and Commodore adds these commands and others, making it easier for you to use and program your computer. The Acorn, Dragon and Tandy do not need a toolkit such as this as they already have these commands in their standard BASIC.

The program is in machine code so it can remain in your computer at the same time as a BASIC program. The instructions for loading, saving and using the programs are given under the separate sections below.

The Spectrum toolkit offers eight new routines—renumber, block delete, bytes free, program length, auto line numbering, a tape cataloguer and hex to dec or dec to hex conversions. These are all called with RANDOMIZE USR followed by a number as described below. Once a routine is called, the program will prompt you for the relevant inputs such as the lines to be deleted, the number to be converted or whatever.

The toolkit uses several of the routines from the cross-referencer program so you will need to join these two programs together. All the instructions for doing this are contained within the toolkit program. So simply type in the program, RUN it and follow the prompts. The program will tell you if you have made any mistakes entering the DATA. These have to be corrected before you can save the machine code.

The program will save the combined toolkit and cross-referencer under the name "TOOLKIT" CODE. To load it in at the start of a programming session use:

```
CLEAR 63488
LOAD "TOOLKIT" CODE
```

The RENUMBER routine is called with RANDOMIZE USR 63489. You'll be prompted to enter the line increments—any number between 1 and 255.

To find the length of your program type:

```
RANDOMIZE USR 63889
```

To find the number of bytes free type:

```
RANDOMIZE USR 63860.
```

AUTO line numbering is called with RANDOMIZE USR 64154. The prompts ask you for the start line (1 to 9900) and the line increments (1 to 9900). To cancel AUTO enter two zeros after the line number appears. The routine will then stop with a 'nonsense in Basic' message, which you can ignore.

For BLOCK DELETE call RANDOMIZE USR 64000. The prompts ask you for the start and end line numbers of the section to be deleted.

For the TAPE CATALOGUE call RANDOMIZE USR 63919. The border will start flashing. Position your tape at the start of a program and press PLAY. Information from

the tape header is then displayed.

To convert from DEC to HEX type RANDOMIZE USR 64394 and to convert from HEX to DEC type RANDOMIZE USR 64453. (For the last routine, there is no need to press ENTER after the hex number.)

The Commodore toolkit adds a total of 43 new commands that can be used in the same way as the normal BASIC keywords.

Type in the program then RUN it to check for any mistakes. If you've made an error in the DATA lines the program will tell you in which line it occurred.

When the program is correct save the BASIC version using:

SAVE "TOOLKIT"

Then RUN it, type SYS 52480 as instructed to create the machine code, then save the machine code with one of the new commands

you've created:

@MSAVE 49152,53247"MCTOOLKIT",1,1

It is the machine code version that you'll need to load in when you want to use the program. To load it type:

LOAD "MCTOOLKIT",1,1

Now for the commands. Because of the way the program works, all of the new commands consist of an existing BASIC word plus one or two extra characters, and they all start with @. The syntax of each command is given followed by an explanation and an example where necessary.

THE NEW COMMANDS

@PPOKE location, address. Eg @PPOKE 51,16384 pokes the low and high bytes of 16384 into locations 51 and 52.
@PPEEK location. Eg @PPEEK 51 prints the value of PEEK(51) + PEEK(52)*256.
@POKER start address, number of items (0 to 255), list of items. Eg @POKER 49152,4,12,51,34,15,21 pokes the five data items into memory starting at location 49152.
@POKES start address, end address, value to be poked (0 to 255). This fills up a section of memory with the value stated. You can erase a section of memory by poking zeros, or you can fill part of the screen with characters by poking in the screen code for the character in the screen memory.
@BNEW address. This moves the start of BASIC to the address stated.
@GETNEW resets the start of BASIC to normal. This is equivalent to an OLD command. These last two commands can be used to store several programs in memory at the same time. For example, you can load in one program as usual, then move BASIC with, say, @BNEW 16384, and load in another program. @BNEW 2048:@GETNEW then returns you to the old program.
@NNEW performs a cold start, erasing all programs in memory.
@GGOTO variable allows you to GOTO a variable line number.
@LRESTORE line number lets you restore a READ command to a specific line.
@MGOTO start address, end address, new start moves the section of memory between the start and end addresses to the new start address.
@MREAD start address, end address prints out the contents of the section of memory and adds up the total of all the values. A typical result might be 12, 32, 65, 22, #131. This can be used to create DATA lines of machine code with checksums at the end of each line. There's a short program to demonstrate this at the end of the section.
@MSAVE start address, end address name of machine code program, device number, 1 saves a section of memory as a machine code program. As an example, see the command for saving the toolkit program in the first place.
@MFRE tells you the total amount of free memory available for BASIC.
@COST low byte, high byte. Eg @COST6,1 prints out the value of 6+1*256.
@POST number converts the number into its low and high bytes.
@CCHR$ number of characters (0 to 255), ASCII code to be printed. Eg @CCHR$ 6,65 prints out six letter As.
@D' decimal number converts to hexadecimal.
@H'$ four digit hex number converts to decimal.
@ONTO step of count (0 to 255) gives automatic line numbering in the step stated. The start line is whatever you type in first.
@ONTO RETURN turns off the previous @ONTO command. Press SHIFT plus RETURN before using this command.
@RLIST, start line number, step. This is the renumber command. Eg @RLIST,10,5 renumbers a program starting at line 10 in steps of five. Note the first comma in the command—you must put this in.
@WAITGET variable waits for a keypress, the name of the variable is unimportant.
@WAIT' 711*number of seconds delays for specified number of seconds. (The 711 is a scale factor.)
@PRINT% ink(0 to 15), X (0 to 39), Y (0 to 24), text to be printed. This prints the text in the ink colour at coordinates X,Y.
@COR ink (0 to 15), border colour (0 to 15), paper colour (0 to 15) sets up colour scheme.
@CLR' number (0 to 24) clears line specified on the screen.
@SCLR clears whole screen.
@SNEW resets screen.
@UP scrolls screen up one line.
@SYS1 turns screen off.
@SYS0 turns screen on.
@ASC1 forces lower case characters.
@ASC0 forces upper case characters.
@ON1 disables SHIFT.
@ON0 enables SHIFT.
@DEF1 disables RUN/STOP.
@DEF0 enables RUN/STOP.
@FN1 turns on auto repeat.
@FN0 turns off auto repeat.
@KCLR clears keyboard buffer (equivalent to POKE 198,0).
@TOP cursor home.
@SIF clears all sound chip registers.
@SON voice, volume, A/D, S/R, waveform, high byte of note, low byte of note. This can be used to set up the parameters for a note.

CREATING DATA LINES

As mentioned under @MREAD you can create a BASIC program consisting of machine code DATA by reading a section of memory. This command was used to create the listing of the toolkit itself. Assuming you've used an assembler to create some machine code starting at location 49152, the method is as follows. First enter these lines then press RETURN:

```
A=0:X=49152:@CCHR$79,32:?"■■□"
A*10"DATA□";:@MREAD X+A*15,14+
X+A*15
```

When you press RETURN you'll see the first DATA line printed on the screen. Run the cursor over the line and press RETURN to store the line.

Then move the cursor to the A=0 and increment the 0 to a 1, press RETURN and continue the process with the next DATA line. Keep incrementing A until all the DATA has been entered. This routine gives a program with line steps of 10. Change the figure 10 before the work "DATA " for different line steps.

BLOCK DELETE

There is no block delete command but this can be achieved using two other commands. Turn on auto repeat with @FN1 then type @ONTO with a suitable line step relating to the step between the lines you want to delete, and enter the first line number to be deleted. Holding down RETURN will effectively delete all lines from that point onwards, only stopping when you take your finger off RETURN.

```
5 CLEAR 63488: BORDER 0: PAPER 0: INK 6:
  CLS
10 PRINT AT 0,10; INVERSE 1;" TOOLKIT "
12 PRINT AT 8,2;"□Press any key to load
  cross-□□referencer machine code.":
  PAUSE 0
14 LOAD "CREF"CODE
15 CLS : PRINT "Poking TOOLKIT machine
  code.□□□□Please prepare a cassette for
  saving."
20 LET L=100: RESTORE L: FOR N=63489
  TO 64560 STEP 16
30 LET T=0: FOR D=0 TO 15
40 READ A: POKE (N+D),A: LET T=T+A:
  NEXT D
50 READ A: IF A<>T THEN PRINT
  "CHECKSUM ERROR IN LINE□";L: STOP
60 LET L=L+10
70 NEXT N
100 DATA 62,12,205,48,252,205,60,250,237,
```

```
   67,155,248,42,83,92,1,2019
110 DATA 0,0,126,254,128,40,9,197,205,184,
   25,193,3,235,24,242,1865
120 DATA 205,43,45,58,155,248,205,40,45,
   239,4,56,205,162,45,33,1788
130 DATA 15,39,167,237,66,48,2,207,5,33,
   145,248,126,60,40,32,1470
140 DATA 35,229,237,91,83,92,42,75,92,167,
   237,82,68,77,235,237,2079
150 DATA 177,197,229,245,204,157,248,241,
   225,193,234,80,248,225,24,220,3147
160 DATA 42,83,92,58,155,248,54,0,35,119,
   205,40,45,239,49,192,1656
170 DATA 56,42,83,92,205,184,25,42,75,92,
   43,167,237,82,216,235,1876
180 DATA 229,239,224,15,49,56,205,162,45,
   225,112,35,113,43,24,228,2004
190 DATA 201,224,228,235,236,239,246,255,
   0,0,10,0,229,6,4,35,2148
200 DATA 126,254,14,40,4,16,248,225,201,
   197,35,35,35,78,35,70,1613
210 DATA 42,83,92,217,1,1,0,217,205,149,22,
   43,235,167,237,66,1777
220 DATA 235,48,11,217,3,217,197,205,184,
   25,193,235,24,234,217,197,2442
230DATA217,209,42,155,248,205,169,48,235,
   42,104,92,35,35,115,35,1986
240 DATA 114,235,62,0,167,1,9,0,237,66,56,
   17,60,1,90,0,1115
250 DATA 237,66,56,9,60,1,132,3,237,66,56,
   1,60,209,225,229,1647
260 DATA 245,130,214,4,245,6,0,56,9,79,40,
   12,205,85,22,35,1387
270 DATA 24,6,237,68,79,205,232,25,193,241,
   197,79,6,0,9,65,1666
280 DATA 229,197,35,35,235,42,104,92,1,5,0,
   237,176,193,225,229,2035
290 DATA 197,239,224,164,5,58,193,164,4,
   224,1,3,225,192,2,56,1951
300 DATA 205,213,45,193,225,198,48,119,43,
   16,228,229,42,104,92,35,2035
310 DATA 35,126,225,198,48,119,241,193,
   245,42,83,92,167,229,237,66,2346
320 DATA 225,48,8,197,205,184,25,193,235,
   24,242,235,35,35,193,126,2210
330 DATA 128,119,201,62,0,205,48,252,205,
   26,31,33,0,0,62,0,1372
340 DATA 237,66,229,193,205,43,45,62,254,
   205,1,22,205,227,45,201,2240
350 DATA 42,75,92,237,75,83,92,62,0,237,66,
   229,62,1,205,48,1606
360 DATA 252,193,205,43,45,205,227,45,62,
   2,205,48,252,201,221,33,2239
370 DATA 32,255,17,17,0,175,55,205,86,5,62,
   3,205,48,252,221,1638
380 DATA 33,32,255,221,126,0,198,6,221,229,
   205,48,252,62,13,215,2116
390 DATA 62,4,205,48,252,221,225,221,35,
   221,126,0,254,255,40,10,2179
400 DATA 6,10,221,126,0,221,35,215,16,248,
   62,13,215,62,5,205,1660
410 DATA 48,252,221,33,32,255,221,78,11,
   221,70,12,195,163,249,205,2266
420 DATA 142,250,62,10,205,48,252,205,60,
   250,205,110,25,229,62,13,2128
430 DATA 215,62,11,205,48,252,205,60,250,
   205,110,25,193,32,16,229,2118
440 DATA 35,35,126,35,95,126,87,225,237,90,
   17,4,0,237,90,229,1668
450 DATA 197,62,0,237,66,218,87,252,195,89,
   252,6,0,197,205,95,2158
460 DATA 252,205,115,252,193,254,13,40,26,
   254,58,48,240,214,48,56,2268
470 DATA 236,245,4,120,254,6,32,4,5,241,24,
   225,241,245,198,48,2128
480 DATA 215,24,218,221,33,49,255,120,254,
   0,40,207,33,0,0,221,1890
490 DATA 94,0,221,86,1,241,254,0,40,7,237,
   90,56,13,61,32,1433
500 DATA 249,221,35,221,35,5,32,231,229,
   193,201,207,5,42,75,92,2073
510 DATA 237,75,83,92,237,66,192,207,9,62,
   10,205,48,252,205,60,2040
520 DATA 250,34,30,255,62,13,215,62,12,
   205,48,252,205,60,250,34,1987
530 DATA 28,255,33,48,48,34,59,255,34,61,
   255,237,75,30,255,205,1912
540 DATA 115,251,62,2,50,107,92,50,107,92,
   205,149,23,205,176,22,1708
550 DATA 62,0,205,1,22,33,59,255,6,4,126,
   229,197,205,129,15,1548
560 DATA 193,225,35,16,245,205,44,15,205,
   23,27,221,33,58,92,221,1858
570 DATA 203,0,126,32,13,42,89,92,205,167,
   17,62,255,50,58,92,1503
580 DATA 24,206,42,89,92,34,93,92,205,251,
   25,120,177,32,10,223,1715
590 DATA 254,13,40,174,205,176,22,207,1,
   237,67,73,92,42,93,92,1788
600 DATA 235,33,85,21,229,42,97,92,55,237,
   82,229,96,105,205,110,1953
610 DATA 25,32,6,205,184,25,205,232,25,
   193,121,61,176,40,47,197,1774
620 DATA 3,3,3,3,43,237,91,83,92,213,205,
   85,22,225,34,83,1425
630 DATA 92,193,197,19,42,97,92,43,43,237,
   184,42,73,92,235,193,1874
640 DATA 112,43,113,43,115,43,114,237,75,
   28,255,205,115,251,241,195,2185
650 DATA 195,250,33,62,255,126,60,254,58,
   40,8,119,11,121,128,176,1896
660 DATA 200,24,239,62,48,119,43,24,236,
   62,14,205,48,252,205,60,1841
670 DATA 250,62,13,215,197,62,15,205,48,
   252,193,46,2,96,124,203,1983
680 DATA 31,203,31,203,31,203,31,230,15,
   205,189,251,215,124,230,15,2207
690 DATA 205,189,251,215,97,45,32,230,62,
   13,215,201,198,48,254,58,2313
700 DATA 216,198,7,201,62,16,205,48,252,
   17,85,255,6,4,213,197,1982
710 DATA 205,95,252,205,115,252,215,245,
   241,193,209,18,19,16,239,62,2581
720 DATA 13,215,62,17,205,48,252,221,33,85,
   255,17,0,16,33,0,1472
730 DATA 0,14,4,221,126,0,221,35,214,48,
   218,87,252,254,10,56,1760
740 DATA 2,214,7,254,16,210,87,252,71,254,
   0,40,3,25,16,253,1704
750 DATA 203,58,203,27,203,58,203,27,203,
   58,203,27,203,58,203,27,1964
760 DATA 13,32,208,229,193,205,43,45,205,
   227,45,62,13,215,201,203,2139
800 CLS : PRINT AT 5,5;"□
   COMPILATION COMPLETE.□"
810 PRINT AT 7,2;"PREPARE A CASSETTE
   FOR SAVING."
820 PRINT AT 9,4;"FILENAME IS "
   "TOOLKIT""□CODE "
830 SAVE "TOOLKIT"CODE 63489,2000
```

```
100 DATA 32,158,183,142,134,2,32,253,174,
   32,158,183,138,72,32, #1725
101 DATA 253,174,32,158,183,104,168,24,32,
   240,255,32,253,174,32, #2114
102 DATA 164,170,96,32,158,183,142,134,2,
   32,253,174,32,158,183, #1913
103 DATA 142,32,208,32,253,174,32,158,183,
   142,33,208,96,32,247, #1972
104 DATA 183,32,253,174,32,235,183,142,19,
   3,169,0,133,2,32, #1592
105 DATA 253,174,32,158,183,138,164,2,145,
   20,204,19,3,240,5, #1740
106 DATA 230,2,76,74,192,96,32,138,173,32,
   247,183,165,20,133, #1793
107 DATA 251,165,21,133,252,32,253,174,32,
   138,173,32,247,183,32, #2118
108 DATA 253,174,32,158,183,134,2,165,21,
   197,252,144,35,208,6, #1964
109 DATA 165,20,197,251,144,27,165,2,160,
   0,145,251,165,251,197, #2140
110 DATA 20,208,6,165,252,197,21,240,9,
   230,251,208,234,230,252, #2523
111 DATA 76,141,192,96,32,138,173,32,247,
   183,165,20,133,251,165, #2044
112 DATA 21,133,252,32,253,174,32,138,173,
   32,247,183,165,20,133, #1988
113 DATA 253,165,21,133,254,32,253,174,32,
   138,173,32,247,183,165, #2255
114 DATA 254,197,252,144,41,208,6,165,253,
   197,251,144,33,160,0, #2305
115 DATA 177,251,145,20,165,251,197,253,
   208,6,165,252,197,254,240, #2781
116 DATA 15,230,251,208,2,230,252,230,20,
   208,228,230,21,76,223, #2424
117 DATA 192,96,32,138,173,32,247,183,165,
   20,133,251,165,21,133, #1981
118 DATA 252,32,253,174,32,138,173,32,247,
   183,160,0,165,20,145, #2006
119 DATA 251,165,21,200,145,251,96,32,158,
   183,32,255,233,96,32, #2150
120 DATA 138,173,32,247,183,160,0,177,20,
```

```
170,200,177,20,32,205,#1934
121 DATA 189,96,32,138,173,32,247,183,165,
    20,133,251,165,21,133,#1978
122 DATA 252,32,253,174,32,138,173,32,247,
    183,165,21,197,252,144,#2295
123 DATA 61,208,6,165,20,197,251,144,53,
    169,0,133,253,133,254,#2047
124 DATA 24,160,0,177,251,170,101,253,133,
    253,165,254,105,0,133,#2179
125 DATA 254,169,0,32,205,189,169,44,32,
    210,255,165,251,197,20,#2192
126 DATA 208,6,165,252,197,21,240,9,230,
    251,208,214,230,252,76,#2559
127 DATA 104,193,169,35,32,210,255,166,
    253,165,254,76,205,189,32,#2338
128 DATA 138,173,32,247,183,162,0,232,208,
    253,198,20,169,255,197,#2467
129 DATA 20,208,245,198,21,197,21,208,239,
    96,32,158,183,160,0,#1986
130 DATA 224,1,208,2,160,7,224,2,208,2,
    160,14,132,2,32,#1378
131 DATA 253,174,32,158,183,142,24,212,32,
    253,174,32,158,183,138,#2148
132 DATA 164,2,153,5,212,32,253,174,32,158,
    183,138,164,2,153,#1825
133 DATA 6,212,32,253,174,32,158,183,138,
    164,2,153,4,212,142,#1865
134 DATA 19,3,32,253,174,32,158,183,138,
    164,2,153,1,212,32,#1556
135 DATA 253,174,32,158,183,138,164,2,153,
    0,212,206,19,3,173,#1870
136 DATA 19,3,164,2,153,4,212,96,162,0,138,
    157,0,212,232,#1554
137 DATA 224,25,208,248,96,76,68,229,76,24,
    229,76,234,232,169,#2214
138 DATA 0,141,138,2,96,169,128,141,138,2,
    96,169,0,133,198,#1551
139 DATA 96,169,237,141,40,3,96,169,251,
    141,40,3,96,76,102,#1660
140 DATA 229,169,27,141,17,208,96,169,11,
    141,17,208,96,169,21,#1719
141 DATA 141,24,208,96,169,23,141,24,208,
    96,169,9,76,210,255,#1849
142 DATA 169,8,76,210,255,32,138,173,32,
    247,183,76,163,168,32,#1962
143 DATA 138,173,32,247,183,169,0,168,145,
    20,24,165,20,105,1,#1590
144 DATA 133,43,165,21,105,0,133,44,76,154,
    227,169,62,32,210,#1574
145 DATA 255,169,18,32,210,255,165,55,56,
    229,45,170,165,56,229,#2109
146 DATA 46,32,205,189,169,96,160,228,76,
    30,171,169,0,133,198,#1902
147 DATA 165,198,201,1,208,250,76,146,171,
    169,8,160,1,145,43,#1942
148 DATA 32,51,165,24,165,34,105,2,133,45,
    133,47,133,49,165,#1283
149 DATA 35,105,0,133,46,133,48,133,50,96,
    32,138,173,32,247,#1401
150 DATA 183,165,20,133,63,165,21,133,64,
    32,19,166,56,165,95,#1480
151 DATA 233,1,133,65,165,96,233,0,133,66,
    96,162,0,181,43,#1607
152 DATA 149,251,232,224,4,208,247,32,138,
    173,32,247,183,165,20,#2305
153 DATA 133,43,165,21,133,44,32,253,174,
    32,138,173,32,247,183,#1803
154 DATA 165,20,133,45,165,21,133,46,32,86,
    225,162,0,181,251,#1665
155 DATA 149,43,232,224,4,208,247,96,32,
    158,183,134,2,32,253,#1997
156 DATA 174,32,158,183,142,19,3,165,2,201,
    0,240,11,173,19,#1522
157 DATA 3,32,210,255,198,2,76,79,195,96,
    76,154,227,32,138,#1773
158 DATA 173,32,247,183,170,169,72,32,210,
    255,169,39,32,210,255,#2248
159 DATA 169,36,32,210,255,138,32,139,195,
    138,32,144,195,152,32,#1899
160 DATA 139,195,152,32,144,195,96,24,106,
    106,106,106,41,15,24,#1481
161 DATA 105,48,201,58,144,2,105,6,32,210,
    255,96,169,68,32,#1531
162 DATA 210,255,169,39,32,210,255,32,186,
    195,133,34,32,186,195,#2163
163 DATA 170,165,34,32,205,189,76,228,167,
    32,203,195,10,10,10,#1726
164 DATA 10,133,35,32,203,195,101,35,133,
    35,96,32,115,0,201,#1356
165 DATA 58,41,15,144,2,105,8,96,32,138,
    173,32,247,183,169,#1443
166 DATA 91,32,210,255,169,0,166,20,32,
    205,189,169,44,32,210,#1824
167 DATA 255,169,0,166,21,32,205,189,169,
    93,32,210,255,96,32,#1924
168 DATA 158,183,134,2,32,253,174,32,158,
    183,138,166,2,76,205,#1896
169 DATA 189,5,0,0,32,121,0,208,6,169,0,
    141,14,196,96,#1177
170 DATA 169,1,141,14,196,169,53,141,4,3,
    169,196,141,5,3,#1405
171 DATA 32,138,173,32,247,183,165,20,141,
    12,196,165,21,141,13,#1679
172 DATA 196,96,173,0,2,201,48,144,59,201,
    58,176,55,173,14,#1596
173 DATA 196,240,50,32,124,165,132,2,173,
    12,196,24,101,20,133,#1600
174 DATA 99,173,13,196,101,21,133,98,162,
    144,56,32,73,188,32,#1521
175 DATA 223,189,133,254,132,255,160,0,
    177,254,240,6,153,119,2,#2297
176 DATA 200,208,246,132,198,164,2,96,76,
    124,165,0,0,0,0,#1611
177 PRINT"":X=49152:C=76:
    GOSUB 200
178 DATA 169,11,141,8,3,169,205,141,9,3,96,
    32,115,0,201,#1303
179 DATA 64,240,3,76,231,167,160,1,177,122,
    133,255,160,2,177,#1968
180 DATA 122,133,2,162,0,189,128,206,197,
    255,208,9,232,189,128,#2160
181 DATA 206,197,2,240,16,202,201,0,240,6,
    232,232,224,128,208,#2334
182 DATA 230,162,11,108,0,3,134,2,32,115,0,
    32,115,0,32,#976
183 DATA 115,0,166,2,189,255,205,133,252,
    189,0,206,133,253,169,#2267
184 DATA 76,133,251,32,251,0,76,174,167,96,
    0,0,0,0,0,#1256
185 X=52480:C=6:GOSUB 200
186 DATA 0,192,33,192,64,192,96,192,169,
    192,1,193,36,193,43,#1788
187 DATA 193,61,193,163,193,189,193,36,194,
    48,194,51,194,54,194,#2150
188 DATA 57,194,63,194,69,194,74,194,80,
    194,86,194,89,194,95,#1971
189 DATA 194,101,194,107,194,113,194,118,
    194,123,194,132,194,159,194,#2405
190 DATA189,194,202,194,233,194,8,195,65,
    195,97,195,100,195,159,#2415
191 DATA 195,215,195,251,195,15,196,73,197,
    0,0,0,0,0,0,#1532
192 X=52736:C=5:GOSUB 200
193 DATA 153,37,67,176,151,82,151,83,77,
    137,80,151,156,39,80,#1620
194 DATA 194,77,135,146,39,83,145,83,139,
    83,156,83,162,85,80,#1690
195 DATA 165,48,165,49,75,156,150,48,150,
    49,164,80,158,48,158,#1663
196 DATA 49,198,48,198,49,145,48,145,49,71,
    137,66,162,77,184,#1626
197 DATA 146,161,161,162,76,140,77,148,67,
    199,78,162,68,39,72,#1756
198 DATA 39,185,84,190,84,145,164,82,155,0,
    0,0,0,0,0,#1128
199 X=52864:C=5:GOSUB 200:GOTO 205
200 FOR Z=0 TO C:T=0:FOR ZZ=0 TO
    14:READ M:POKE X,M
201 PRINT"LINE"PEEK(63)+PEEK(64)*
    256:T=T+M:X=X+1
202 NEXT ZZ:READ X$:IF VAL(RIGHT$(X$,
    LEN(X$)-1))<>T THEN 204
203 NEXT Z:PRINT"OK":RETURN
204 PRINT"ERROR IN LINE!":END
205 K=50505:T=0
206 READA:IFA=-1THEN209
207 POKEK,A:K=K+1
208 T=T+A:GOTO206
209 IFT<>52549THENPRINT"
    CHECKSUM ERROR":END
210 IFK<>50928 THENPRINT
    "NUMBER OF VALUES ERROR":END
211 PRINT"USE SYS 52480 TO EXECUTE
    MACHINE CODE"
212 END
213 DATA 32,253,174,32,107,169,165
214 DATA 20,133,53,165,21,133,54
215 DATA 32,253,174,32,107,169,165
216 DATA 20,133,49,165,21,133,50
217 DATA 32,142,166,32,201,198,32
218 DATA 201,198,208,33,32,2,198
219 DATA 32,201,198,32,201,198,208
220 DATA 3,76,212,198,32,201,198
```

```
221 DATA 165,99,145,122,32,201,198
222 DATA 165,98,145,122,32,13,198
223 DATA 240,226,32,201,198,32,201
224 DATA 198,32,201,198,201,34,208
225 DATA 11,32,201,198,240,197,201
226 DATA 34,208,247,240,238,170,240
227 DATA 188,16,233,162,4,221,235
228 DATA 198,240,5,202,208,248,240
229 DATA 221,165,122,133,59,165,123
230 DATA 133,60,32,115,0,176,211
231 DATA 32,107,169,32,32,198,165
232 DATA 60,133,123,165,59,133,122
233 DATA 160,0,162,0,189,0,1
234 DATA 240,17,72,32,115,0,144
235 DATA 3,32,82,198,104,160,0
236 DATA 145,122,232,208,234,32,115
237 DATA 0,176,8,32,97,198,32
238 DATA 121,0,144,248,201,44,240
239 DATA 186,208,152,165,53,133,99
240 DATA 165,54,133,98,76,142,166
241 DATA 165,99,24,101,49,133,99
242 DATA 165,98,101,50,133,98,32
243 DATA 201,198,208,251,96,32,2
244 DATA 198,32,201,198,32,201,198
245 DATA 208,8,169,255,133,99,133
246 DATA 98,48,14,32,201,198,197
247 DATA 20,208,16,32,201,198,197
248 DATA 21,208,12,162,144,56,32
249 DATA 73,188,76,223,189,32,201
250 DATA 198,32,13,198,240,209,32
251 DATA 114,198,230,251,32,165,198
252 DATA 230,45,208,2,230,46,96
253 DATA 32,114,198,198,251,32,141
254 DATA 198,165,45,208,2,198,46
255 DATA 198,45,96,32,124,198,160
256 DATA 0, 132, 17, 132, 251, 96, 165
257 DATA 122, 133, 34, 165, 123, 133, 35
258 DATA 165,45,133,36,165,46,133
259 DATA 37,96,164,17,200,177,34
260 DATA 164,251,200,145,34,32,190
261 DATA 198,208,1,96,230,34,208
262 DATA 236,230,35,208,232,164,17
263 DATA 177,36,164,251,145,36,32
264 DATA 190,198,208,1,96,165,36
265 DATA 208,2,198,37,198,36,76
266 DATA 165,198,165,34,197,36,208
267 DATA 4,165,35,197,37,96,160
268 DATA 0,230,122,208,2,230,123
269 DATA 177,122,96,32,51,165,165
270 DATA 34,166,35,24,105,2,133
271 DATA 45,144,1,232,134,46,32
272 DATA 89,166,76,116,164,0,137
273 DATA 138,141,167,−1
```

