Hi - Speed Windows

A series for beginners By Richard Sargent

The way in which computers organise their screen layout is by no means simple, and no standard screen layout or operating technique has developed during the short evolution of the home micro. The Amstrad screen is, of necessity, complicated; the computer has to be capable of producing an 80-column screen for business and programming, a multi-colour screen for games and a general purpose screen for everyday use. The computer has three screen modes to help it with these tasks, and there are some things that can be done in one mode but not in another. The reason why the screen display on home micros is the victim of compromise is quite simple - there is insufficient memory (and often insufficient processing speed) to produce the ideal screen which would have high-resolution graphics combined with 256 shades of colour and fast text scrolling into the bargain.

However, if you understand the peculiarities of your micro's screen, you will be able to use it to the full. This and subsequent articles will show you how to manipulate the Amstrad screen in Basic and, more importantly, will demonstrate machine-code routines which will develop screen displays which Basic would be too slow to achieve.

The Basic Screen:

Mode 1

The Amstrad screen occupies &4000 or 16K of memory, but is regarded by Basic as a piece of graph paper divided into tiny squares. There are 640 of these along the horizontal or X axis and 400 along the vertical or Y axis. On some computers these squares represent the actual screen pixels. This area is the graphics screen, and its origin is 0.0 at the bottom left-hand corner of the screen. This graph paper may also be ruled off into larger squares, the character blocks, of which there are 40 along the horizontal axis, or columns, and 25 along the vertical axis, or rows. The origin of this text screen is the top left-hand corner of the screen and it is 1,1. On the text screen, coordinates 0,0 are illegal.

Windows

Although graphics and text share the same physical RAM memory and the same screen (or graph-paper), different ways of labelling that graph-paper are possible and can exist simultaneously. For the text screen, these labelling systems

are known as windows, and you can create as many as eight of them at any one time. When you first switch on the computer or give MODE 1 command, WINDOW 0 is automatically created to the dimensions 1,40,1,25, which is the full dimension of the screen 1 to 40 columns wide and 1 to 25 rows deep. The computer manuals (Ch.5, p.10 for the 464 and Ch.9, p.26 for the 6128) explain WINDOWS and the examples given there should be studied carefully. The main point to be aware of is that data can be directed to a specific window and that a window can be scrolled, cleared and used to print INPUT prompts just as though it were a mini-screen in its own right.

The window has its own coordinate system, so that the top-left of a window is always accessible by LOCATE# n,1,1, no matter where the window is positioned on the screen. The windows are numbered 0 to 7 and the default size for them all is 1,40,1,25. Since this covers the entire screen, the windows are not particularly noticeable until they are cut down in size. Changing the dimensions of a window is achieved simply by issuing a new set of coordinates by telling the computer whereabouts (with reference to the whole screen) the new edges of the window are to be.

Thus the command WINDOW# 7,38,40,23,25 creates a small window in the bottom right-hand corner of the screen and thereafter LOCATE# 7,1,1 and LOCATE 38,23 both point to precisely the same physical point on the screen. LOCATE# 0,28,23: if the window you want is 0, the 0 may be omitted. Overlapping windows are permitted. Program listing and the 'Ready' prompt appear in WINDOW 0 so it is inadvisable to set it to small dimensions. Consult one of the manuals which advises that before you