THE FORTH DIMENSION

So far, you have seen how to use the basic words in FORTH and how to store information. But in order to build up a useful program, you need to be able to build up structures...

In this final part of the series of articles on FORTH, we take a look at some of the operations you may need to use when writing a FORTH program.

FORTH provides control switches which enable conditional (comparison) and program looping operations in much the same way as in BASIC. Most of these structures use logic tests, and comparisons.

COMPARISONS

Comparisons in a FORTH routine are made by a familiar-looking set of *logical operators* (similar to >, <, = and so on in BASIC) which set up a logical value termed a *flag*.

Flags are used extensively in FORTH and are actually numeric values placed on the stack to show the outcome of a test. The flag value is a zero (Ø) if the outcome of a particular comparison is *false*, and a non-zero value (usually, but not always, 1,) if it's *true*.

A typical FORTH routine can be made to return a condition—true or false—to indicate whether or not something has taken place or whether a certain value has been reached, perhaps within a specified range. The value of the flag may itself be tested by program branches and loops (see below), so regulating the subsequent program flow.

Out-and-out comparisons are possible by comparing the topmost (first) value on the stack with, typically, the second value. The comparison operators that test a relationship between two stack items take the general form:

<first value> <second value>
COMPARISON

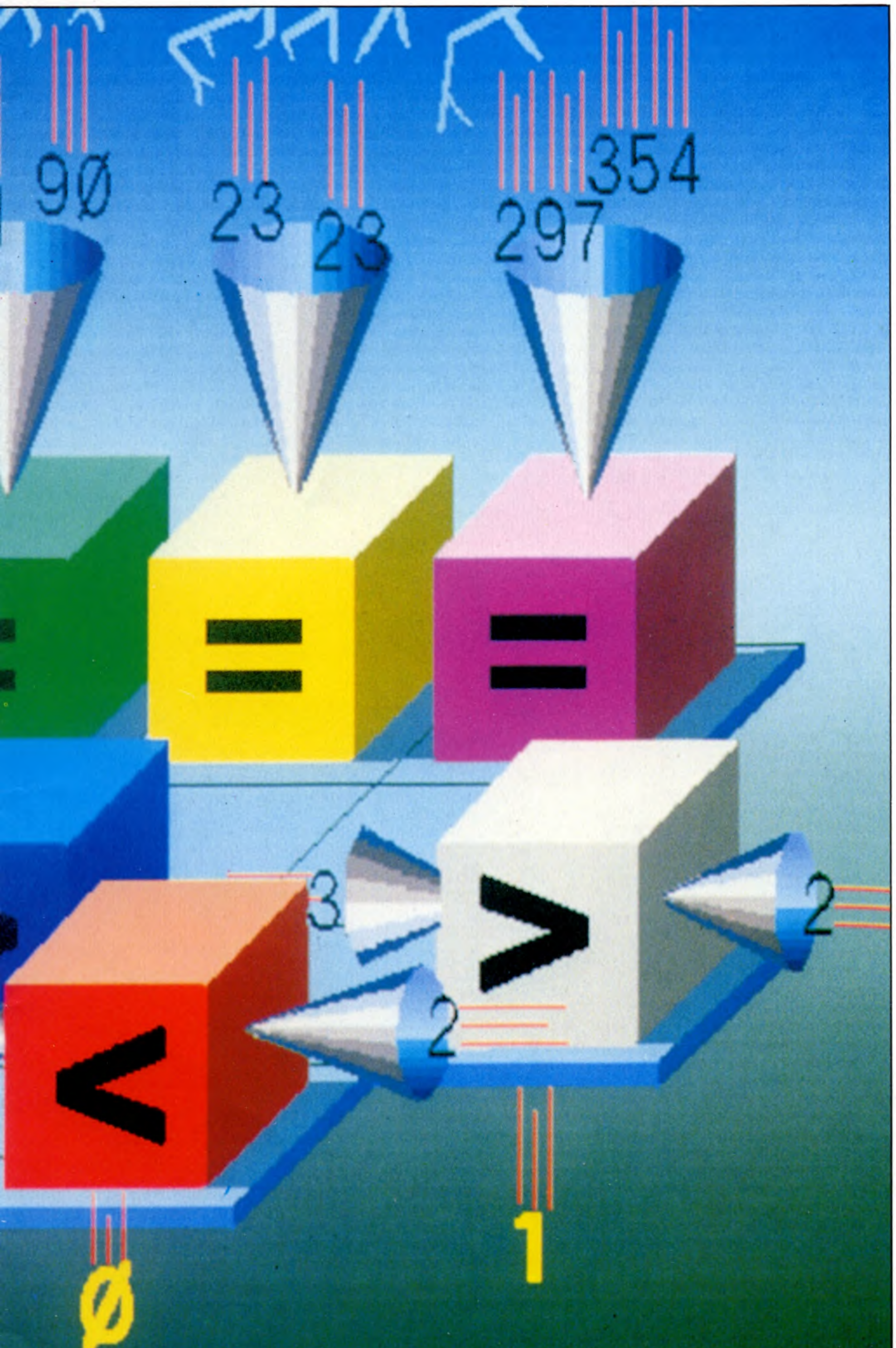
So let's look at the comparison operations possible. Alongside each word is the 'before and after' stack notation, and below this the name and description of the word, followed by simple 'true' or 'false' examples:

Word/Purpose	*Before/After Example*
<	n1 n2 — — f
less-than: leaves a true flag if n1 is less than n2, otherwise leaves a false flag	5 56 < . 1 OK 12 4 < . Ø OK
>	n1 n2 — — f
greater-than: leaves a true flag if n1 is greater then n2, otherwise leaves a false flag	12 4 > . 1 OK 9 9 > . Ø OK
=	n1 n2 — — f
equals: leaves a true flag if n1 equals n2, otherwise leaves a false flag	9 9 = . 1 OK 7 9 = . Ø OK
Ø<	n — — — — f
zero-less: leaves a true flag if the n is less than zero (negative), otherwise leaves a false flag	−5 Ø< . 1 OK 64 Ø< . Ø OK
Ø=	n — — — f
zero-equals: leaves a true flag if n is equal to zero, else leaves a false flag. Can be used as a NOT logical function	Ø Ø= . 1 OK 1 Ø= . Ø OK
Ø>	n— — — f
Leaves a true flag if n is greater than zero. Some implementations require you use the new colon definition (word) : Ø> MINUS Ø< :	37 Ø> . 1 OK −45 Ø> . Ø OK

ORDER OF ENTRY

Note that the operands and comparison operators are entered in the same order as they would be for a mathematical operation. When a FORTH routine encounters the words < = and > the top two values are removed from the stack and the relevant comparison operation is performed. Subsequently a true or false flag is pushed onto the stack and the two values are discarded.

The zero comparisons pull only the topmost value off the stack and compare this with zero, returning a 1 or Ø to the stack depending on the outcome of the comparison. The words < = > Ø< Ø= and Ø> assume—and are used for—*signed* single words for testing the relationship between bigger signed and unsigned double precision integers and these include: DØ= D< D= D> and DU<.



LOGICAL OPERATIONS

A full range of logical operations very similar to those carried out in any other language are also possible in FORTH. (See pages 284 to 288 for explanations of bitwise logical operations in BASIC.)

Some FORTH implementations do not support NOT, which is identical to Ø=, so the latter is usually used rather than wasting dictionary space on an unnecessary definition.

Each of the other logical operators—AND, OR, and XOR—pulls the top two values from the stack before executing the logical operation. This is performed in a bit-by-bit fashion, as normal (see page 288):

Word/Purpose	*Before/After Example*
AND Leaves the logical bitwise AND result of n1 and n2	n1 n2 —— n3 1 1 AND . 1 OK 1 Ø AND . Ø OK Ø Ø AND . Ø OK
OR Leaves the result of bitwise OR between n1 and n2	n1 n2 —— n3 1 Ø OR . 1 OK 1 1 OR . 1 OK Ø Ø OR . Ø OK
XOR Leaves the result of an exclusive—or of n1 and n2	n1 n2 —— n3 1 Ø XOR . 1 OK 1 1 XOR . Ø OK Ø Ø XOR . Ø OK

As in BASIC (and others), the logical operators may be used for masking operations to switch bit values in memory selectively.

BRANCHES AND LOOPS

FORTH uses three important command sequences which amplify the structured nature of this language.

One of the most powerful is the DO ——— LOOP which is similar to BASIC's FOR ... NEXT loop and is used in any routine where a sequence of steps is to be repeated a fixed number of times.

A LOOP in FORTH is a series of commands to be executed repetitively. The loop is set up with a starting value, an end value and the desired increment for each *iteration* or pass—these all form what is called the *body* of the loop definition. The value that changes upon each iteration is called the *index* or the *control variable* of the loop, and the ending value the *limit* which, when reached, causes the routine to terminate, or exit the loop.

A DO——— LOOP definition thus takes the form:

```
: <routine name> <limit +1> <start
   value> DO <chosen code> LOOP ;
```

A value *1 greater than* the desired loop limit is pushed onto the stack, followed by the chosen value, then the loop is entered. A value 1 greater has to be used because the index is incremented before it is compared with the specified limit.

Suppose, for example, you wanted to print out the character set. The DO——— LOOP for this takes the form:

```
: SET 90 65 DO I EMIT LOOP:
```

The code used between DO and LOOP here makes use of the FORTH word I which fetches the current index of the loop and pushes this onto the stack. EMIT is an output word covered in an earlier article (see page 1510).

By executing SET, upper case portions of the ASCII code, starting at 65 (ASCII code for A) and ending at 90 (ASCII code for Z) are displayed.

The equivalent BASIC code for this DO——— LOOP is:

```
FOR I=65 TO 90: PRINT CHR$(I): NEXT I
```

IF——ELSE——THEN

The main tool for conditional branching, as in BASIC is the conditional IF——ELSE——THEN routine—often used in conjunction with the DO ... LOOP. These three FORTH words are used only in a specific colon definition in the format:

```
<condition> IF <truepart> THEN
   <continuation>
```

or

```
<condition> IF <truepart> ELSE
   <falsepart> THEN <continuation>
```

When the word which defines this routine is executed it first tests the condition (flag) left on the top of the stack. IF takes the flag and causes the routine to branch to the relevant machine code. If the flag is true (that is, it is non-zero), execution continues through the true part of the definition—the code between IF and ELSE in the original definition. If the flag is false (0), execution skips to the false part—between ELSE and THEN or just after the THEN part. In both cases, execution continues from this point onwards.

As in the versions of BASIC that support it, ELSE is optional. If it is not present then a pure IF —— THEN condition exists. In this case the execution run skips over the conditional routine if a false value is returned.

INDEFINITE LOOPS

Quite often you come across a situation in which the number of iterations is not known before execution of a loop routine. Special looping facilities are provided for these *indefinite loops*, corresponding to the 'do while' and 'repeat until' commands of languages that support structured programming.

The simplest is BEGIN —— UNTIL which takes the form:

```
BEGIN <loop conditions and code> <flag>
   UNTIL
```

Here the flag is tested just *before* UNTIL (so that the loop code is always executed at least once). If the flag returns false, execution loops back to just after the BEGIN part. If it returns true, execution is directed to after the UNTIL part.

A simple example which performs a key test to see if Y has been pressed is:

```
: KTEST BEGIN        (start loop)
        GET          (read keyboard)
        89 = UNTIL ; (ASCII value
                      of Y=89)
```

So until key Y is pressed, a false condition is returned and the program effectively waits until a true situation occurs.

BEGIN WHILE REPEAT

A variation, BEGIN —— WHILE —— REPEAT forms a structure which loops indefinitely until a specified condition is met, whereupon the routine is exited. This takes the format:

```
BEGIN <first part> <flag> WHILE
   <second part> REPEAT
```

The first part of the routine code is executed—if it exists—as soon as this loop structure is entered. If the flag returns a true report the second part is executed and execution returns to just after the BEGIN. If a false report results, the program leaves the loop and execution continues with the code after REPEAT.

The 'first part' is usually the conditional part of the loop. WHILE actually tests the value on top of the stack.

NESTING

As with FOR ... NEXT loops in BASIC, DO ——— LOOP and IF —— ELSE —— THEN routines can be nested. An example of the format for a DO ——— LOOP is:

```
: <routine name> <limit +1> <start
   value>
DO
<second limit +1> <start value>
   DO <chosen code>
   LOOP
LOOP :
```

In normal circumstances, the index is incremented on each pass through the loop, but other values—including negatives to reverse the direction—may be introduced using the word +LOOP. This takes the top stack value and adds that to the loop index. The latter is compared against the limit.

The nesting of an IF —— ELSE —— THEN routine can be shown thus:

```
IF          <*************
   <a>                   *
   <b>                   *
   IF       <*********   *
     <c>             * (outer)
     <d>             *   *
   ELSE         (inner)* *
     <e>             *   *
     <f>             *   *
   THEN     <*********   *
ELSE                     *
   <g>                   *
   <h>                   *
THEN        <*************
```

You will often see screen source code for colon definitions laid out in this form, with appro-

Q+A

Do any of the versions of FORTH now available have graphics facilities?

There are versions available both for the Commodore 64 and the Acorn which support graphics facilities. However, because FORTH is designed to deal with text and numbers and also to be totally transportable, any facility for graphics will only be usable on the machine for which the implementation was designed and cannot be used on any other micro. These facilities are much quicker than BASIC.

priate annotation in brackets, just so that the level of nesting can be seen clearly.

The various looping and branching routines may themselves be nested in each other. The allowable configurations include the following:

```
BEGIN —— IF —— ELSE —— THEN ——
   UNTIL
DO —— BEGIN —— WHILE —— REPEAT —
   — LOOP
IF —— DO —— BEGIN —— UNTIL ——
   LOOP —— THEN
```

Whatever arrangement is used each structure must be fully contained within another: the standard 'rule' about nesting!

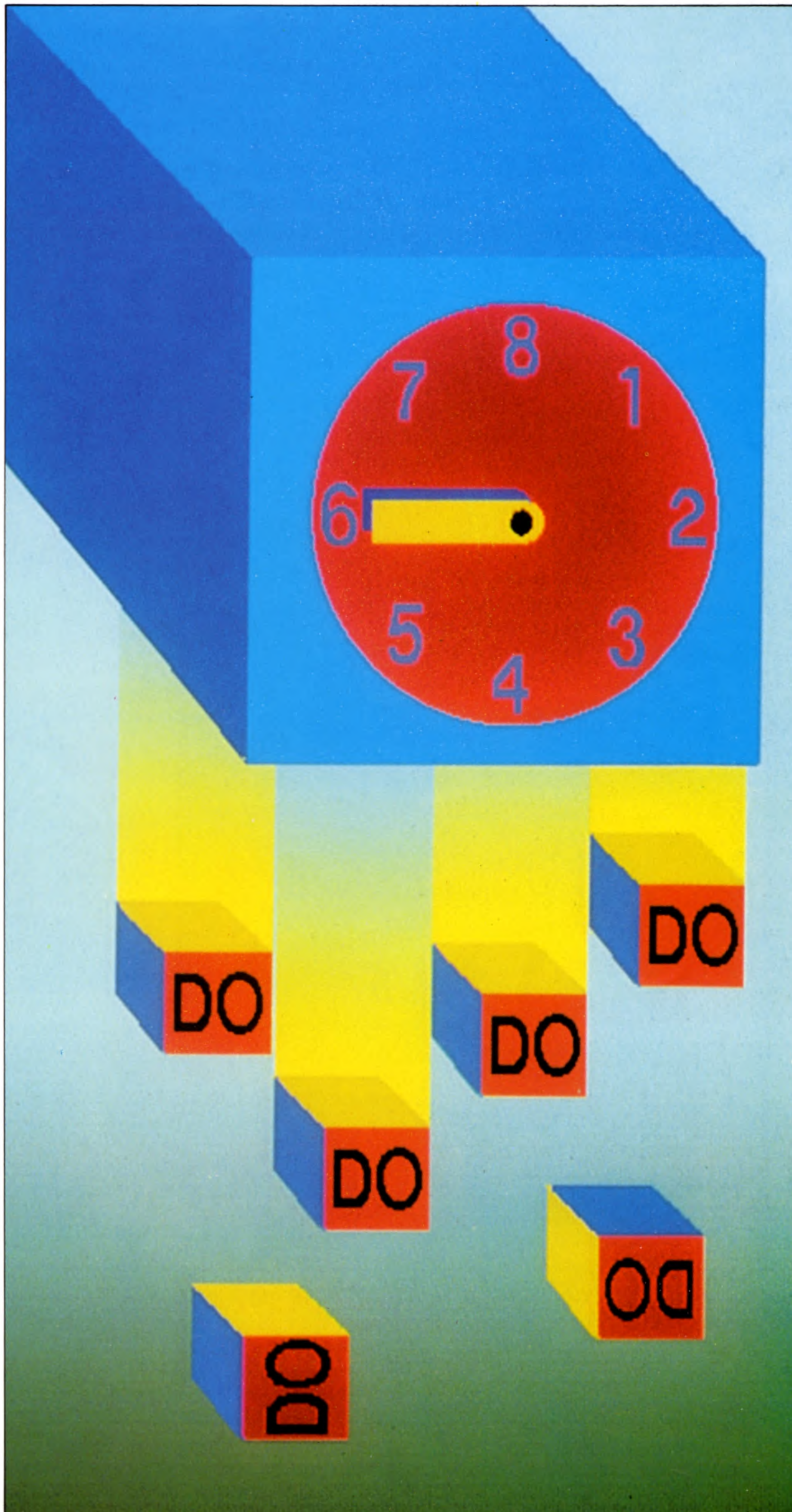
DIALECTS

There are several important FORTH subsets, or dialects, the three most important being standard fig-FORTH, polyFORTH and FORTH-79. A new standard FORTH-83 has been added to update the latter. There are also many machine-specific implementations based on these standards.

Because FORTH can be extended to suit the user, the question of standards is much less important than for other languages. It is easy to adapt a program to another, simply by using colon definitions to introduce the missing word routines to the CURRENT work dictionary.

Here, for instance, is a selection of colon definitions for transforming a fairly typical fig-FORTH implementation into one that will run FORTH-79 programs, in effect by merging the two:

```
: FORTH-79 ;
: VARIABLE      HERE VARIABLE ;
: 2VARIABLE     VARIABLE 2 ALLOT ;
: CONVERT       (NUMBER) ;
: >IN           IN ;
: ?DUP          —DUP ;
: CREATE        VARIABLE —2 ALLOT;
: SAVE-BUFFERS  FLUSH ;
: NEGATE        MINUS ;
: DNEGATE       DMINUS ;
: Ø>            MINUS <Ø ;
: FIND          —FIND DUP IF 2DROP
                CFA THEN;
: EXIT          R> DROP ;
: DEPTH         SP@ SØ @ SWAP — 2 / ;
: WORD          WORD HERE ;
: MOVE          2* CMOVE ;
: U/MOD         U/ ;
: D<            ROT 2DUP = IF ROT
                ROT DMINUS
                D+ Ø< ELSE SWAP <
                SWAP DROP
                THEN SWAP DROP ;
```



Even in something as mundane as a sequence of standard conversions you can see the real power of a definition and of FORTH as a whole. Take the word D< listed above if you're in any doubt—a whole sequence of conditional tests and operations summed up by just two characters!

CHOOSING A SYSTEM

If the choice of dialect does not matter very much, there are still other considerations you ought to keep in mind to ensure that the version you buy is usable. For obvious reasons, FORTH is rather better suited to use in conjunction with disks than with cassettes as a certain amount of toing and froing between dictionaries and screen inputs is necessary. It is possible to save and reload work screens from cassette but this can be tedious, detracting greatly from the fluidity and general freedom in programming that forms much of FORTH's appeal.

Another point to note is that because FORTH gives you complete access to—and thus control of—the computer down to machine level (unlike most other high level languages), you can actually overwrite parts of the FORTH system which cannot be protected. System crashes are therefore quite frequent when you first start experimenting—and experiment you must to get a real 'feel' for the power of FORTH. The simple solution is to save your current work frequently!

CLIFFHANGER: SETTING IT OFF

The 'CLIFFHANGER' listings published in this magazine and subsequent parts bear absolutely no resemblance to, and are in no way associated with, the computer game called 'CLIFF HANGER' released for the Commodore 64 and published by New Generation Software Limited.

The scene is set, Willie is in position. The snakes and the sea are waiting in the wings. The boulders are piled at the top of the cliff. This program now cries: 'ACTION!'

And now the moment of truth has come. So far you should have keyed in and tested all the separate routines that make up Cliffhanger. Now, this final routine calls them all in the correct order and runs the whole game.

When you've keyed this routine in and run it, the game should work. And all the effort you've put into typing each part of Cliffhanger will be worthwhile!

If, however, it does not work properly, there will be a special article on debugging Cliffhanger in the next part of *INPUT*.

The following program is the main loop which completes the game:

```
        org 58702
alp     call 59153
        call 58993
        call 59823
        call 58882
        call 58795
        call 58751
        ld a, (57336)
        cp 1
        jp z,59788
        cp 2
        jp z,59652
dll     ld b,5Ø
delb    ld a,255
dela    dec a
        jr nz,253
        djnz 251
        ld a,254
        in a,254
        bit Ø,a
        jr nz,alp
        ret
```

When you have this routine, and all the others, in memory, start the game by keying in the instruction:

LET L = USR 58576

As you will see, the call address is that of the label **gbin** at the beginning of the initialization routine on page 1101.

WILLIE, YOUR CALL

The routine calls, in turn, the man-moving routine at 59,153, the boulder routine at 58,993, the snake routine at 58,882, the sea routine at 58,882, the cloud routine at 58,795 and the gull routine.

REWARD: DEAD OR ALIVE

Next the routine looks at the state of the so-called die variable. It is loaded up from 57,336 and compared to 1.

If it is 1, the **jr z** instruction sends the processor off to the reward routine at 59,788. This increments the score and sets up the next screen.

The contents of the accumulator are then compared to 2. If they are 2, the processor is sent off to the die routine at 59,652. This is the one that lays Willie in his grave and finishes the game off.

SLOW MOTION

If the processor was just allowed to run round and round this routine, even though it calls all the other routines in memory, the game would be unplayably fast. So, to slow down the motion, two nested loops are constructed which delay the processor by around two-thousandths of a second. That may not sound long, but it mounts up because this routine is called so often.

B is loaded with 50 and A is loaded with 255. Then the contents of A are decremented and the **jr nz** instruction following it sends the processor round the **dela** loop 256 times until the contents of A have been decremented to zero.

The **djnz** instruction then decrements the contents of the B register and sends the processor back to load A with 255 again until B has been decremented to zero. So this outer loop is executed 50 times and the inner loop is executed 255 × 50 times.

But the B register only contains 50 when the game starts out. When Willie reaches a reward, a new number is poked into the **ldb, 5Ø**. In fact, if you look back to page 1475, you'll see that the number in this location is loaded up, decremented and loaded back each time Willie reaches a reward. So the processor goes round this delay loop one less time—speeding the game by something like 90 microseconds!

BREAK OUT

Finally, all good programs should have some way of breaking out of them without having to switch off the computer and losing everything in memory. Here you check to see whether the BREAK key has been pressed. This is done using the **in** command in exactly the same way as it was on page 731.

Then the **jr nz,alp** instruction loops the processor round to the beginning of this main loop again if the BREAK key has not been pressed. If it has, the processor goes on to the **ret** and returns to BASIC.

Before you key in the main loop you need a little loop that will clear all the flags:

```
ORG  256ØØ
LDA  #$ØØ
STA  $DØ15
STA  $CØØ5
STA  $CØØ6
STA  $CØØC
STA  $DØ1E
STA  $DØ1F
LDA  $DØ18
AND  #$FØ
ORA  #$ØC
STA  $DØ18
RTS
```

So Ø is loaded into A and stored in $DØ15, the sprite enable byte—in other words, all the existing sprites are turned off. Then the same Ø is stored in the $CØØ5—the jump-right flag; $CØØ6—the jump up flag; and $CØØC—the sea counter. This makes sure that Willie does not start off by jumping and that the sea is started at the bottom of the screen.

The contents of memory locations $DØ1E and $DØ1F are loaded into the X register. Nothing is going to be done with them there, it is simply that these two locations are the sprite collision detection registers and the act of reading them by loading their contents up into a register automatically clears them.

Next the contents of $DØ18 are loaded up. This is the VIC control register. The most significant nybble contains the screen base address. You don't need to move the screen so this is ANDed with $FØ.

The least significant nybble contains the base address of the character set. This does need to be changed as you are using a redefined character set. So it is ORed with $ØC, which effectively POKEs $ØC into the lower nybble and shifts the pointer to the character set beginning at 53,ØØØ.

The result of these operations is stored back in $DØ18 and the processor returns.

MOVING OBJECTS

Next you need a little routine that deals with moving the various sprites around:

```
ORG 26112
JSR $59ØØ
JSR $58ØØ
JSR $57ØØ
JSR $565Ø
JSR $56ØØ
RTS
```

This jumps to the subroutine at $5900 which moves the boulder; then the subroutine at $5800 which moves the cloud; then the subroutine at $5700 which moves the sea; then the subroutine at $5650 which moves the gulls; then the subroutine at $5600 which flicks the snakes' tongues in and out.

When all that is done, the processor returns.

STARTING OVER

Then you need a short routine that initializes everything and puts them in their proper place:

```
ORG  26368
JSR  $6400
JSR  $6300
JSR  $5850
JSR  $6150
JSR  $6100
JSR  $6000
RTS
```

This jumps to the subroutine at $6400 which clears the flags; then the subroutine at $6300 which prints up the score; then the subroutine at $5850 which puts the boulder at the top of the slope; then the subroutine at $6510 which puts the cloud in its starting position; then the subroutine at $6100 which starts the snakes off; then the subroutine at $6000 which puts the sprites up on the screen.

With all that done, the processor returns.

MAIN LINE

And finally, you come out of the subroutine sidings and onto the main loop routine which runs the whole game. Its start address is the one you call to run the game.

```
          ORG 26448
          JSR $4000
          JSR $6500
          JSR $6300
          JSR $67A9
          LDA #$00
          STA $D015
          LDA $D01E
          LDA $D01F
          LDA #$38
          STA $D017
          JSR $6700
          JSR GETSCORE
          LDA $D01E
          LDA $D01F
LOOP      JSR $6600
          JSR $6200
          LDX $C002
LOOPB     LDY #$FF
LOOPA     DEY
          BNE LOOPA
          DEX
          BNE LOOPB
          JSR $6450
          BEQ LOOP
          JSR PUTSCORE
          LDA #$00
          STA $D015
          LDA $D01E
          LDA $D01F
          LDA $C001
          BNE $675C
          LDA #$15
          STA $D018
          JMP $6750
PUTSCORE  LDX #$00
          LDY #$06
          LDA $047E,X
          STA $C382,X
          INX
          DEY
          BNE $67AD
          RTS
```

```
GETSCORE LDX #$00
         LDY #$06
         LDA $C382,X
         STA $047E,X
         INX
         DEY
         BNE $67BC
         RTS
```

This jumps to the subroutine at $4000 which prints up the title page; then the subroutine at $6500 which gives initial values to the lives, levels and score, fixes the gulls' initial position and sets up the sprites; then the subroutine at $6300 which prints up the scenery; then the PUTSCORE subroutine which puts the score into the variable area.

A is loaded with 0 which is stored in $D015 which switches the sprites off. Then the two sprite collision registers at $D01E and $D01F are read, which clears them.

The number 56 is loaded into the accumulator and stored in $D017. This location gives double height sprites to those with the appropriate bit set. 56 sets bits three, four and five—the bits corresponding to the snakes' sprites.

The processor then jumps to the subroutine at $6700 which initalizes the level. Then it goes to the GETSCORE subroutine which gets the score from memory.

The contents of $D01E and $D01F are loaded up into A to clear them again. Then the processor goes into the main loop.

LOOPING THE LOOP

Once the game has been initialized, the processor heads off into the action. It jumps to the subroutine at $6600 which moves the objects around; then it jumps to the subroutine at $6200 which moves Willie.

Already things are moving too fast, so the action has to be slowed down a bit at this point. The processor moves so fast that the game would be unplayable if it was allowed to continue at that rate. So a delay routine is added at this point.

X is loaded with the contents of memory location 49,154. This is the delay variable which tells the game how fast to go—the game is speeded up during play by decrementing this delay.

Y is loaded with 255. This is used as a counter for the inner loop of the delay. Y is decremented. If it has not been decremented to zero the processor loops back to be decremented again.

When the processor drops out of that loop, the contents of the X register are decremented. The processor branches back if that register has not been counted down to zero and loads Y up with 255 again.

When the processor has finally dropped out of the outer loop, it jumps to the sub routine at $6450 which checks to see whether Willie is dead, or whether he has reached a reward.

If he's alive and unrewarded, the accumulator contains 0 when the processor leaves this subroutine and the BEQ instruction takes the processor back round to LOOP.

DEAD BUT NOT FORGOTTEN

If Willie is dead or has been rewarded, the processor continues and jumps to the PUTSCORE subroutine which prints the score up on the screen.

A is then loaded with 0 which is stored in $D015. This turns the sprites off again. And the collision registers are cleared by loading the contents of $D01E and $D01F into the accumulator.

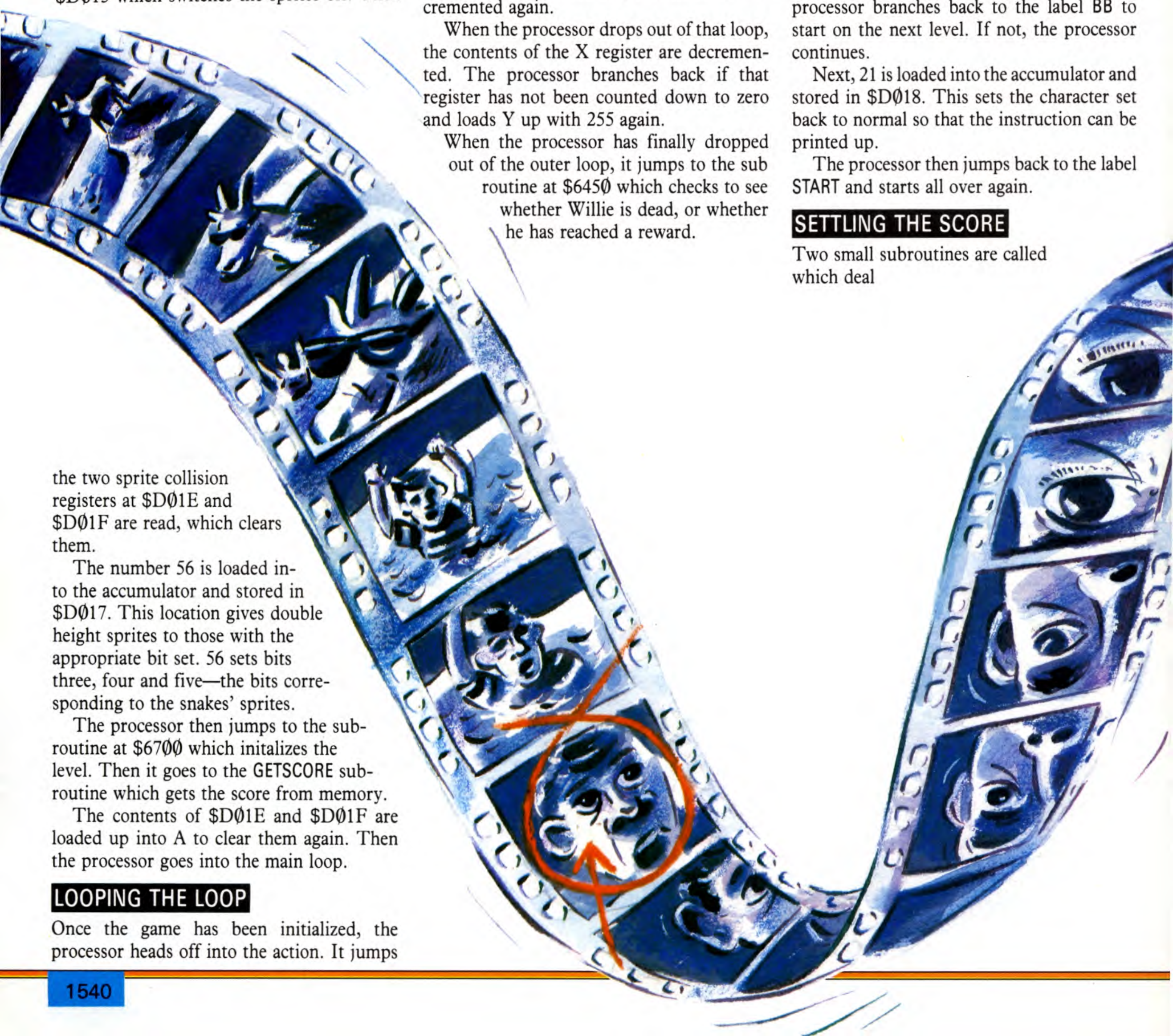
The number of lives—which has now been decremented—is loaded up from $C001 into the accumulator. And if they are not zero, the processor branches back to the label BB to start on the next level. If not, the processor continues.

Next, 21 is loaded into the accumulator and stored in $D018. This sets the character set back to normal so that the instruction can be printed up.

The processor then jumps back to the label START and starts all over again.

SETTLING THE SCORE

Two small subroutines are called which deal



with the score. The first, PUTSCORE, copies the score from the screen and stores it in some free memory. It does this when Willie dies, to save the score when the scene is changed.

The second, GETSCORE, copies the score from free memory back onto the screen when the new screen is printed up.

In PUTSCORE X is loaded with Ø. This is going to be used as an offset to move across the digits. And Y is loaded with six. It is going to be used as a counter—there are six digits in the score.

The accumulator is loaded with the contents of $Ø47E offset by X. This is one of the digits. It is then stored in $C382, offset by X, part of free memory. X is then incremented to move the screen pointer onto the next byte, and Y is decremented. If it has not counted down to zero the BNE instruction branches the processor back to deal with the next digit. If not, the processor proceeds, hits the RTS and returns.

GETSCORE works in exactly the same way, except that LDA $Ø47E,X and STA $C382,X instructions are replaced by LDA $C382,X and STA $Ø47E,X. This simply reverses the direction of the transfer and copies the score for memory onto the screen.

Now load all the parts of Cliffhanger into memory and key in:

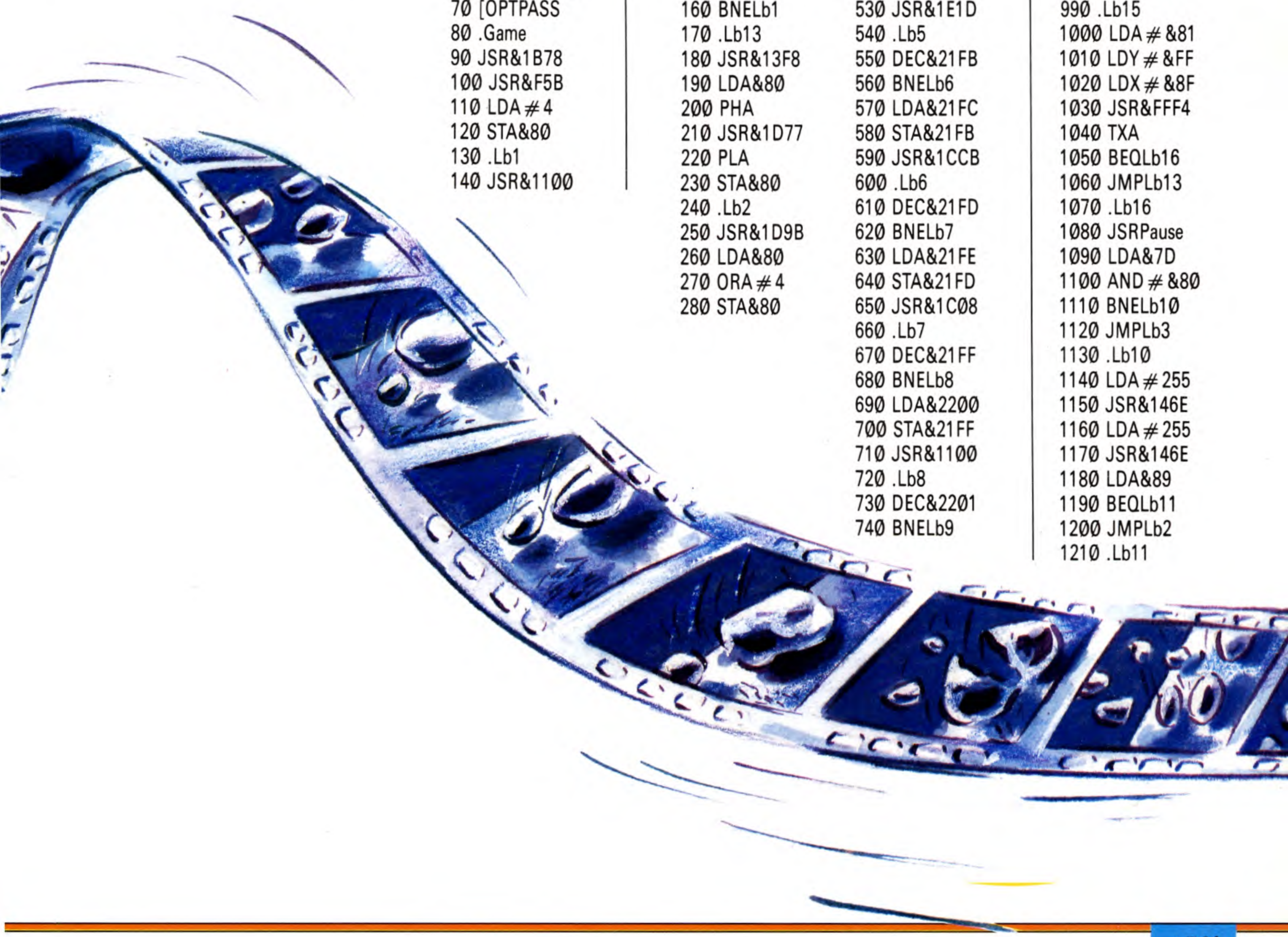
```
SYS 26448
```

to start off the game.

The following program ties all the separate routines together and turns them into a games program. After the game has been set up, each routine is called in turn. But between each call there is a delay.

Don't forget to set up the computer in the usual way before you key in this program:

```
30 DATA1,3,1,2,1,1,1,5,1,5,1,1,1,1
40 FORA%=&21F7TO&2204:READ?A%:NEXT
50 FORPASS=ØTO3STEP3
60 P%=&2205
70 [OPTPASS
80 .Game
90 JSR&1B78
100 JSR&F5B
110 LDA#4
120 STA&80
130 .Lb1
140 JSR&1100
150 LDA&80
160 BNELb1
170 .Lb13
180 JSR&13F8
190 LDA&80
200 PHA
210 JSR&1D77
220 PLA
230 STA&80
240 .Lb2
250 JSR&1D9B
260 LDA&80
270 ORA#4
280 STA&80
290 LDA#15
300 LDX#Ø
310 JSR&FFF4
320 LDA#Ø
330 STA&7Ø
340 LDA#2
350 LDX#&7Ø
360 LDY#&Ø
370 JSR&FFF1
380 JSR&1E99
390 LDX&83
400 LDA&1B2D,X
410 AND#&4
420 BEQLb3
430 JSR&1DEE
440 .Lb3
450 DEC&21F9
460 BNELb5
470 LDA&21FA
480 STA&21F9
490 LDX&83
500 LDA&1B2D,X
510 AND#&4
520 BEQLb5
530 JSR&1E1D
540 .Lb5
550 DEC&21FB
560 BNELb6
570 LDA&21FC
580 STA&21FB
590 JSR&1CCB
600 .Lb6
610 DEC&21FD
620 BNELb7
630 LDA&21FE
640 STA&21FD
650 JSR&1C08
660 .Lb7
670 DEC&21FF
680 BNELb8
690 LDA&2200
700 STA&21FF
710 JSR&1100
720 .Lb8
730 DEC&2201
740 BNELb9
750 LDA&2202
760 STA&2201
770 LDX&83
780 LDA&1B2D,X
790 AND#&2
800 BEQLb9
810 JSR&21A6
820 .Lb9
830 DEC&21F7
840 BNELb4
850 LDA&21F8
860 STA&21F7
870 JSR&1FD5
880 JSR&2141
890 JSR&21Ø7
900 .Lb4
910 DEC&22Ø3
920 BNELb15
930 LDA&22Ø4
940 STA&22Ø3
950 LDA&7C
960 AND#&4
970 BNELb15
980 JSR&1EB6
990 .Lb15
1000 LDA#&81
1010 LDY#&FF
1020 LDX#&8F
1030 JSR&FFF4
1040 TXA
1050 BEQLb16
1060 JMPLb13
1070 .Lb16
1080 JSRPause
1090 LDA&7D
1100 AND#&8Ø
1110 BNELb1Ø
1120 JMPLb3
1130 .Lb1Ø
1140 LDA#255
1150 JSR&146E
1160 LDA#255
1170 JSR&146E
1180 LDA&89
1190 BEQLb11
1200 JMPLb2
1210 .Lb11
```



```
1220 LDA#&81
1230 LDY#&FF
1240 LDX#&9D
1250 JSR&FFF4
1260 TXA
1270 BEQLb11
1280 JMPLb13
1290 .Pause
1300 .Lb14
1310 LDA#1
1320 LDX#&7Ø
1330 LDY#&Ø
1340 JSR&FFF1
1350 LDA&7Ø
1360 CMP&84
1370 BCCLb14
1380 SEC
1390 SBC&84
1400 STA&7Ø
1410 LDA#2
1420 LDX#&7Ø
1430 LDY#&Ø
1440 JSR&FFF1
1450 RTS
1460 ]NEXT
1470 ?&1D92=255
1480 ?&1D8F=8
1490 ?&21AC=1
1500 ?&15D9=4:?&15DA=12:?&1B
    6A=3:?&10F1=2:?&10F3=2:?&10F7=
    10:?&DF6=85
1510 DATA 4,1,Ø,Ø,Ø,Ø,Ø,Ø,2,253,−127,
    −127,40,Ø
1520 FOR A%=&149F TO &14AC:READ
    ?A%:NEXT
1530 ?&14C5=1Ø
1540 ?&14CE=&9F
```

For the Electron make these changes:

```
1550 ?&1100=96: ?&1435=2Ø:
    ?&2214=&EA: ?&2215=&EA:
    ?&23Ø9=96
```

You'll need a short BASIC program to run the game for you.

NEW first, then:

```
1Ø*KEY1Ø ?&DØØ=22:?&EØØ=12Ø:
    ?&EØ1=86!MCALL&21C7!M
2Ø *TV255
3Ø PRINTTAB(15,12)"PLEASE WAIT"
4Ø VDU28,Ø,24,39,1Ø
5Ø *RUN"CliffMC"
```

SAVE this routine to tape under the file name 'Cliff', then load up all the machine code and SAVE it using the instruction:

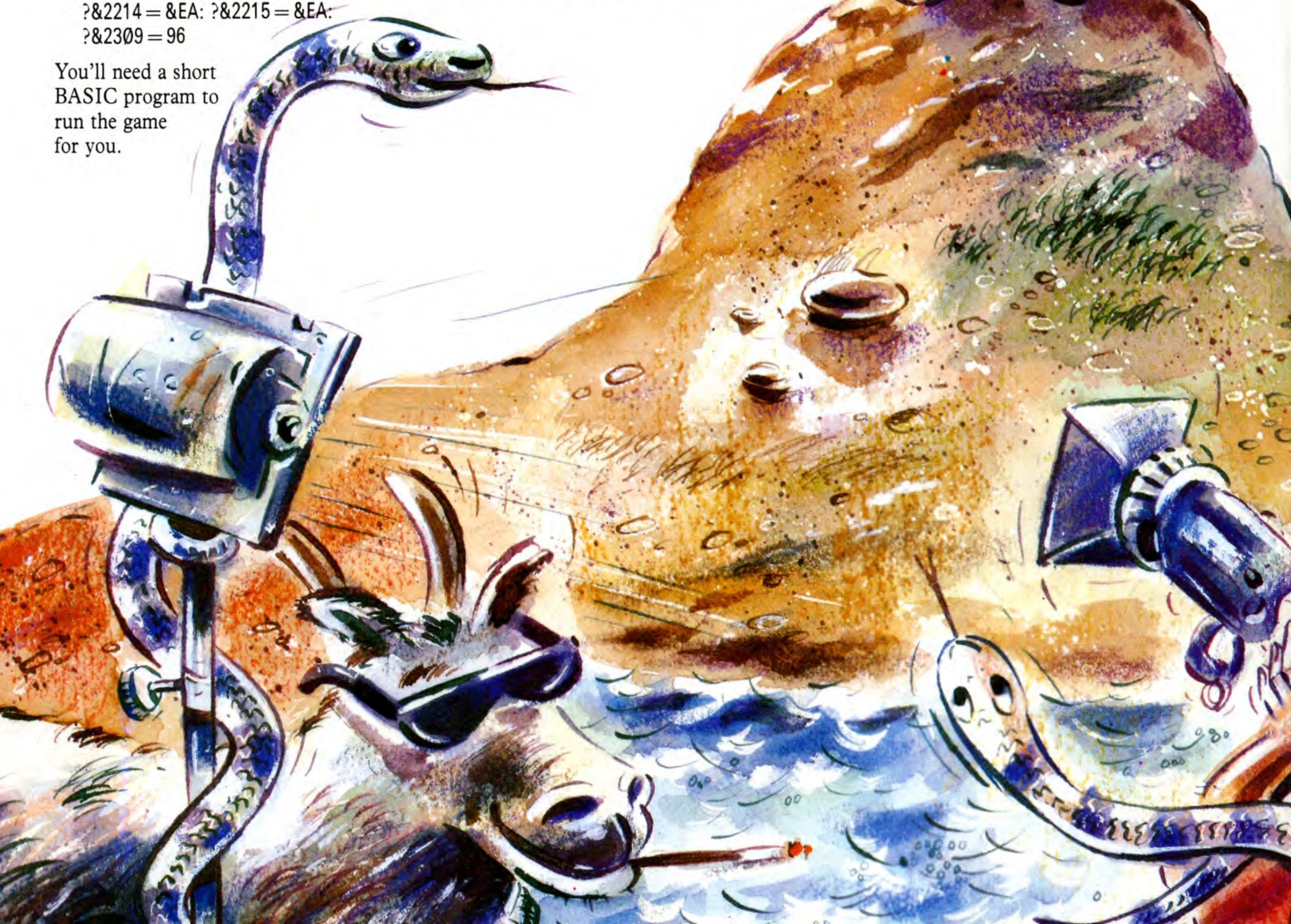
```
*SAVE"CliffMC" ØDØØ□24ØØ□22Ø5
```

READY...

Lines 3Ø and 4Ø READ in more DATA into a data table at &21F7 to &22Ø4. These locations carry the initial and reset values of all the delays.

The instruction in Line 9Ø jumps the processor to the subroutine at &1B78 which sets the 'loud' envelope. Next it jumps to the subroutine at &F5B which prints the title.

The number 4 is then loaded into the accumulator and stored in &8Ø, the location that controls how long the tune is played. The instruction in Line 14Ø then jumps to the



subroutine at &11ØØ and plays the tune.

The tune routine automatically decrements the contents of &8Ø. So after the processor has returned the contents of &8Ø are loaded into the accumulator and the BNE instruction in Line 16Ø branches the processor back to play more of the tune if the contents of this location have not counted down to zero.

STEADY...

The processor then jumps to the subroutine at &13F8 which prints up the instruction page. The contents of &8Ø are then loaded up into the accumulator and pushed onto the stack, to save it temporarily while &8Ø is used for other purposes.

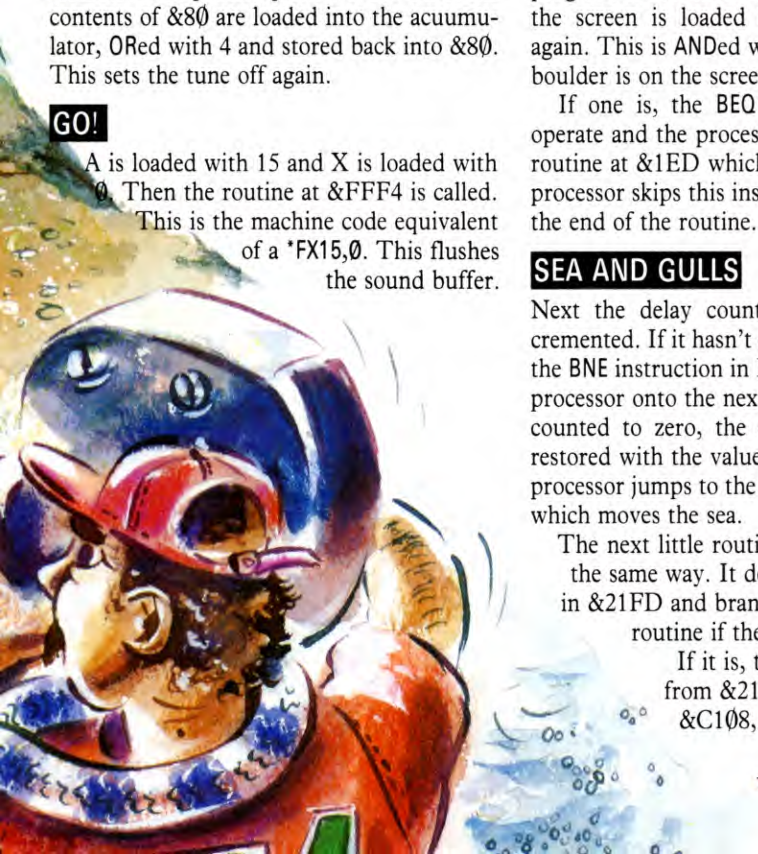
The processor jumps to the subroutine at &1D77, which initializes all the variables, and the contents of the accumulator are pulled back off the stack and stored back in &8Ø.

The pushing and pulling of the contents of &8Ø may seem unnecessary as they were counted down to zero by the routine above. But this is not the only time the routine is called. When the game is over, the processor jumps back to the instruction page again with a different value in &8Ø.

The next jump is to the subroutine at &1D9B. This prints up the screen. Then the contents of &8Ø are loaded into the acuumulator, ORed with 4 and stored back into &8Ø. This sets the tune off again.

GO!

A is loaded with 15 and X is loaded with Ø. Then the routine at &FFF4 is called. This is the machine code equivalent of a *FX15,Ø. This flushes the sound buffer.



Next Ø is stored in &7Ø and the X and Y registers. And 2 is loaded into A. So when &FFF1 is called in Line 37Ø the least significant byte of the time is set to zero. This zeros the clock so that it can be used to time a delay later.

The processor jumps to the subroutine at &1E99 to print up Willie. Then the contents of &83, the location that carries the level number, is loaded into the X register. It is then used as an offset in the LDA&1B2D,X instruction in Line 4ØØ which loads the accumulator with a number whose bits indicate what is required in that screen.

Firstly, the contents of the accumulator are ANDed with 4 to see if a boulder is required, the BEQ instruction will not operate and the processor will jump to the subroutine at &1DEE and print a boulder on the screen. If not, the BEQ instruction branches the processor over the subroutine call.

UP AND RUNNING

Now that everything is set up, the processor moves into the main routine.

The delay counter in &21F9 is decremented and the BNE instruction in Line 46Ø branches the processor forward to the next routine if it has not counted down to zero.

But if it has counted down to zero, the processor continues and restores the counter in &21F9 with the value from &21FA. Then X is loaded with the contents of &83, the level number, again and the byte which tells the program which items need to be printed on the screen is loaded into the accumulator again. This is ANDed with 4 again, to see if a boulder is on the screen.

If one is, the BEQ instruction does not operate and the processor jumps to the subroutine at &1ED which moves it. If not, the processor skips this instruction and moves to the end of the routine.

SEA AND GULLS

Next the delay counter in &21FB is decremented. If it hasn't counted down to zero, the BNE instruction in Line 56Ø branches the processor onto the next routine. But if it has counted to zero, the counter in &21FB is restored with the value of &21FC. Then the processor jumps to the subroutine at &1CCB which moves the sea.

The next little routine operates in exactly the same way. It decrements the counter in &21FD and branches to the end of the routine if the result is not zero.

If it is, the counter is restored from &21FE and the routine at &C1Ø8, which moves the seagulls, is called.

MUSIC SHAKES THE SNAKES

The music delay counter in &21FF is decremented. If the result is zero the processor branches on to the next routine.

If it is, the counter is restored from &22ØØ and the processor jumps to the routine that plays the tune at &11ØØ.

The instruction in Line 73Ø then decrements the snake counter in &22Ø7. And if it has not been decremented to zero the BNE instruction following branches.

If it has counted down to zero, the processor continues and the counter is restored from &22Ø2. Then the level number is loaded into X and the screen extras are loaded from &1B2D by X again.

This is ANDed with 2 to see if there are any snakes on this screen. If there are the processor jumps to the subroutine at &21A6 and moves them. If not the BEQ in Line 8ØØ jumps over that instruction.

THEN THERE WAS WILLIE

The main routine counter in &21F7 is then decremented, checked to see if it is zero and restored from &21FF if it is.

The routine at &1FD5 is called which moves Willie, then the one at &2141 to sort out the score, then the one at &21Ø7 to check if Willie has reached the end of a screen.

Next the delay counter for Willie's death routine in &22Ø3 is decremented. If it has reached zero it is restored from &22Ø4. Then the data on Willie's physical condition is ANDed with 4. This checks to see whether he is dead and calls the routine at &1EB6 to finish him off.

ESCAPE FROM CLIFFHANGER

Pressing the ESCAPE key while you are playing Cliffhanger allows you to escape from the game and start again. A is loaded with &81. The OSBYTE routine at &FFF4 is going to be called. This gives the same effect as an OSBYTE &81 call and reads the keyboard in the same way as explained on pages 1382 and 1383.

The &FF in Y and the &8F in X specify the key to be scanned—in this case the ESCAPE key. If it has been pressed a &FF is returned in X, if not, a Ø is returned.

The result of the scan is transferred into the accumulator and the BEQ in Line 1Ø5Ø branches over the next instruction if the ESCAPE has not been pressed. But if it has been pressed the processor hits the JMP instruction and jumps back to the beginning of the game to start all over again.

If the ESCAPE key has not been pressed, the processor jumps to the subroutine Pause—

which begins in Line 129Ø—to give a pause before going on with the game.

Then the accumulator is loaded with the contents of &7D which stores Willie's condition. ANDing with the number &8Ø checks to see whether a reward has been reached. If it hasn't, the processor jumps back to Lb13 and starts the main loop again. If it has, the processor skips this instruction, loads A with 255 and jumps to the delay routine at &146E twice. This is the delay routine which was used to give you enough time to read the instructions.

Next the contents of &89 are loaded into the accumulator. This is the location that carries the number of lives Willie has left. If he has lives left the processor jumps to Lb2 and continues with the game. If not the processor skips the jump instruction.

Next the OSBYTE &81 routine is used again to check whether the space bar has been pressed. The BEQ instruction in Line 127Ø loops the processor back, so it goes round and round this check until the space bar is pressed. When it is, the processor jumps back to Lb13 and starts the game again.

EASING THE PACE

The rest of the program is a delay to slow the game down enough to make it playable. This is done by reading—and resetting—the clock. to do this an OSWORD call is made by jumping to the subroutine at &FFF1.

When this is done with a 1 in the accumulator—as in Line 131Ø—the clock is read and its five-byte value is written into memory starting at the address given in the X and Y register. Here Y contains Ø and X contains &7Ø, so the time is recorded in zero-page memory locations &7Ø to &74.

The low byte of the time in &7Ø—the hundredths of a second—are then compared with the delay value in &84 by the instructions in Lines 135Ø and 136Ø. The BCC in Line 137Ø jumps back to the beginning of the pause routine to read the clock again. It goes on jumping back and reading the clock again until the number of hundredths of a second in &7Ø is greater than the delay number in &84. When this happens the compare—which is an unrecorded subtraction, remember—sets the carry flag so the Branch on Carry Clear does not operate and the processor continues.

The clock then needs to be reset. But you don't want it to start again from zero. Instead, it is going to be reset to the value given by the number of hundredths of a second in &7Ø and the delay—in other words, the amount the clock has been incremented past the delay.

So the carry flag is set by the SEC in Line 138Ø. The accumulator is still carrying the contents of &7Ø picked up by the instruction in Line 135Ø, so the SBC&84 in Line 139Ø performs the subtraction and the result is stored back in &7Ø by Line 14ØØ.

The accumulator is then loaded with 2 which allows you to write to the clock when the JSR&FFF1 in Line 144Ø jumps to the OSWORD routine. The data that is used to reset the clock is in the five bytes starting at the address given by the contents of X and Y. Again this is zero-page locations &7Ø to &74.

The following program is the main action loop which completes the game. Tandy owners should change the JSR 32774 to JSR 414Ø9.

```
       ORG 2Ø932
ALP    LDA #5
       STA 18258
       CLR 18261
       JSR ELB
BLP    JSR MAN
       JSR BAR
       JSR SNK
       JSR SEA
       JSR MOVSUN
       LDA 18252
       CMPA #1
       LBEQ RWD
       CMPA #2
       LBEQ DIE
DLL    LDB #1ØØ
DEL    CLRA
DELA   DECA
       BNE DELA
       DECB
       BNE DEL
       JSR 32774
       CMPA #3
       BNE BLP
       RTS
MOVSUN EQU $4DØF
ELB    EQU $4B59
MAN    EQU $4DBE
SNK    EQU $5178
SEA    EQU $4CDE
RWD    EQU $5ØF1
DIE    EQU $5Ø5Ø
BAR    EQU $4D45
```

Now LOAD in the rest of Cliffhanger and assemble this short program on top of it.

```
       ORG 19572
       JMP ALP
ALP    EQU $51C4
```

Key in the instruction:

EXEC 19426

and Cliffhanger should now work!

READYING THE ROUTINE

A is loaded with 5 which is then stored in 18,258. This sets the sun delay.

Then memory location 18,261 is cleared. This is the man-jump variable and it has to be cleared to prevent Willie jumping at the beginning of a screen.

The processor jumps to the ELB routine to put up the extra bits and pieces needed for the higher levels of the game on the screen. Next it jumps to the MAN subroutine to deal with moving Willie. Then it jumps to BAR to move the boulder, then SNK to move the snake, then SEA for the sea, then MOVSUN for the sun.

The contents of the die variable at 18,252 are loaded into the accumulator. This is compared to 1 which means that Willie has reached a reward. And if 1 is found the LBEQ makes the processor take a long branch back to the routine that gives Willie his reward.

If the die variable is not 1, the CMPA #2 checks whether it is 2, the number signifying that Willie is really dead. And if it is 2, the processor makes a long branch off to the DIE routine which buries him.

ROUTINE DELAY

B is loaded with 1ØØ and is used as the counter in the major delay loop. You will note that the memory location occupied by the 1ØØ when the program is assembled—$51EE—is the byte decremented in the score routine.

A is cleared and decremented, making its contents 255. The BNE instruction then checks to see whether it is zero and, if it is not, goes back to decrement it again.

When A has been decremented down to zero, the processor drops out of this loop. Then it decrements B and branches back to clear A and decrement it again if the contents of B have not counted down to zero.

In other words, at first the processor goes round the inner loop 256 × 100 times to slow the game down. But when Willie has had some success in reaching the rewards the game is speeded up by sending the processor round this loop only 256 × 99 times, or 256 × 98 times, or 256 × 97 times and so on to slow it down slightly less.

BREAK DUNCE

Jumping to the subroutine at 32774 checks the keyboard. And comparing the value returned in the accumulator with 3 checks to see if the BREAK key has been pressed.

If it hasn't been pressed, the BNE insruction jumps back to continue the game. But if it has been pressed, the processor continues, hits the RTS and returns to BASIC.

CLOSING THE CIRCLE

One last little refinement has to be added to the program. In the routine you originally entered that scrolls on the appropriate screen and sets the score, there were an RTS and two NOP instructions.

These were used to leave enough space to put in a jump instruction which would loop when the game was first called.

It wasn't filled in then because at that time the action routine hadn't been written. Now it has. So a JMP ALP is assembled in that address. This closes the circle, completing Cliffhanger.

ESCAPE: THE CODED TEXT

Now you have the completed BASIC control program, you can start entering the text. How to escape, though, remains a mystery, because it's all in code and compressed

You're over half-way there. The text program which builds up over the remaining parts of Escape will generate the text file needed to print out the game's messages. Do not RUN the program until you have the complete listing, or it will not work.

```
5 PRINT AT 10,5; "POKING DATA. PLEASE WAIT"
10 RESTORE 6000: DIM Z(1750): DIM A(204): LET
   DI=PEEK 23627+256*PEEK 23628: LET L=6000:
   LET DI=DI+3: FOR N=1 to 533
20 READ A$: LET TOT=0: FOR D=1 TO 32 STEP 2: LET
   B$=A$(D TO D+1)
30 GO SUB 1000
```

```
40 LET TOT=TOT+B: POKE DI,B: LET DI=DI+1:
   NEXT D
45 LET B$=A$(33 TO 34): GO SUB 1000: IF
   B<>TOT-256*INT (TOT/256) THEN PRINT "C
   HECKSUM ERROR IN LINE ";L: STOP
57 LET L=L+2
60 NEXT N
70 FOR N=1 TO 204: READ A(N): NEXT N
80 PRINT : PRINT "READY. SAVING ARRAYS."
90 SAVE "DATAA" DATA A()
100 SAVE "DATAZ" DATA Z()
999 STOP
1000 LET B=16*(CODE (B$(1))-48-7*(B$(1)>
   "9"))+CODE (B$(2))-48-7*(B$(2)>
   "9"): RETURN
6000 DATA "01D60600003F210038C965A33DE4B2F50E"
6002 DATA "90C8656CAD94B2DF2BE499C64B2D32A6B9"
6004 DATA "51C9656CBCB165267590CC650CB6CA791E"
6006 DATA "699532324CEB2E197CBD64B2B33E3259AB"
6008 DATA "1976C9651CBACB8CA795D95C864B2A65BB"
6010 DATA "E323218B29329919632B650CAB96AC5904"
6012 DATA "190C864658CAB95F2E724DB6432D72D383"
6014 DATA "2B6465362C8C96499EF2194C8CB195F27A"
6016 DATA "E72595B2E18B29652CBE1B6CF3953259D1"
6018 DATA "6ACE3259699519EB2595B2B650CBBC8C4A"
6020 DATA "D36572F2CE325968DB6D91CB4CB19632C6"
6022 DATA "BE499C64B2D19BE55CAD9599D64320CF49"
6024 DATA "7919192CC6432EB2590C59498B2865F1D0"
6026 DATA "90CE3259699519B655CB197C3259262C48"
6028 DATA "8C867AC9656CAD9432EF232AE4335C96DE"
6030 DATA "54CA99572865DE46686038C965A3219546"
6032 DATA "32195C86419E3286465F198C8E4652C8FC"
6034 DATA "CAECC657CAD95B34D8B2323232195CA23C"
6036 DATA "CDF232595F2194CB4C8CB9CAB91976CF9B"
6038 DATA "194B2DF2E58B23219BE4656CAD94590C2D"
6040 DATA "8C862C8C86632BE56CACCAE5165F2D729E"
6042 DATA "DB29E5C36DB2196D9432A653679CA9924E"
6044 DATA "CB567192CB4CA8CEB2325978C96465DBD3"
6046 DATA "23919532865BE69B164643259195B2E15A"
6048 DATA "9E329652CA8CAE5162CA4C59190CDF2B0D"
6050 DATA "E51C8CD0C021979C8CAF90C596F9499E77"
6052 DATA "32865C651CFB5166F94B2A65F198CE32A3"
6054 DATA "5968CF792CBD6432D94B29651C965CE52D"
6056 DATA "7C9316464335CA595329E50CA8CA64B2FB"
6058 DATA "D323218B2321B67BCAB919779194597C24"
6060 DATA "B9CA59699779262CA4CF392CA99632BBAB"
6062 DATA "2596D92CB962C8C8678CA594B2A66860B7"
6064 DATA "2B95325919EABACA59699190CC646465AE"
6066 DATA "1C8CA2CB64B232BB1646436DB66F91956F"
6068 DATA "B2B65164B24C59195728B2992CA596B977"
6070 DATA "A6CB64B2EF2594D98C965AE4651CBCE58A"
6072 DATA "4C965AB20C8E5AE4B2B6566759432A6415"
6074 DATA "6499BE55C8C8CA596ECF79572EF24D90CD"
6076 DATA "CBACA597AC867192CB464323218B232351"
6078 DATA "232195CA2C964656CB864B232BB3C652B6"
6080 DATA "CA595334D8B232AE5165728B1646433197"
6082 DATA "94328E4656C86DB6D8B2A64650CA39690D"
6084 DATA "9532E33FD64B232BB259262C8C8663234D"
6086 DATA "2328E465165728B64B2DB259739A18169D"
6088 DATA "464337C8CAD95B28B21953219E72BE510C"
6090 DATA "C8CA594B232BE499BE55C96436D92CAD29"
6092 DATA "978CC6532D32591970C8654CB5CA596937"
6094 DATA "9533FC66B94B28E43259265724CAE51C31"
6096 DATA "B4C964B2B64651CA39A6C591919190CA5B"
6098 DATA "E5164B232B65C3259195DB6D9E32965207"
6100 DATA "CA99A6C59195728B2B9458B232190CA9BA"
6102 DATA "96B94B2D32A65C668638C965A33DE4B2C3"
6104 DATA "F590CBE5CE4B2BE4646499067BCB4CBB11"
6106 DATA "C8CC6432AE54CB8CC6C8E4652C8CAECF8F"
6108 DATA "792CAD95B3D56F91970CDF2AE4B2191CE6"
6110 DATA "8CA19572A6516DB6CB64650CBC64B232F2"
6112 DATA "ED9A65B3219190CF194B2DF2E734D97CA3"
6114 DATA "B9CA59532D72965A649B37CA594B2F19AA"
6116 DATA "1972CF194B29654CA2CB64650CBC337C45"
6118 DATA "AF947233192CA99739432595DB3C652C4B"
6120 DATA "A595328B2394B2D329E43219190C833669"
6122 DATA "CAD9432AE433C652CA5951B6CBE50CA3C8"
6124 DATA "919A18239699195B2597792CA6C86464A0"
6126 DATA "33CE54C8CA397B6432AE5464324CB6501B"
6128 DATA "CA3919C191CB7CAF95F29E7AA8E5AE4684"
6130 DATA "56CAD92C8CBB67BC8C8CBACA2C86464366"
6132 DATA "3CE56CA596F95F2328E50CA990C8656C2E"
6134 DATA "ACCA64B2D323219C64B2D334D9C64B2D73"
6136 DATA "190CA990CAE432196F9195B2B646499D80"
6138 DATA "E50CA9978C8CB9CFF5906472DB2AE54C62"
6140 DATA "A1959B6DB6D91C8CA19572A658CA194745"
6142 DATA "24C596F95728E465166996D96F919532F5"
6144 DATA "F194590C8C86DB1646464646432195321903"
6146 DATA "E329652CA994595C862C8C86472965A6CE"
6148 DATA "53C864323232DF2BE51C866F9195B2B6A3"
6150 DATA "51CD0C20CDF232A65AB2396B9195B2B31C"
6152 DATA "25926759432F8C96465DB3BCA99195F2DE"
6154 DATA "9E516D9C64B2D1E7AB38C965A321953262"
6156 DATA "195C862C8C867C64B2596323232E734DBB"
6158 DATA "B164654C8CAECF194B29654CA2C964B28E"
6160 DATA "F395F2197CBEB2AE4B28B2394B2D329ED3"
6162 DATA "466365F28651C864323219BE57CA3919B1"
6164 DATA "A18020CB650CA997790CB650CBC65C3267"
6166 DATA "B90C8337C9655C96576432DB2A64B21BC2"
6168 DATA "65B29652CBE328B34CB664324CE325960A"
6170 DATA "99A6C8E5EF2194C8CB192CBBCB9CAB97CC"
6172 DATA "E64651CB8C965A66862397BC86779532EA"
6174 DATA "32BE53CA2CE325968C86516432192CADC2"
6176 DATA "978679CA99572BE5CE4669472F790CAE86"
6178 DATA "51679CA992CB567192CB4CA8CF592CAD73"
6180 DATA "95B2865DE466862397BC862CA197C64666"
6182 DATA "5164194C8C967AAF794B29B36C8CBE32ED"
6184 DATA "3219E72A64B2D5B1646433CE55CA990C85"
6186 DATA "86464321978CBD650C8CB195F28B165238"
6188 DATA "CB7C8CA9978CA2DB38C965A335C8CB9C89"
6190 DATA "A19632AE465DE56CB8CD0C38C965A321C6"
6192 DATA "9532190C9316464339CA595329E432F905"
6194 DATA "7390C96498B23219BE55C8CAF9EAB19593"
6196 DATA "F2B656CA195334D9066F94B2965E323254"
6198 DATA "E591C8CA19572A658CA19472595B232839"
6200 DATA "65E328B34CB664323218B2323232195CC2"
6202 DATA "A2C836DB6DB1654CA196D9EABC652CA536"
6204 DATA "95195C862C8C8679CAD94B29654CD0C09F"
6206 DATA "16464337C8CAD95B28B259262C8CAB94EC"
6208 DATA "594C9652CB56432A6436DB6632597AC8C3"
6210 DATA "C8C8C9337CAB92C866328654CBD64649AF"
6212 DATA "92CA197C336CA19195F2B651CD325953FB"
6214 DATA "2E72864B2BB3C652CB4CBAC96433C652B0"
6216 DATA "CA595328B2596D92CB962CA4C59190CE8D"
6218 DATA "729654CA790C8646432B919472AE5E32BA"
6220 DATA "18B232192CB6C965CB3C652CA595328BB4"
6222 DATA "CFDABE50CA3919D595C8C965262C8C8697"
6224 DATA "75943232F9C8CAF94F25919779EABDE491"
6226 DATA "B2B6566432A643219532D72324CEF2D3D6"
6228 DATA "28654CBC651CD0C02B9265728B3C650C72"
6230 DATA "A997C32B90CF79191919A62C8C865325AD"
6232 DATA "94B2D5B6C8E5AE4656CAD94592C933ACEA"
6234 DATA "9652CA597864323219E72BE50CA99B21CC"
6236 DATA "9459C64B2D32A32F97195F28B21978CB74"
6238 DATA "EF232590C59498B232196C965DE4B29E48"
6240 DATA "5C64B2D191C965767ACA19532AE52CB417"
6242 DATA "C932B91947232A65A65AE465CE4B28B202"
6244 DATA "5926C594B2A653CB4CA99198C86432F9C3"
6246 DATA "6B96D94F2E19D654CA197E64B2BE432133"
6248 DATA "9190DB165BE52678CA594B2A651CC64716"
6250 DATA "2965A653C8643232197CA19472334316DF"
6252 DATA "4655CA2CAE5162C8C86779699432A65E95"
6254 DATA "33D5532594B2D734D91CAD96996B96D97C"
6256 DATA "2C9650CB19532594728B1646431650CBCF"
6258 DATA "AC96432B90C59190CD72AE5E32F195B2DB"
6269 DATA "1B259262C8C86532594B2D595CA2C8337E"
6262 DATA "BCB4CA19532F19FEB3DE436DB6432DB205"
6264 DATA "DB259432A65162CA4CEB2190CA395B2352"
6266 DATA "2BE5B669B20CDB2596F94F2BE4335CB41D"
6268 DATA "CBB67BC96516759195F2E59F197394B223"
6270 DATA "C65F1919262CA4C59190DB6CF395B2964A"
6272 DATA "52CA99A6C591919190C8654C8678CA59FD"
6274 DATA "4B2A65162CA4C59190D97CA194723319EE"
6276 DATA "BE57CA390C86464339CA595329E46686DB"
6278 DATA "2391953286465DE466336CB4C8CBBC8C1C"
6280 DATA "B9C932396B9195B2B651679CAB95B2B3DF"
6282 DATA "16464650CAB95334D9066B94B2A653CA4F"
6284 DATA "195190C8C866D92CA39195B2190C8C86A7"
6286 DATA "D9EB28654CAB94B2D324C8E4652C8CAEFC"
6288 DATA "CEB29653CA795B2328B2D92C8CAEC59199"
6290 DATA "90D91CBDE4656CBD64651CD362C8CAB919"
6292 DATA "4595CA2C59190DB6DB6DB2196D9194F29C"
6294 DATA "3232C650CA99719F6A2CA64B2965AE6313"
6296 DATA "2B9267BC9651678CA594B2A65167394B87"
6298 DATA "2A653C864323219BE57CA3919A1835C8B7"
6300 DATA "CB964323219BE52CB5C8CB96572198C94B"
6302 DATA "656CA59699532F3969959B6DB3192CA301"
6304 DATA "9196996B9194590CA990CBE50C964B23AE"
6306 DATA "2ED90C8C8678CA9957232F8C96465DE553"
6308 DATA "0C931650CBAC96465166D9572B646499CB"
6310 DATA "EF2AE5DE499BE55C96433CE4B2965E1BBB"
6312 DATA "219190CF19532AE465F19A6CE325968C11"
6314 DATA "8654C864324C59190CDF2BE51C865F2E20"
6316 DATA "72192C931646433ACA19197CE4657CA707"
6318 DATA "92C8CBBCF55EF2595B2B669B165262C898"
6320 DATA "C866F95F28E4321953219DE5A650CA992C"
6322 DATA "78C8C933C652CB4CBAC96436DB3C652C30"
6324 DATA "A595328B219190C591919190CAE5164147"
6326 DATA "91CB5C9656CACCF194B2965462CA4C592C"
6328 DATA "190CE729654CA7919A18319432A654CB8C"
6330 DATA "8CAB91976C59190CDF2AE465767BC965BA"
6332 DATA "1679C9652CAD919262C8C8CBBCB4CA19C9"
6334 DATA "532F1945E7AB1646465C33C652C8CFB507"
```

```
6336 DATA "36472595F2D58B2933DE4B2F59191919E1"
6338 DATA "63259779739572FCC8C9338C965A66863A"
6340 DATA "219ØCBE56CA197594B2A65ØC9657CC6764"
6342 DATA "9C965A654CF55B64651C8CAF9ØCEB232EF"
6344 DATA "4DB2392CAF96B9458B2933CE55CAD959AD"
6346 DATA "8B232194C9652CB5CD362C8CAB94595C21"
6348 DATA "A391919ØCF8C96465DE7DA8B3AC8C96565"
6350 DATA "4C9652CB5CD362C8C8C8C86432A6436291"
6352 DATA "CB7CA4CF194B29654CA398CB6465ØCBC8F"
6354 DATA "64B232ED97CA194721 9ØC8C86DB66F955E"
6356 DATA "F28E466860164655CA2CA64B2965AB2382"
6358 DATA "96B9195B2B65164B24CB65ØCBD6465CE68"
6360 DATA "46464669B657218B23219AE55CBC65E317"
6362 DATA "2B6432391943232328B2ØCC65ØC8CA5983"
6364 DATA "6D9572BE499D646499BE55C96432F96B4F"
6366 DATA "975953259572F191919192678CA99432Ø7"
6368 DATA "DB2594734DB62C8CAB94595CA2C5919Ø3E"
6370 DATA "CF9C8CAF95F2E73ED459D6464B2A64B226"
6372 DATA "965AE69B6C591953232BB3C652CA595331"
6374 DATA "28B3BCA4CBE5ØCA39198CDF2BE51C864BD"
6376 DATA "32321B6472965A653C8CDØCØ259195D926"
6378 DATA "2C931646433E7232BE57CB9CFB5165B21F"
6380 DATA "865E32595F28B2B9ØDB66F919572592CBØ"
6382 DATA "8CBB65728B3195325963232329E4B23294"
6384 DATA "ED91C965ØC9657C93259ØC832D94B2DDD8"
6386 DATA "91C8CA594B2B669B219459C64B2D197CD4"
6388 DATA "B9C9657C86472F79ØD9F79196B96D94535"
6390 DATA "8B2933DE46433CE57CAF94F34338C965CA"
6392 DATA "A321953219ØC931646432F94328E432FD7"
6394 DATA "9739ØC96498B2321B3ACA19197CE46571D"
6396 DATA "CA792C8CBBCF55EF2595B2B669B2ØCEBFD"
6398 DATA "2965ØCA8CAE516DB6532594328E464B237"
6400 DATA "32ED92C864191CB6CAB9264B2F5919ØC69"
6402 DATA "5919ØDB6CF39572A64669B16464646434E"
6404 DATA "2B9459Ø6472D72595B2B33C652CA5951A2"
6406 DATA "8B29316464339CA595329E46686Ø3DE4B5"
6408 DATA "B2B65E59Ø678CA597E6465CB3BCB4CA1C5"
6410 DATA "9532F19459ØCA99ØCB65ØCB4CBBC96554C"
6412 DATA "C8CBBCA1919ØCF19532AE465F192C8CBD5"
6414 DATA "B64324C83165ØCBAC964339CAF9432A355"
6416 DATA "6D9C64B2D334D8B23232E196CA594B2F28"
6418 DATA "866996CC865167192CB466D94328E51C33"
6420 DATA "DØCØ38C965A3219532195C864191C8CAEØ"
6422 DATA "59195D9DE5ØCA99532329E69B6DB1646F3"
6424 DATA "4646432B9459Ø6759532865F992CAF9Ø12"
6426 DATA "CEF2B652CB7CAB91976CAE436C5919ØC29"
6428 DATA "D72AE5E32F195B2192C931646433CE5638"
6430 DATA "CA594B2A321919ØDB6C591953259699735"
6432 DATA "79264191CB5C9656CACCDF2AE464652CFC"
6434 DATA "B767192CB466328643647 2595D8B2321D3"
6436 DATA "9AE52CA192679CA19519D64656CA596F34"
6438 DATA "9A62C8CAB94595CA2C5919ØCE72AE5DE69"
64ØØ DATA "4B29E5DE5A65ØCA9978CFB516532594B55"
6442 DATA "2D73195C933DE4B28B6C596F9499E32973"
6444 DATA "652CA9947319BE57CA39ØC864643239444"
6446 DATA "B2D329E4668616464337CAB919194B2D81"
6448 DATA "D95CA2CDF2AE5E32197CB9CA59699779BE"
645Ø DATA "2679C96546D9C64B2D18B2932397CE4655"
6452 DATA "57CBF3218B232A64B2D32EF259A6CE3216"
6454 DATA "5968CF792CBD64653C8C86532596D92717"
6456 DATA "9A1Ø8621 9ØCD3232592CBC65C32992CAØØ"
6458 DATA "596B98C5919ØCDB2B65ØCAB9ØDB6DB3C24"
646Ø DATA "E4B2D32A67AADB2328E4657C86759192AD"
6462 DATA "65728B2D92C9653CAF91919266F95725C9"
6464 DATA "91978C9651CB19432A65E32324CF79692C"
6466 DATA "9192C96465DB3195B29653C8C8CA39A62A"
6468 DATA "C59195728E55CA2C5919ØCE32596991 9Ø4"
647Ø DATA "76CDB2A655C8CB195F3ED4594C9652CB65"
6472 DATA "5CD36DB1646464 6432B9459Ø678CA594C9"
6474 DATA "B2A3165262C8C867394B2A653C86432351"
6476 DATA "232833C652CA59518B2931646432F94325"
6478 DATA "28E46686164643219572A33DE46464327D"
648Ø DATA "B94591C8CA195B219ØC8C86DB6D9EB25E2"
6482 DATA "9195F2EF28654CB8CC67BCB4CBBC866FB7"
6484 DATA "94B2965E3232E598CA19472F8CA2C865CF"
6486 DATA "4C8653286465DE464995C86532596F94D3"
6488 DATA "59E32965BE5CB16464337C8CAD95B28E1A"
649Ø DATA "69B164655CA2CAE5162C8C866F9572328C"
6492 DATA "BE7AAC657CAD95B28654CD362CA4C591BC"
6494 DATA "9197394B2A653C867192CB466328643232"
6496 DATA "392CAEC8323919432AE54CB194328E494B"
6498 DATA "B6432321 9Ø678CA594B2A32596D92CB9C7"
65ØØ DATA "CA2C59498B232197CA19472192C9338C63"
65Ø2 DATA "965A66861646431654CA196D9EABC65296"
65Ø4 DATA "CA595195CA2CB652CB197C6464B343385D"
65Ø6 DATA "C965A321953219ØC9316464337CAF94751"
65Ø8 DATA "2197CB9C864B24C836D96CA591976CDB6B"
651Ø DATA "28651CA39432EF2334D8B2323232190C9D"
6512 DATA "A99ØCF194B29654CA2D9ØCAD94B232EDDF"
6514 DATA "9D652CA79ØC8E55CBC6465164323219Ø2Ø"
6516 DATA "65B28654CBBC8CC6D91C8CA59195D91CØB"
6518 DATA "8CA19572A658CA194724CB6465ØCBC6541"
652Ø DATA "1678CA596F9739A18Ø38C965A3219532Ø2"
6522 DATA "195C862C8C866B9572F1978CAD9ØC96429"
6524 DATA "98B2321B6CB652C8CBB66D94328E51CA3Ø"
6526 DATA "1977919A6C591919190C8654C8678CA511"
6528 DATA "94B2A6516C96433ACA594F2191CAB978DB"
653Ø DATA "C8CA39A18Ø38C965A3219532190C9316AB"
6532 DATA "46432F94328E432F9739ØC96499Ø6DB2E8"
6534 DATA "D94B232ED9B65ØCA39472865DE4669B26A"
6536 DATA "ØC8E5B655CA99432B36DB6DB239194325Ø"
6538 DATA "AE54CB194328E499BE55C8C8CA2CF194EC"
654Ø DATA "B2DF2E59ØC8C862C8C8C8C86C8654C868B"
6542 DATA "78CA594B2A6516432B652C8CBB67594BDC"
6544 DATA "29E43239572F191947343Ø38C965A321Ø5"
6546 DATA "9532195C862C8C866B94B2864B34DB6DFE"
6548 DATA "B6DB66991979C9654CA796991943259588"
655Ø DATA "F2B65C31646432196D9194F23232C65Ø46"
6552 DATA "CA997ØCF792CA39759195F2E597CBEB2C5"
6554 DATA "325919572D7232E72592C8CBB66F957229"
6556 DATA "59ØCDB232AE51CA5919459ØC8C867BC913"
6558 DATA "65166D919699532324CF79196B95B296EB"
656Ø DATA "58CBE3319EF2AE516DB65F2FAC9654CAD7"
6562 DATA "196329B25926432D72591957259419AEØ1"
6564 DATA "5A64B2C65ØCA9957283219190CF79572D6"
6566 DATA "8B3AC8CAD95B2865E3296465CE5Ø6572E2"
6568 DATA "3259498B23219AE52CA192CD36CE3259DD"
657Ø DATA "68C8654C8679CAF95F2B64B232ED9EB2B2"
6572 DATA "3232E191CAD9195F2DB2E3343Ø38C9657D"
6574 DATA "A3239699532A6465CE46464321919ØC8E2"
6576 DATA "654C862CA197C6465CB3ACA19632F8CA8D"
6578 DATA "79499C64B2D32A33195F2B656CDØCØ16BE"
658Ø DATA "4646464321953219CE5265F2E72BE5AE32"
6582 DATA "55CAF9459EF23232338366F92CA4CF191E"
6584 DATA "499C64B2D18B232AE465FØCE325968C866"
6586 DATA "654C8CC678CA59ØDB6DB3E7232AE5F5984"
6588 DATA "4B29E57CF3183DE52CB76699197794323A"
659Ø DATA "B656CA1919267192CB46432A6467Ø6C545"
6592 DATA "919ØDB6CA65A656C8CA2C964998C8CABFØ"
6594 DATA "97ACA195B2A65C33CE4B2A65D655CBC3C1"
6596 DATA "6DB6D9E72AE5DE4B29E4B232ED9AE4B229"
6598 DATA "AE5E3232E728B219190C86465C65BE46ØØ"
66ØØ DATA "5C6636D91CBDE467DB1652CA4CEB232884"
66Ø2 DATA "65A653C965E72D32B67Ø1816464B24CDA8"
66Ø4 DATA "F2328B39CA594D9Ø67AC8C8CB866F95781"
66Ø6 DATA "28E49B6DB6CF1919632AE51CAB94B21E68"
66Ø8 DATA "7AB164655CA2CF196F919532F2CAE51658"
661Ø DATA "4B232BB6D8B2965A65DE499EF2965AE5BA"
6612 DATA "D654C8CEØ33DE432592CBCE57C8CA2CEB4"
6614 DATA "325968C5919ØC8E5B64B2B65664B24DBC7"
6616 DATA "6D95C8CBD64B28E55CBACAB95B2592CB39"
6618 DATA "8CC66992C932B9262CA4CEF29316526D1F"
662Ø DATA "9ØC8CB866D9432A65364B24C5919ØCC67B"
6622 DATA "5ØCA39195B219BE55C964652CB4CA6D988"
6624 DATA "D64655C8CBB678C8CA795F2C653CAF92AA"
6626 DATA "CDØC3DE432592CBCE57C8CA2CE325968BD"
6628 DATA "C5919ØC8E5B64B2B65664B24DB6D95C89E"
663Ø DATA "CBD64B232C655CBACAB95B2592CB879E3B"
6632 DATA "ACAE499BE55CAD9599ØCAD95B2965BB6Ø1"
6634 DATA "CE325968C59499E3232F395F286499ØCB1"
6636 DATA "8CB865F2E72BE5AE5C337CAB92C8CA5973"
6638 DATA "6994D9E72AE5DE4B29E4B232EF343Ø215A"
664Ø DATA "95B2B33BCA59195F38367192CB4679CA95"
6642 DATA "F95F2B329919432F197ØC5949B6C59435E"
6644 DATA "2F8C864B218B23219BE52CA995B2F19BA4"
6646 DATA "34CBBC967Ø6C59192C933C64B2F1973971"
6648 DATA "F6A6C594328E465762CA4CEF29652CBC2F"
665Ø DATA "33ACB4CA6CE325968CF395F2BE56663219"
```

```
Ø DATA 87FFØ4FF58FF91ØØF3ØØ35ØØ92Ø1Ø1Ø13DØ1
2 DATA 65Ø188Ø2B4Ø2F1Ø25ØØ26FØ2B1Ø31EØ378Ø3
4 DATA EEØ41CØ465Ø4BDØ5FAØ541Ø578Ø5EAØ617Ø6
6 DATA 74Ø6D7Ø7E7Ø729Ø753Ø791Ø8Ø8Ø822Ø849Ø8
8 DATA 87Ø9ADØ9E4Ø91BØ937Ø95FØ985ØAB1ØAC8ØA
1Ø DATA E2ØAØ2ØA2EØA44ØA66ØA7FØA9AØBB9ØBD3ØB
12 DATA EFØBØ5ØB27ØB3FØB56ØB8ØØCA8ØCBEØCCFØC
14 DATA F4ØCFFØCØ5ØC19ØC28ØC3BØC4AØC53ØC68ØC
16 DATA 7BØC93ØDB6ØDC6ØDD5ØDDDØDEBØDØ4ØD1AØD
18 DATA 2BØD3DØD4FØD6CØD78ØD9EØE56ØEBBØEC3ØE
2Ø DATA CCØED6ØEE2ØEECØEF7ØEFDØEØ1ØEØBØE15ØE
22 DATA 1BØE2ØØE29ØE31ØE3BØE44ØE49ØE56ØE5BØE
24 DATA 63ØE6CØE72ØE88ØF93ØFAØØFCØØFD1ØFDDØF
26 DATA EAØFF9ØFØ6ØF21ØF25ØF2BØF2DØF31ØF3EØF
28 DATA 43ØF45ØF4AØF56ØF5BØF5DØF4FØF61ØF63ØF
3Ø DATA 6DØF72ØF74ØF78ØF7AØF7EØF8Ø1Ø861Ø881Ø
32 DATA 8D1Ø8F1Ø941Ø981ØA21ØA41ØA61ØA81ØB21Ø
34 DATA B81ØBC1ØC41ØC61ØC81ØCC1ØCE1ØDØ1ØD21Ø
36 DATA D61ØD81ØDD1ØE71ØEB1ØED1ØF21ØF81ØFD1Ø
38 DATA Ø11ØØ51ØØC1Ø1Ø1Ø141Ø191Ø1D1Ø2Ø1Ø241Ø
4Ø DATA 291Ø2D1Ø311Ø341Ø381Ø3C1Ø4Ø1Ø441Ø471Ø
42 DATA 4C1Ø4F1Ø541Ø581Ø5D1Ø5F1Ø611Ø631Ø651Ø
44 DATA 671Ø691Ø6F1Ø791Ø8411Ø911B859BF6A5ØEØ
```

```
46 DATA EB4AFBA642D1115572D31F4244351E93975B
48 DATA BBF9D666423AE7BFF30ABA8DD4EF8D4850C4
50 DATA 215CDA0683143F28435CAC4C21266C73B17D
52 DATA D5B7EA5178F2C5AA6168C7E8DACF6CCCB2BD
54 DATA DF0AC71CB18A5D3D968C3DEAEB3D07066C15
56 DATA E386C869CD42607BAD2776ADBE1A12679FA1
58 DATA EB4231D4FAA9D2AF6CCC8C37DA37D0749C1E
60 DATA 081E998DA48A3C5D56CCB0FFB6DD62B769A0
62 DATA 4D4D5488EFCAC0F6AD673802C0DAF8BC2CEA
64 DATA 977CF69E8E1AC292FE67357BAD277A160475
66 DATA 5B88C86E1A6187215C3E92943D61549987BC
68 DATA 62651B6EB62064B756F1FE2044726DEEAA7B
70 DATA E63D968E5AF3BB49D15B5D338F58D26F23A1
72 DATA 501987ED2763DDF8169C26A35BF1298A396B
74 DATA 39679F88CA9830AC919D3D4AC0778890F644
76 DATA 5A06396AC5C8B455B45157334B2D54DD96A3
78 DATA DF99AD5B9C74D6B5E3E9949F1BDD212C5FB9
80 DATA 769BBA9621EF346F867F9D61645A07216FC3
82 DATA 254B5310D8B9459FF883CB0CD083F4A699F0
84 DATA 37216E4820778890F58A16CA1CCF403F772C
86 DATA E8B7D07DEF11D22D1EF4C3943C65BA670E1B
88 DATA A9AB34A5FF8ED943F24D29D5AF432C45B9A5
90 DATA A006D8B896E048BA7128D3334455AEB0E81B
92 DATA A9AB34A699A55B783037DA3EEBCF6FC93826
94 DATA 87EECECB4709538B4711C5DAEFD935B25772
96 DATA 07BCD0E1C2B30BB08F076DE8337FB6DA712C
98 DATA 79CCB2FB4A994B4377933C1B3F1115216C42
100 DATA 43DD71329B94B9813BF0A951AE3152620037
102 DATA BDAC8A8E44A4C6DAF783CCD7A3A0E5D91B7F
104 DATA BF33A3EC42B3F04D50881C189DEEF9E62B70
106 DATA 54B411DA761CC5BF9F9ACB7A116F20E0EB39
108 DATA AD0070B34D3205A20506D1C06E59427C429D
110 DATA 8E727D6FEDB563B4E90AB88F2D3088A537AB
112 DATA C8BCB6DB27873802EDB380E0C1F714A89750
114 DATA 207016929C006F770DBC4B59B9A0001F7728
116 DATA 2B8D25AB649ED0F7C76E3A161C5E8B461EAE
118 DATA 05A20506D19FFF08E0384D30942D7C49D48A
120 DATA 12F2F1B56FF4D5169968B3305E109C006F77
122 DATA 19A0001F6D09F93AC3C68EA203DCD55C4369
124 DATA A1D9D2F70BD2E6DF1F0ACCD2F1F5BA17DDF2
126 DATA DFC1AF842667E0226C7C50BB62F7DA37C5BF
128 DATA F2B607E515BEA66811F5BAC479A155EF34A2
130 DATA 74ACE875E685A3EBEA1E845B9D041EBD6536
132 DATA 31EF8FCC96AC83F2DA0D80E063D38C5DAD4A
134 DATA F29DA2F7DA506A728C536EDBE685B859B84F
136 DATA F04C83E5D4976FC7649987BCCBE62B6E97B4
138 DATA 214B531F77933C1B388F2D3088A6E21DA2F3
140 DATA 59BF99FFF58A035306CE44A4C750207DED28
142 DATA D333B00FBDCADA2DC9A13173D2D4428242F3
144 DATA 827A550E90A62138C3E06D01768DE991272F
146 DATA D77DA5A6C87DA529C5B8ECB8EA4C83E5D497
148 DATA 600D246A60679CB5DCB57C49D48A4FDC6112
150 DATA BAB90A4C83E987C8F9A23CF476ACE6DBA837
152 DATA D350BEF3053BC735AA9CE2FBC29E8EA27605
154 DATA E9879D13357054B422E31BE8CA5D9C0079BB
156 DATA A999982097C1528D7C5DAE1DBEC3B52501D8
158 DATA F6A57859B8AA7A84AB2C93A89299EA7122E3
160 DATA 1A94A7A48CC681DEDDA6475307E138DF8D6E
162 DATA A7E1D2A3F31BE5A346FFF783CCCF4010D8B8
164 DATA 459FFDE88A774F59A96A1A63EDA53B236189
166 DATA A8C4340CC3A3BA8C83D53E3BD871CD022295
168 DATA 2EB0F0EEC852467E2FD761B54A130B51FD0D
170 DATA 31B0FDB2CC82596EBB9969DA76A17A2E4BF7
172 DATA 66577812D2679F9ACA534D30804103F0A951
174 DATA AD482076044B47BC623783E5D4981A17A251
176 DATA 8765E3DAC3B214E2296AC5F82DAB5037D698
178 DATA 26406B29CE13F0063288E1BF9FA7D2B15137
180 DATA 122D5B6E294DF6216C8A340DBECAFA221A61
182 DATA AB16D8B38DB7EC4BB109F19EF173D2D6B1D4
184 DATA BF69AFC325473E64C98D80F63C4B50BFB538
186 DATA F0063CB4B9749E1425A4BCE8E784934CC8B7
188 DATA D0760246EDFF7C5127C1BB6FCD4224446D5A
190 DATA 89FA2F6AFAA0795170DF8FC4AA7BE6984SD0
192 DATA 428202679EF6097DC723503D207B1799BBCE
194 DATA DF6956CBED48207A5E2205065E8F2780BF11
196 DATA 53A3BF111521600C803945C04754EB3E9294
198 DATA 9A92C0F81525E685B109D3891DB38F79177D
200 DATA 545E152A899FFDE88A770E794C37FE23E7DB
202 DATA 5C547569CFD97439985C735E413D9698217F
204 DATA 00EEE874AA40DF307959D33CF621DC7C89C4
206 DATA DF5735257CD556679D1335CCB1140BABDA06
208 DATA 708D73EDADB55BE7D451573343FF61402DE3
210 DATA 7C49D48DE3B08F076DB6F843F0069968B333
212 DATA B010D8B9A96963A8EB44493AC3D948182653
214 DATA E81AEF5978F0E1C3004441E184B52DD2684E
216 DATA FAA036DAF7D62A297800BF1306C9B9A5A2B0
218 DATA F58AC0FDCBDE1CFA0BE7D4515733444F3BF0
220 DATA A941AFB8BBC1A6625EF09C0038CAC9983090
222 DATA F723503E12F6673D62EC8F0971CEDA986BB8
224 DATA D48D5D6171440099BB70D056CE4203B33770
226 DATA A52AE69888BD9696655BF1F5BFB2DDA1549E
228 DATA 307DEF786685B859B84FF04C83BBA23A7238
230 DATA A68036DAEDBD207B1799BBCEDF6956CCB20E
232 DATA 29410EB0F44F088D5C8688ECD4E4681987E5
234 DATA 3432CC3F8EB0F07055594DB662512351E81E
236 DATA B2A34CCF403C2B78F07DA5F7F2FA52137116
238 DATA 8A0D50D2355236FFEDBE94A8708D5C12034F
240 DATA 690E1BC10F8426680837F6933E1D1B37C5EF
242 DATA 347AC2304D0B61676140546FF123DDF68A4F
244 DATA 1E680800562D8DBD290E45707768EF523750
246 DATA 207957C60B0DC99FEDBEB69328DF8D13DA75
248 DATA 1C1BF55533DD212C5F0AC7879156E81D3281
250 DATA FE60C7D0E96E1A61AB16D8B38DB579AEEA7C
252 DATA 6909F8CE38B9A96A18E91FB496B613F0A951
254 DATA B98C37BC6237838B34981A17903C6A115DD5
256 DATA AF79781C79951DA6655070B3EA4C371E12F6
258 DATA 7C19A8B36D6519C151FADE8501D0ECB2C20F
260 DATA 0420007F741171C22295398C83D950A77CAC
262 DATA A99D3B10AE357A122C5C129C413C6A866146
264 DATA 90A6013EFA089DB60B49E11B5310D8B9459F
266 DATA FC2C8EDD54A80BE65C5AEDD0CB5B530F83AB
268 DATA DC1E09490D48297B4A929C00383E92949A92
270 DATA C0EE8898217F9037D68E580D3170A160401E
272 DATA 245E38520CC66F4D4DE4AF06AF0F0FC3DC27
274 DATA BA3F6C4D23A15E2E44B6659F5CB42399E22A
276 DATA 9757340EFA964F5BDED50532CDA8B40E1951
278 DATA E3E0E8F081C4DDC04484E987C53783827128
280 DATA D16B398AD2679DE091C8BE1519A0000FB547
282 DATA ED9D3E24AA3945D9D8F79667E037DA37D06F
284 DATA CD025EF09DF89C9203B30EDAEDB634229875
286 DATA E697001BA83E9294A2E0E91C3EA4B1B55C6A
288 DATA 608A0037BB411D980E230BB710ACB1072761
290 DATA 9E22625A73999837DA37D0749A062D7BF8AF
292 DATA 649987689DB55B6EE845B0EB4D308037D07D
294 DATA A529C5C7FD49E38CFC104D3B5837DA37D06F
296 DATA CD023C21BBF1B92687689DB55B6EE845B0EB
298 DATA 4D318E3AD68F33888AD29089FBD3A2FBCB7D
300 DATA DB7FB3FF6140789329168B476DF4A98DB5EF
302 DATA C733B037DA37D0749A062C9F334B50F1293C
304 DATA 882A5B8501B7EC4BB109F0FC28BD0BF16FF3
306 DATA A2B91CA3F6F62A1A58ABE444401E351631A4
308 DATA 84FC562E59AC258E0CC66EB0F7F9D485A6E3
310 DATA 1C34C5539EB9AC411582003EF58AEDBD0B59
312 DATA 9E23F306D14820794A0AB4CD968D941AEDB6
314 DATA 66F6323AFE82071A97DA07999085B859B8E8
316 DATA CA1B00DF8E139A8641CA6A177869CF84251D
318 DATA AF6AE685900205064C9C79BD35BBDFAFB382
320 DATA B751FE237859B6D0D3E070B3F0A018CE9218
322 DATA 72566A523E780C3C4B7621DBAD6789EF346F
324 DATA 219B9880534A9171CDB57056542CCE1E154D
326 DATA 2BE079AEEA676DE92BDF662D80DAF0375CE2
328 DATA B1C6599EF169D27C90D1EBCD16CD830F13DB
```

```
330 DATA ØBE5D36FFE82Ø7BCE29EØ314ØD63BCF4E8AB
332 DATA FC6DD3CA2C5BØBB4AD3ACFDB35E9Ø66F983C
334 DATA ØE4A4DC73E15ØDØ2ØEC95671CEA62519755A
336 DATA A5C5E688BAA524FA1351FØ37FB411371CDØ2
338 DATA 4421D86C6F53CB483E891DBE84EEØ9DCE685
34Ø DATA BD8DCBCC48BD968C838D4AD5F24D27A3E2BC
342 DATA D6C8F19Ø354EFAD94E1534B6FC2C9A927Ø78
344 DATA 3C12Ø1C23F26578577A9A5A35F9976A5ØEF2
346 DATA C32ØEB3D645EB832Ø5A2737A59CFØ78D9A92
348 DATA DØF79C85Ø1EEAA5A466F98336D66862ØA7EØ
35Ø DATA F8E6BEE55421DDF6A977FD68B7C151B55F9A
352 DATA 5ECE8B5C2C4D76EØ5ØF2C23D459998ØF13F5
354 DATA 23CDFDCC789ØF68Ø2A569F18ØE3D46DAEDB5
356 DATA ØAØFD7E5AD486215333F7BD194FØ9CDFBØØF
358 DATA BB4247186B52BD76AA9Ø5BF522FB42D4B97D
36Ø DATA D51E8B4B471EE3DD36B26652FEE98ØF7DA39
```

```
1Ø DIM Z(31):FOR T=1 TO 31:READ
   Z(T):Y=Y+Z(T):NEXT
2Ø IF Y<>1586 THEN PRINT"CHECKSUMS
   WRONG":END
3Ø F=Ø:Y=Ø:H=OPENOUT("CODE")
4Ø FOR T=1 TO 2Ø4:PRINT#H,FNW(4):NEXT
5Ø FOR T=1 TO 1165:PRINT#H,FNW(8):NEXT
6Ø CLOSE #H
7Ø IF JM<>13 THEN PRINT"ERROR IN LINES
   41ØØ-417Ø"
8Ø END
1ØØ DEF FNW(D)
11Ø IF F=Ø THEN READA$:F=1
12Ø B=EVAL("&"+LEFT$(A$,D)):
   A$=MID$
```

```
   (A$,D+1): IF LENA$=0 THEN F=0
130 JM=ABS((JM+B)MOD 100):X=X+D:IF
   X=320 THEN X=0:Y=Y+1:IF JM<>Z(Y)
   THEN PRINT "ERROR IN LINES□"; Y*100
   +900; "□TO□"; Y*100+ 1000: CLOSE#H:END
140 =B
990 DATA 1,17,80,67,79,6,43,58,38,96,63,68,38,80,
   6,7,62,80,80,98,83,20,50,45,9,51,22,42,
   54,75,68
1000 DATA 000C008A00DD0116017801B802160285
1010 DATA 02C102E9030C0338037603D703F60438
1020 DATA 04A60500057705A505EF0649068706CE
1030 DATA 0705077807A30801086408740B8608DF
1040 DATA 091D099609B009D70A150A3C0A730AAA
1050 DATA 0AC60AEF0B150B410B580B760B960BC2
1060 DATA 0BD80BFA0C130C2E0C4D0C670C830C98
1070 DATA 0CBA0CD20CE90D130D3B0D510D630D88
1080 DATA 0D930D9D0DB10DC00DD30DE20DEB0DFF
1090 DATA 0E120E2A0E4E0E5E0E6D0E750E830E9C
1100 DATA 0EB20EC30ED50EE70F070F130F380F50
1110 DATA 0F550F5D0F660F700F7C0F860F910F97
1120 DATA 0F9B0FA50FAF0FB50FBA0FC30FCB0FD5
1130 DATA 0FDE0FE30FF00FF50FFD1006100C1023
1140 DATA 102E103B1061106F107B1088109710A5
1150 DATA 10C310C710CD10CF10D310E110E610E8
1160 DATA 10ED10F910FE1100110211041106110
1170 DATA 11151117111B111D112111231129112B
1180 DATA 113011321137113B114511471149114B
1190 DATA 1155115B115F11671169116B116F1171
1200 DATA 117311751179117B1180118A118E1190
1210 DATA 1195119B11A011A411A811AF11B311B7
1220 DATA 11BC11C011C311C711CC11D011D411D7
1230 DATA 11DB11DF11E311E711EA11EF11F211F7
1240 DATA 11FB1200120212041206120812 0A120C
1250 DATA 1212121C12271234
1260 DATA 6B3F5A384B6BE1D0D2C2A77BD4F25691
1270 DATA 36C4439F5C17949E6756FA3BA3643BC2
1280 DATA A8BB13D7D4004FE5155A0C9D3170A0D5
1290 DATA 359CF25B12DAC4D21743C77EA5865BE5
1300 DATA 5A348F2F7E9416BECC0EAD95F0E52BD3
1310 DATA 18D371B468E1D3BDB3B6DEDBC17E70E8
1320 DATA 86743866073ED4A46A8FD2C226B9E1E3
1330 DATA B456FA81AA471D3BCC0E2BA55A38C0D0
1340 DATA 1C755038191F881FBC8B248E31CDD65E
1350 DATA EEC53A001C9A8E2B96F8EDD4720FB4FC
1360 DATA 34808BD6DA8BB70DD777B1CEB4E1F069
1370 DATA 8BE637298FD2C2576D4778A1F98694B8
1380 DATA 2D7220A65129F1C3A0492DD69A89F766
1390 DATA B8E42118217EF2D6EFED73C43E667C2A
1400 DATA F4DA8F165C514A3B590F34DDA2A37052
1410 DATA EE071AD0F95D64A7A4A69D968BA9F2DB
1420 DATA 68B96CB9994A891F442B80A1D052AF67
1430 DATA 5D24021EEE8176914D72D15AB5148D15
1440 DATA B5CB32CD17A9853707578BE698AD551D
1450 DATA 262885FA774BA8F70EA7BDEE2D7CE8A5
1460 DATA 07A0011CE116B95829F1BBC856C43453
1470 DATA 1E68B12E54C31221C6291D21BBC8AD7F
1480 DATA 021E6892E5625D243034E7B25ACB3D10
1490 DATA 9B1F142EA78B94C46FE550BD239A8E19
1500 DATA 2D6B0A87D6EB9F29AA931C512BD18B75
1510 DATA 88698E11452ED601FC2E72B831CD544A
```

TROUBLE SHOOTER

- The text used in Escape has been run through *INPUT*'s text compressor. The resulting hex code has been put into DATA statements. Unfortunately, you can never be sure you have entered coded text correctly, simply because you cannot read it. The programs have been written to incorporate checksums which will indicate if there are any errors in the data. If, when you RUN the completed DATA program, a checksum error is reported, simply look through the indicated line or lines for a typing mistake.

```
1520 DATA 5AAC8B152D6B0A076068C6296B3F5A38
1530 DATA B8CAEFD04EEF0727D30AC7CC4512C78C
1540 DATA B5DA6FDB8773D7B342E250BD0FB18BB4
1550 DATA B3E9ED08F1DB368032CDF92DCB9ACAFC
1560 DATA BC94F7449512BF1CC343EC22C64CBC37
1570 DATA AE604EE58B544AFC38262D23D0DA8303
1580 DATA 544AFCA8115EB66DA1234333809BDA65
1590 DATA ED23343F4E70B4C2199C89D0E779EF1D
1600 DATA B5D471AB1DF6DB919B1FC04570917B4B
1610 DATA 3A6BE1A0A685032D1C2D9C690F8E3A28
1620 DATA EE13D6729F73F014B76D83EB5948AB1D
1630 DATA 886985C462BD3545E5B6E9451BC0453C
1640 DATA 0A07A06D48A5C40F23038DBA0338262D
1650 DATA DE46BC37D01CADE529F720C0ACA58EAB
1660 DATA F8509FE417BA6F47470B5F9CA385AF9E
1670 DATA A05107853960097F2E1431CD8B544AFC
1680 DATA B771F392B2A8AA7F6366D232008E082F
1690 DATA 3413EFCD0AED2030C7433B76DD83A30E
1700 DATA 6AC35D55F852DA21E066D38BD34C0B9F
1710 DATA 3074EC2384BFE6BBD04C0B5F7DEC23C0
1720 DATA F8E2BCD0C045385AE687B7726926BF95
1730 DATA C53AF691F0D5A2F9ADF4A3B486667668
1740 DATA 1F6AEC23051D5C0437E5BE9ECD0FF0B1
1750 DATA F303AD16E1000E5A5E0CD4E39E724B2D
1760 DATA 3D5AF8229D8A614EC0D08CDB50385A38
1770 DATA 39034D70F23B2695EF812639DBEAF952
1780 DATA 5368EDC5E53DC834242E072F29424C2B
1790 DATA BD8887D880067036D5E0621DF87AB621
1800 DATA 34EAA8942AA5CD0F3D108666492E5ACB
1810 DATA 5274B1A2C283C2D5D57B02F4A1A7100F
1820 DATA EDE14339698EF6025730A79248A7257E
1830 DATA 452A257E6AB96CB99539034DA9033826
1840 DATA 071AF81ABF2D972DF32295129908AFF6
1850 DATA 6852D4D54B3E501FAFE71DCDE1366BB5
1860 DATA 899AC241E229A4F72A80A13942FC629D
1870 DATA F6A30E9F1D886906D471B5149BE4A2B5
1880 DATA 1C5E4AE929BCF9011721189AFC8ED2C2
1890 DATA 3E1E2E5E812635C4F8A676D9FAAB385A
1900 DATA 132D2B856A9A12A99AE4A2720CA52795
1910 DATA 5DDF01C78754C7A70DE0B8E252E2276F
1920 DATA 191973A419C031690CCD4478C5BA5811
1930 DATA 0AE97DA0295ACF786D649A6BE124BBA6
1940 DATA B4C5288A3AA4430DBED6038D4D72583C
1950 DATA AE96A20348EF70B1AF7FC653CAB6E1D8
1960 DATA 7D528B147DB1110ED9834CB3E99A3B6F
1970 DATA FAA2F6DBE6F8CB2F5213F8584A9B1F68
1980 DATA 0031CD5429F18342A0492D523113E165
1990 DATA 8023DE9B6652993352226068DB636607
2000 DATA E394B343F9456BA938D0ACAD41A69956
2010 DATA 144E2AEB89B20770A81FC06138D1B252
2020 DATA 6FDB2E9222764EA90EB48BEC237ACB3E
2030 DATA 172B629AB80DB4580A314C6C74719F71
2040 DATA D531D752C42F6A3F99B348A5006E6352
2050 DATA 3613EFCD077039357539B5BCA5A5151E
2060 DATA 8567E93CB8484D134782775052FC006E
2070 DATA 703BC2A745A4434DFB095BEDA17A6BAF
2080 DATA E0F052F97C2AC50FD1C29966688283C2
2090 DATA 5FE2BD190F34C991E6251FA8DAD7F38E
2100 DATA 524CB3B5E2BD1968E97D5038F113780A
2110 DATA F1433A2D002E52295CAC03D0B356751C
2120 DATA A95129F154E60C3431189A999D8A530A
2130 DATA 977A0FB4955FD47E7DA0092B8E780AE9
2140 DATA 7E38CC7ADCDC67244F6AF555183AF4DA
2150 DATA C15FF35DA199163EDA3B208057904A1D
2160 DATA 94567ECC97885DCF5771421FBF492DD6
2170 DATA 079AB5354C732D4516EB6245EF089F81
2180 DATA 6DDBE23E2D4585BE0E404C734FDE1A14
2190 DATA E648A5C48EF0103B11E165DB691919C0
2200 DATA 113034336A29BA58456BA9E3D1433BC9
2210 DATA 54A619C85A6F1B68C461F1F8E2C14580
2220 DATA D3ADB604A17A4FE8D777DBB601F82AAA
2230 DATA CA86143FB122A639CF408B753EE7F792
2240 DATA 1415FAE21431CDB5544AFC2EBC4CEE8B
2250 DATA 7B336622C80670C4A19949CBC95D2480
2260 DATA 2FF90C342EEE737E1CAFF0D0BB9EA9F5
2270 DATA D6ED48950958142F0515B7A33B38E474
2280 DATA 5212774B8B9069B6557E69E1FB631FC7
2290 DATA 492DDA35F7E607FB3860689E7050385A
2300 DATA 082F034D498EFC8EDBB66DA0251FA86F
2310 DATA D7F38EE64CB3B5DAA350EA8322145DAC
2320 DATA C221B7237A39553BADF981060310D30C
2330 DATA 751C5CACF86DB3569AD4A8944C2A7306
2340 DATA AB3D0CCD257E7079D2FB72F80A17F114
2350 DATA B5D3D00E6D00B753F0A914BF8313DC8E
2360 DATA 9B0FE950A6858FC2763888699B1EBE94
2370 DATA B4F04538CD31427B385A380C1C1C7550
2380 DATA C27D1749629A2794838B7500F143AA6F
2390 DATA 5BDA7D5C68D4EDD4E2F1B51E1B6892C3
2400 DATA 2AA5CD6F164503381D07A79DA8F7AC55
2410 DATA D1C2774BDA5504E29FA0AC075AF45198
2420 DATA 8A98E65B032DD6C5DA6BBE6D9E427A9F
2430 DATA 8A2EEE33A73AE65AE4AD45ED8E544AFC
2440 DATA DE9B6D6332328023606866D212EA3D10
2450 DATA 7A2FD6DD96F91DF851E5A79D4D6AB4F0
2460 DATA F7921FB7B4281AFCC29966ED865EFBD1
2470 DATA B36CD1812086104270BC043D63AEA5E8
2480 DATA 2AB4F640674A2B3F8D4BC4AE17EB84BE
2490 DATA 4F70A76451F8A17A4FE029C46D4782DE
2500 DATA 4798D2E22E56C4340B1F68B14A6DB643
2510 DATA C57DBE98B514DDB63A0031CD94F8C1C5
2520 DATA 069AD4A8704C2A734AFCE006034D6A54
2530 DATA A199082F56381080B10ED88FC0612171
2540 DATA B85FA41FEFC78C532FE5CD4E8F102F07
```

```
2550 DATA 3A285CC4A34EEC40C42FDEA2DCA0E5B7
2560 DATA 629AA3B5B458172BCF977A0F85D65E5C
2570 DATA 68524C332695391B108F163E1CDC551C
2580 DATA 849856483F388053CB1695128126B598
```

```
10 PCLEAR5:CLS
20 FORK=0TO359:READA$:T=0:FORJ=0TO12
30 V=VAL("&H"+MID$(A$,J*2+1,2))
40 T=T+V:POKE3074+K*13+J,V
50 NEXT:READC:IF T<>C THEN PRINT"CHECKSUM
   ERROR IN LINE";1000+K*10:END
60 NEXT
1000 DATA 000000E25A3F70F4385AD2DD27,1351
1010 DATA C4B4A455BCB527D0F10DA7A504,1831
1020 DATA FAE1DFD56CFE2A76F16943C6AE,2218
1030 DATA A3953C0352743168575681C0C5,1417
1040 DATA 6FCCDB6E1AE9626D104FD8E862,1751
1050 DATA FCCEDC35297C36DF9A2D5F12D2,1695
1060 DATA 8FD56D6876669D7CBE25A36DB6,1751
1070 DATA E3A6317BA7C4B47BD6D67D0E1D,1827
1080 DATA 586A4D43E0E25A51ED5C7C3926,1507
1090 DATA 81FAADAD0EC751B6DEAA94AC3B,1972
1100 DATA 3343E25A3850751C1F8853E1C4,1386
1110 DATA 91778BDAD9A6DA00758BDC571D,1814
1120 DATA 3439A9DBF12DF9681EE5AD1700,1591
1130 DATA 681E2F6B3AC5DF5DA7C36DB86D,1623
1140 DATA 0A4DF9A2D5F12D28FA178A9DB8,1789
1150 DATA 9486F9A6D03916E1F894A8EB16,2030
1160 DATA A4D03DD44CD0C08792E35BC9F8,2169
1170 DATA 8719F717BCA9F198F8968FDC3D,2002
1180 DATA 0DDA528AE6E9A07ACA93851D16,1729
1190 DATA 80D036FDA3B22EFCAD3BF6D49F,2131
1200 DATA 99531772D972D03F12953343AD,1433
1210 DATA 119EBD4B407808917645DA07B9,1373
1220 DATA 6B4649A145D14B5CD3B2EFDCDE,1926
1230 DATA 16A45F9A2D5C23AAB59F50A504,1366
1240 DATA F7A84B77ED77EDC3A97A1BE25A,2031
1250 DATA 3803400EB1722DC39177E252A6,1406
1260 DATA 69A145D62D03C422586A8423A5,1353
1270 DATA 38CFF5C8BC9340F01122EB172D,1701
1280 DATA 9739A18040F72D68B8507E6F12,1476
1290 DATA 522E9EF54395BC66534470E14D,1602
1300 DATA 65A533FD7ACA2392754DB6DBD1,1879
1310 DATA 2B118E69A00758B916E1C8BBF1,1622
1320 DATA 2953353145D62D0385359694E3,1268
1330 DATA 3430E25A3F70F43BF33B82707E,1564
1340 DATA F4ECCC70AD386DC49177EE16B6,2036
1350 DATA 7AEE70F7AA1C4856916DB6EC43,1814
1360 DATA C54EDE9B3803F12DF9CD3BF2A9,1921
1370 DATA ACB44DBDE52F072FC4A548BB10,1584
1380 DATA F0F751331CA9CC15DBF129522E,1670
1390 DATA 32D263803DAD0A8FC4A548B6DB,1708
1400 DATA BC2266868E85977EE6E00FCD08,1692
1410 DATA FB70AD1C13B42267071DD5E79E,1538
1420 DATA B1D4B591DBF61D45C01F9B4B74,1847
1430 DATA 46141C2D670A2D01F12D334F89,875
1440 DATA 68E141D470DCB584FB853C1CE7,1954
1450 DATA FAE0E3B5690B36DB6DB890AD31,1930
1460 DATA 08A6B7AC48BD36DCA788B8036D,1663
1470 DATA A0070A0FC4A548BA8D038CB498,1427
1480 DATA E01BA88F72D68E686083DCA6AE,1923
1490 DATA 3A96B3927D43E2A76FBA179C5B,1685
1500 DATA 7C4B479EAF8968E141D4681FC5,1678
1510 DATA 56039CD36D142EFC4A548B92F3,1569
1520 DATA 71DFEAAA2E32D2666377844700,1569
1530 DATA 7BA899A18083B429D8ED0F1C3A,1639
1540 DATA 8E0F7F695DC36A21B7694BE22F,1452
1550 DATA 4D9C13E25A669C7D8E86177CD7,1589
1560 DATA F08BE25A66868FB1F742F38BE2,2172
1570 DATA 5A3845C072B787E695BF266A3E,1615
1580 DATA C758BF345ABE25A51FAADAD0EC,1875
1590 DATA CD0C8FB1A87C117074167AFB94,1617
1600 DATA DEC7C03F345AB681F9541C01C3,1686
1610 DATA C7A818BC5A96E53C45B7C4B47A,1858
1620 DATA 9CC3153B5C6686E25A3850704D,1400
1630 DATA 03E55263BF2392681EF52F9EB8,1553
1640 DATA BDAD0A669C1EF29783971215A6,1540
1650 DATA 2114EC43C45E9B3803400EB170,1227
1660 DATA 6DB6DB6A90DB3D7C4A54751A07,1472
1670 DATA E6D295334340F72D68B9268517,1546
1680 DATA 452D5C2836DB785E804F7AA1E2,1449
1690 DATA 14F427287C3DA05ED1CD395398,1488
1700 DATA 2BBF12D3AB0FC4A548B96CB96A,1666
1710 DATA 4D03E55263803A91AF81A072D9,1616
1720 DATA 72FC4A548BCFDDE1133ABA9393,1873
1730 DATA 40FA81F25E734779EBEE16B428,1801
1740 DATA 3853513EF4853C473430AA758B,1316
1750 DATA F10A7C3ABFDB19A6B1D14B56DB,1701
1760 DATA 8EA5AD1724FAB4A5E1C01F9BC2,1931
1770 DATA 99A180845F12D28EFC5E2E1E3E,1523
1780 DATA C4352681D977EE7C4B47t577EA,1634
1790 DATA 14ACB44EA44A69A9CCE2E49B52,1857
1800 DATA A4F4A198E03BEBB4F87D521F95,2054
1810 DATA B8E00DB5BC9F894A91CC719693,1919
1820 DATA 1C01DE113343428BAC5A07DE90,1226
1830 DATA A787452D734E36D35D9270C514,1442
1840 DATA 62D0350EAC3A8D03D6BE3C586E,1409
1850 DATA 49A07452D5D62E0FBD214F19FE,1499
1860 DATA BF636DB70DB650A45A93E8708D,1743
1870 DATA 8BED9A707B2DE771BD74EDFB84,2079
1880 DATA 5F45F97F1CAC7C09A9340DB6FC,1541
1890 DATA DA52A669A0070A0FC4A548B526,1415
1900 DATA 81B77844CC7BA88E00F954999A,1777
1910 DATA 1889481D998F6D0ECE66E3A96B,1492
1920 DATA 45F956B3410AD334E0F595270A,1588
1930 DATA 3803D944B0E00DB7EA14ACB44E,1624
1940 DATA 248BBEA539D88BB22ED0371F65,1561
1950 DATA BD11CD3428BAC5A06DB6DB6DB8,1849
1960 DATA 6C4C310A719F717452D731D53F,1366
1970 DATA 708BF129522F95498E00F75133,1405
1980 DATA 43D4E5C01EF2D4E5D478569694,2129
1990 DATA F3A59B6DBC289A6A45C283BC12,1760
2000 DATA 3B7007E2953E2A7704D43AB117,1250
2010 DATA B56C27ED77EDEF543F329786E0,1866
2020 DATA 0FC52A7C669F12D1C28382681E,1455
2030 DATA EA2377E4724D03EA07C979CD1D,1607
2040 DATA E7AFB85AD9A6A4D03DD4470A0F,1804
2050 DATA BD214F027E25A7561B6DF894A9,1420
2060 DATA 17006801D62E0E3AAB59F894A8,1284
2070 DATA D49A07CAA4CCD0C0C4294E2A76,1818
2080 DATA D03DEA5DFB845F994AC2683BD2,1868
2090 DATA 14F1179EAD070FC48CFB9B8ABE,1707
2100 DATA AD49FB87A030BBE6BF827C4B4C,1853
2110 DATA D0C080EF68752A415F31F99694,1786
2120 DATA CF5D88971F426DC55F58B527E5,1622
2130 DATA 5ACD03A296B9A6851758B40FE2,1626
2140 DATA A779F717D0A8A5AE69A0070A0D,1568
2150 DATA B5BC9F894A91CC7621E2A76EF0,1982
2160 DATA 88E00DB6E32D266686428BA296,1618
2170 DATA AE3A96B45C93B43D1C90A669B7,1668
2180 DATA 51A07AD7C78B0E24022E0F1016,1067
2190 DATA D5BA7D09EF543BEBD5157C009F,1667
2200 DATA 8A43651CD36DA145D62D03F92F,1442
2210 DATA 7E73EE2FA1514B5CD36DA285DF,1773
2220 DATA 894A917DC9DE11331EEA238036,1453
2230 DATA E32D266686917724D03F92F7E7,1741
2240 DATA 3EE2ED0F15BC5D46DBDCAA476E,1702
2250 DATA B178A2C04D1DB8A82BA721C1DA,1763
2260 DATA 5BB89295B34E117C4B477E6771,1456
2270 DATA C7D8DBFB35DA2D49FB07E6F79E,2167
2280 DATA 6860E25A3850704D03BC223BF2,1367
2290 DATA 392681BEA07C979A3BCF5F70B5,1657
2300 DATA B34E0FA9428EB16DB6E8A1111D,1556
2310 DATA B90E0E3B55397A0681B6DF9543,1292
2320 DATA 34D003AC5C1C7556B3F12951A9,1469
2330 DATA 3681F2A9333430F70B79707E25,1399
2340 DATA F872FBD214F1170A0ED0D3BF68,1845
2350 DATA EE00FC52A7C54EDC138350EAC3,1893
2360 DATA E6F0A36DF12D334D070ED297C3,1733
2370 DATA D36708BE25A3DA118E6860E25A,1605
2380 DATA 3850751C1C49177DC294279A6D,1174
2390 DATA B6D003AC5C1F5487E56E3ED4B7,1703
2400 DATA A9DBA8DB40F5AF8F161C9340FC,1915
2410 DATA DA52A38036DB45169DFB7071D5,1801
2420 DATA 5ACF7A84B77C4B47880DC55DA0,1603
2430 DATA 7ACA09F9851F45A5BE69A145D6,1719
2440 DATA 2D03E6BDFB9F7A429E33EE2E8A,1696
2450 DATA 5AE63AA7EE116DADE4FC4A548E,1862
2460 DATA 63DD447007196933343040F7A8,1267
2470 DATA 4B7758BDEBE077E569DFB7E654,2103
2480 DATA 6DF12D1A938FC97BFE0D145A77,1531
2490 DATA EF34F8968FDC3D130745B2CD08,1599
2500 DATA 4380F426BC70E8A5AE6340F6B4,1937
2510 DATA 2A36DB6FCCB4A67AEC44B8FA13,1855
2520 DATA AC5D929DC13DEA8F158E214F02,1476
2530 DATA 7EF4268EDE2D2984734D0A2EB1,1415
2540 DATA 681F12D0EDED4A98BE7DC5D14B,1857
2550 DATA 5CD36DB400EB1707E252A35268,1514
2560 DATA 1F2A931C020FC4A546A4D03BC2,1321
2570 DATA 26686040E15A3F603AC5C48587,1495
2580 DATA 006DB6DFAA97EE14E331FBD3B1,2008
2590 DATA EF29783978847E22E141C29481,1630
2600 DATA D89D4745BC5F896FCB40F72D68,1707
2610 DATA E64C51758B40F7A97CF5C5ED68,2030
2620 DATA 5334D49A07CAA4C7C4B47880E2,1923
2630 DATA AEE0E242B4C4229B700707E252,1689
2640 DATA A396CB98B52681DE111C93E25A,1746
2650 DATA 668640D521B67AF894A8EB1769,1777
2660 DATA 63E059A180E25A3850704D03DD,1566
2670 DATA 4477E4724E0DB7691DBDA118C3,1506
2680 DATA B334D0038507E252A45B70B48E,1579
2690 DATA DF5298E35F045C01C1DA14ECCC,1747
```

COMPUTERS IN CONTROL

A look at how your computer can receive information from the outside world, process it and use the data to operate control devices for a variety of systems

Microelectronics have revolutionized the control of all sorts of domestic and industrial machines and manufacturing processes. In place of the dozens of relays and motor driven timeclocks that used to control equipment like domestic washing machines there is now a single neat package of microelectronic components. And in industry, computers have made robots into a practical reality for all sorts of applications.

Microelectronic control systems come in four main guises: wired logic, custom integrated circuits, semi-custom integrated circuits and microprocessors. Of these, the biggest revolution has been the microprocessor, which has widened the scope and lowered the cost of microelectronic systems by introducing the idea of software control. A single microprocessor circuit can be used for a number of different applications simply by reprogramming it, unlike the other three types whose function can't be changed without changing their circuitry.

Microprocessors are built into many pieces of equipment and given suitable software for the job in hand. Of course, even in this sort of application, the microprocessor can't be used on its own, it has to have all the other chips which virtually turn it into a microcomputer before it can be used to do anything. So every owner of a personal computer has the heart of a potentially versatile control system at their fingertips and now there is a range of special interfaces and construction kits available for the hobbyist—anybody can control the real world with their micro.

The hobbyist has a terrific advantage over the professional system designers—plenty of time, the most valuable asset. There are lots of commercial examples of computer based control systems and for each one a whole series of new applications has yet to be tried.

POSSIBILITIES

Although your computer has the capability to direct a sophisticated control system, it cannot do this without the addition of several other parts.

A typical application requires an input and an output, both of which are outside the terms of the normal working of the computer—in which input is via the keyboard and output is to the screen or loudspeaker. A typical mechanical control system, for example, may need to be triggered when a component has reached a particular temperature, in which case it moves the component itself. So the input is fed via a *sensor*, whose signal is processed by the computer, which then outputs a signal to an *actuator*.

You have already seen examples of a computer operating in just such a way, in the article on robots (pages 884 to 888). Kits like those described there are just one specialized example of a control system. This article looks at the general aspects of using your computer to control external devices. This breaks down into three areas—sensors, actuators and their connection to the computer. But first, let's look at the possibilities.

Four areas worth exploring are:

- Timing and sequence control.
- Regulatory control.
- Data reduction.
- People/machine interfaces.

TIMING AND SEQUENCE

Timing and sequence controls are the systems used in everyday things like washing machines, sewing machines and central heating controls. The computer built into these is much more compact and reliable than the mechanical systems it has replaced. It can cope with much more complex operating sequences, its timing control is more precise and covers a much wider range, and the timing and sequence control can be altered simply by changing the software.

Of course, you would not want to bury your computer in the bowels of a washing machine, but a popular and practical area of application might be in a simple, but slightly more intelligent than normal, burglar alarm. At its simplest a burglar alarm consists of a loop of wire connected to switches on all of the exterior doors and windows. When these are closed the circuit is complete and an electric current can flow. If the circuit is broken by opening a window or door then an alarm bell sounds. The trouble is that with so many mechanical switches the likelihood of a bad contact or failed switch causing a false alarm is high. This is where a computer can make a valuable contribution. If a second circuit is added to all the interior doors and both the circuits are connected to a micro the software can make a few checks before sounding the alarm. The logic goes like this:

'If a burglar breaks in, then shortly after the exterior circuit is broken an interior door is likely to be opened, breaking the interior circuit. In this case both circuits are broken in the correct sequence and so the alarm is activated. If, on the other hand, either the exterior or interior circuits are broken on their own then there is likely to be a fault rather than an intruder so don't sound the alarm but report a fault.'

With suitable communication systems which are already available, the micro could even ring either the police to report a break-in, or the maintenance people to report a fault.

A lot of people without burglar alarms use simple time-switches to turn a light on and off to deter burglars. But sometimes the regularity of the light signals an empty house. A micro can be used to control a number of lights and other things like televisions and radios in a complex pattern, different for every day over a period of up to a month, say, and sequence could even be triggered by the dawn and dusk.

Another prime candidate for micro control is the control of a model railway layout with the software checking on the interlocking of points and signals. A further possibility is to control a number of slide projectors. Two or more slide projectors can be linked so as to produce interesting effects on the screen. The micro could control the brightness of all the lamps, creating fades up or down the mixes from one slide to another. It could control all the slide positions and report the number of the slides in each projector—or even synchronize the changes of slides to music.

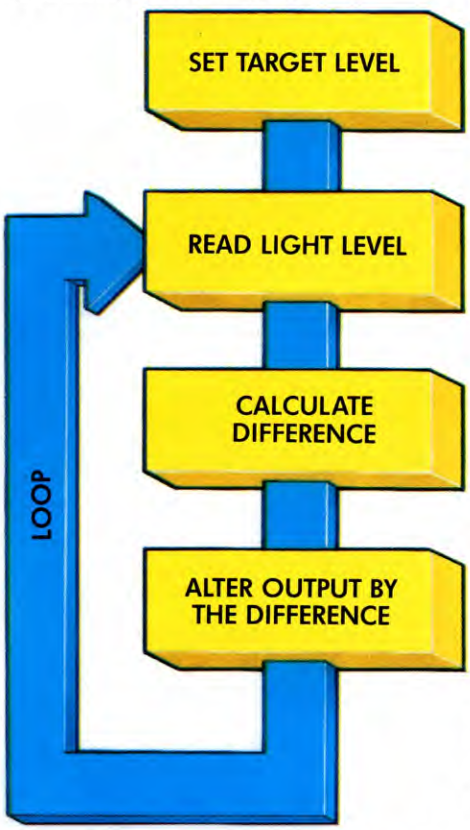
REGULATORY CONTROL

Regulatory control is used in any application where an action has to be taken based on comparing readings taken by the micro with desired readings prescribed in the software. In a central heating controller the micro can, for example, monitor the temperature of

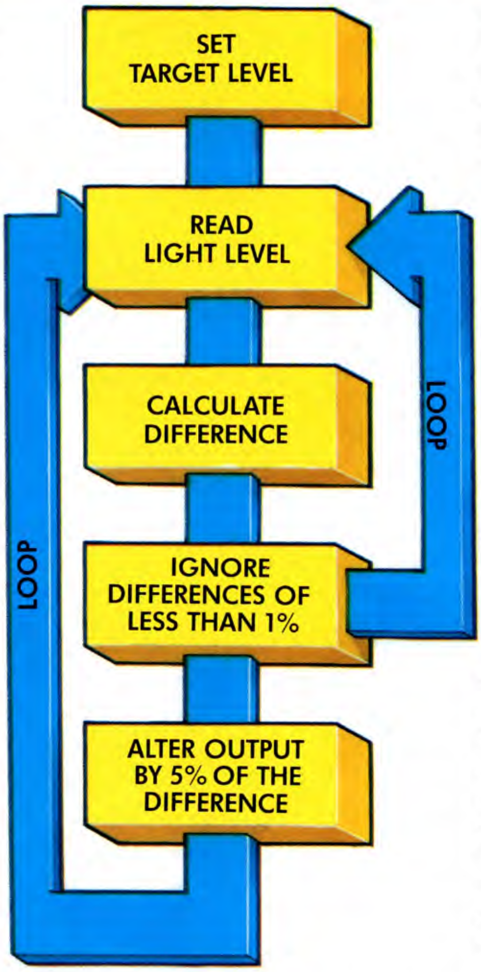
- MICROELECTRONIC CONTROLS
- POSSIBILITIES
- TIMING AND SEQUENCE
- REGULATORY CONTROL
- DATA REDUCTION
- DETECTION
- ACTUATION
- CONNECTIONS
- MECHANICAL SYSTEMS
- SOFTWARE

different rooms, the boiler and the hot water tank and operate water valves and control the boiler accordingly. The entire system is an example of a servo-mechanism control, relying on *closed-loop* control.

Looking at exactly what the term closed-loop means will also demonstrate why the micro is so versatile. Consider this problem—you want to regulate the intensity of an electric lamp so that the level of light is constant, taking into account changes in the daylight coming through a window. The computer is connected to an interface that controls the intensity of the lamp and a light sensor which supplies it (feeds back) information about the room's intensity. A closed-loop feedback system could then be set up in software something like this:



When this program is run, the brightness of the light will oscillate wildly, first getting brighter, then darker than the target level. This is caused by thermal lag in the lamp, which can't respond as quickly as the computer. When the voltage fed to the lamp is changed it takes a little while for its filament to cool down or heat up, but in the meantime the computer has taken another reading of the light level and, noting that there is still an error, it increases the correction. Eventually, when the lamp catches up, the correction is too big and the process reverses. To avoid this some more simple rules have to be added:



This time, the system will gradually adjust the light to the target level and keep it there. Rules can be added to cope with abrupt or transitory changes in external illumination.

DATA REDUCTION, INTERFACES

Data reduction and people/machine interfaces form an important part of any control system—but are also interesting on their own count. In many control applications, the micro receives information from lots of sources like sensors and converts it to a more intelligible or usable form—as in the light example above. The software checks for values falling within set limits, performs unit conversions, filters data and checks for logical inconsistencies.

Two examples of applications for data reduction techniques would be to produce a good colour analyser for amateur photographers or perhaps an automotive analyser for people wishing to tune their car engines—and there are already commercial machines to do this. In the latter example, suitable sensors connected to the computer could monitor the carbon dioxide level in the exhaust fumes, timing of the ignition, dwell angle of the points, condition of the plugs and adjustment of the tappets, ready for remedial action to be calculated by the software. Such a system could be interactive, with the micro giving instructions as to the direction and amount adjusting screws have to be turned and giving a warning when the correct adjustment is made.

People/machine interfaces are the areas in which the possibilities for development are enormous. Even at the simple level of displaying information this can range far beyond a numeric readout on a visual display unit or liquid crystal display, to full colour graphics. For example, a model railway control system could incorporate a module to let the user design the track layout on the screen and then carry out all of the control functions like changing points, selecting trains and controlling speed by using a light pen and pointing at the screen.

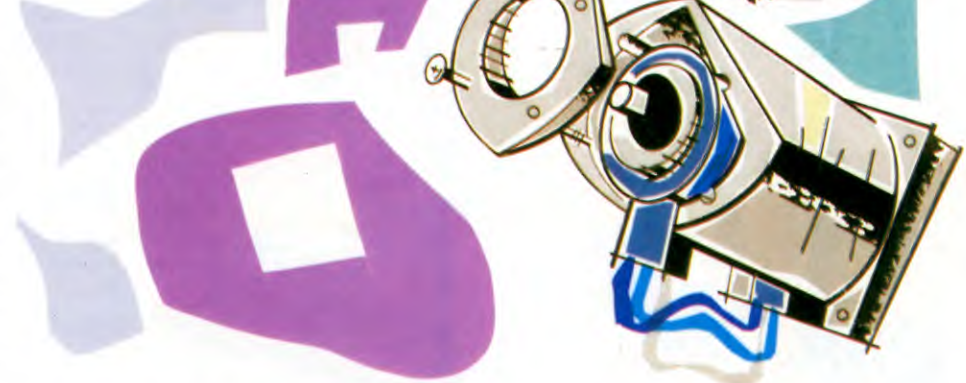
The micro has also opened up new possibilities for designing machines and control systems to help with all kinds of disabilities. Software control means that a single switch operation can start an extensive range of complex operations, such as controlling the selection of radio or television and then tuning either of them, adjusting the heating system, or turning room lights on and off and altering their intensity. Interesting developments include wordprocessing without having to use the whole keyboard. And the technology of non-keyboard input and output devices is coming to the point where voice synthesis for the dumb, or voice and pattern recognition for the blind, are almost a reality.

DETECTION

All four of the main areas of computer control are interrelated, and have a number of common elements. One of the most important is how computers get their information about the outside world. They do it rather like we do, through sensors.

All our five senses of touch, smell, hearing, taste and sight are available to the micro, although they are not yet at such a sophisticated level as the human equivalent. The micro also has the potential for some additional 'senses' like the ability to detect magnetic fields, atomic radiation, radio waves and other forms of electromagnetic radiation well outside the limits of our own eyes.

In order to detect any of these, the computer needs a suitable external sensor. Among these suitable are ultrasonic detectors, temperature probes, thermostats, gas sensors, liquid flow sensors, proximity detectors, optical sensors and switches of all different types

and sizes. Common sources include Radio Spares or Maplins, both of whose electronics component catalogues list hosts of interesting sensors.

Whatever the sensor senses, it produces one of two types of electrical signal, either a digital one coded as binary words (or, in the case of a switch, volts or no volts) or an analogue one in which the level of voltage represents the quantity of whatever is being sensed. The important difference between these is that analogue signals can't be handled directly by the computer but have to be processed by an analogue-to-digital (A to D) converter.

Some sensors have intelligence built into them. For example, an ordinary thermistor (a resistor whose electrical resistance varies with temperature) does not give a linear change of resistance with temperature change. For every degree rise in temperature, a different amount of change in resistance is produced. This has to be accounted for in any software using one of these devices. But there are also some intelligent temperature probes which work out the corrections and directly supply a linear voltage output—obviously at greater cost than a simple thermistor.

ACTUATION

To make things happen in the real world, the micro needs actuators—mechanical output devices. In most applications, the basic actuator is linked to a mechanical operating system. Unlike the enormous range of sensors, there are only five main types of actuator:

1. Pneumatic
2. Hydraulic
3. AC and DC electric motors
4. Solenoids
5. Stepper motors

The pneumatic actuator is rather like a steam engine. It consists of a cylinder containing a piston connected to a driving rod. Compressed air can be directed to either end of the cylinder by an electrically operated valve, to force the piston up or down. The hydraulic actuator works on the same principle but uses a liquid instead of air. It works with more precision and can be more powerful.

Ordinary AC and DC electric motors are widely available but are difficult to control with a micro. If a precision positioning has to be done, a sensor needs to be attached to the motor's shaft—usually a device to count the number of revolutions the shaft has made, indicating its speed and position. Some encoders, as they are called, split each revolution up into hundreds of bits allowing very accurate control of the motor.

Because of such complications, the solenoid and stepper motor are the two actuators most frequently used in micro systems. This is because they only need to be switched on and off, a simple thing for the computer to do. A solenoid is an electromagnet which moves an armature of some kind. When an electric current passes through it the coil becomes magnetized and moves the armature—which can then be used as a control system.

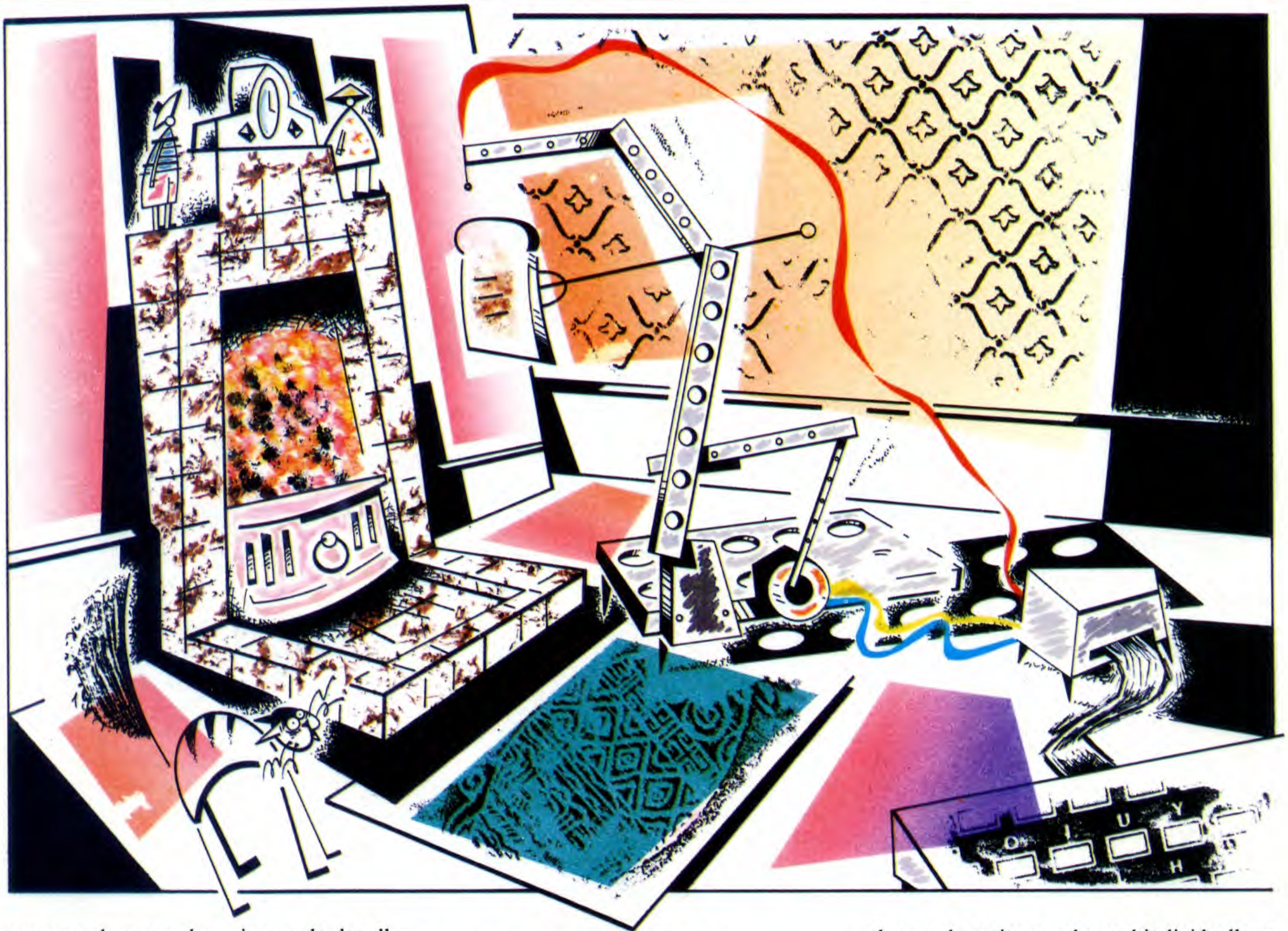
The stepper motor uses a number of solenoids to produce rotary movement like that of an ordinary electric motor. But although steppers are easy to use with a micro they do have the disadvantage of not being very powerful by comparison to similar-sized AC and DC motors.

In order to understand how stepper motors work, it is easiest to think of a four-pole stepper motor with four electromagnets arranged at right angles surrounding the rotor, an iron armature with two pole pieces which is free to revolve on a shaft. If one of the magnets is switched on, the rotor will be attracted to it and revolve to face it. If that one is switched off and a different one switched on, then the rotor will turn to face this one. If two adjacent coils are switched on, they will attract the rotor equally and it will move to a point mid-way between them.

Now, if the micro switches each of the coils on in sequence, the rotor will revolve—the faster the switching, the faster the rotation. With the example of four coils, there are eight divisions to one full rotation, the four actual positions of the coils and the four positions between each pair. Real stepper motors use a great many coils, about two hundred in fact, so the rotation is very smooth and the rotor can be accurately positioned without the need for feedback sensors indicating its position.

CONNECTIONS

Most of the sensors and actuators supply or demand between five and twelve volts at quite high currents. They can't be connected directly to the input and output ports of a



computer because the micro only handles very small currents and would be damaged almost instantaneously. To solve this problem you need to use an interface and there are a number of commercial interfaces that enable the sensors and activators to be connected without even needing a soldering iron.

These interfaces offer high current relay outputs, switch inputs, 8-bit output ports, high speed analogue to digital and digital to analogue convertors. Although A to D convertors are built into some computers, extra ones are often needed because the on-board convertors do not have a very high sample rate. For example, the BBC micro has one that runs at 100 Hz and a lot of applications require much higher sample rates. D to A convertors produce continuously variable analogue outputs from the computer's digital signals, suitable for controlling the speed of DC motors or low-voltage lamps.

Controlling mains equipment is even possible with 'intelligent plugs' that are sent instructions through the mains itself. But you must *never* connect a micro to the mains without a proper interface.

MECHANICAL SYSTEMS

There are a number of answers to the problem of linking the actuator to a mechanical system—many of these make practical use of a range of toy model systems. Meccano and Technical Lego are both useful and Fischertechnik construction kits offer real engineering in miniature with all the components you could possibly want.

Obviously, the greater the flexibility of the kit, the better—because mechanical linkages almost always need to be individually designed for each application. Typically, you may need to gear the output of a motor up or down, or connect it to a system of levers and cranks in order to produce the correct motion.

SOFTWARE

Writing software for control systems is no harder than writing for any other application. The commercial interfaces make the control of external devices quite straightforward. It is, of course, very important to analyse the control problem carefully and break the software down into small parts or sub-sections that can be written and tested individually as procedures or subroutines.

One of the difficulties of getting really interested in control projects is that it is inconvenient to have your micro permanently connected to your alarm or model railway layout. To overcome this you can even use a proper development system that connects to the micro and enables you to produce a dedicated single board computer specifically designed for the job. It is unlikely that such an application will need all of the facilities that make a complete home computer expensive.

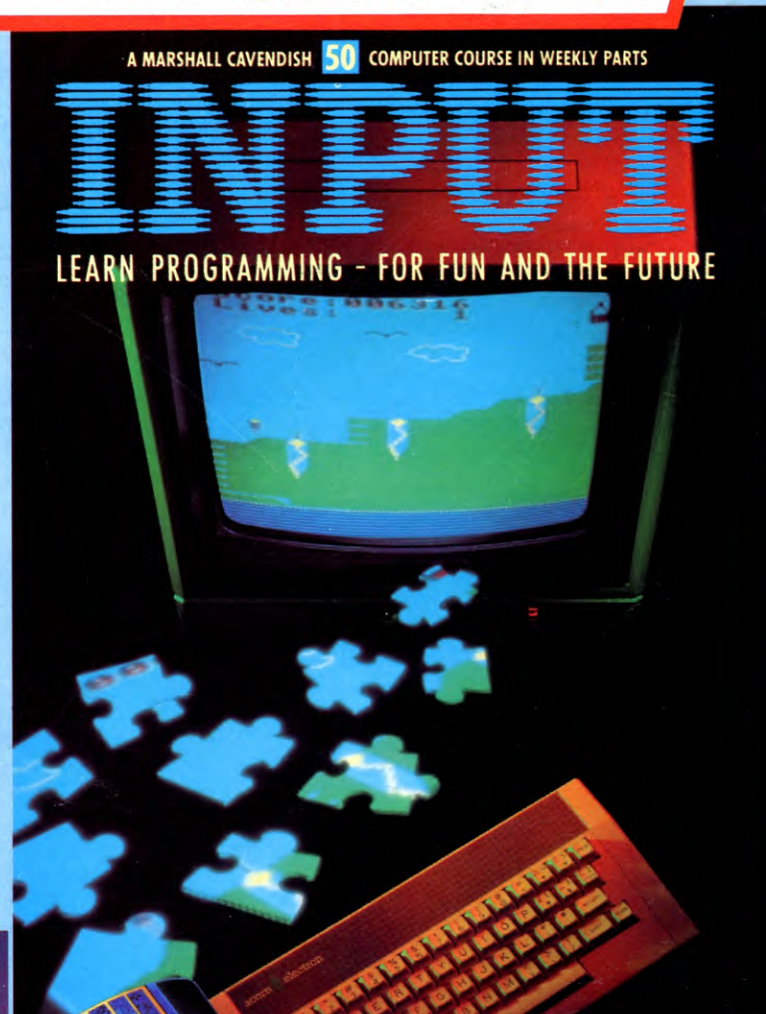
Development systems come with a range of hardware so that the computer can be tailored to the job. If the display is numeric then it doesn't need a CRT and graphics display controller, if the input requires only two buttons then it doesn't need an entire QWERTYUIOP keyboard. But if there are a number of analogue sensors then extra A to D convertors can be plugged in. Software supplied with these systems help you develop a program quickly and then enable it to be put into an EPROM (Erasable Programmable ROM) for permanent use.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 50...

- Many popular forms of puzzle can be solved by applying mathematical techniques for dealing with complicated equations. Find out how your micro can help with PUZZLE SOLVING, and look at some more serious applications for the techniques
- The dungeon door has swung shut and unknown terrors lie in wait. COMPLETE THE ADVENTURE game program and start to make your escape
- Discover how you can TUNE IN TO CODED MESSAGES and use your computer to catch a satellite
- Look beyond the limits of the home computer to the wide world of ALTERNATIVE LANGUAGES, and see what they might have to offer the micro user of the future
- Let your computer take control by finding out how to CONTROL A STEPPER MOTOR
- Plus a HEX DUMP and guide to CHECKING CLIFFHANGER



ASK YOUR NEWSAGENT FOR INPUT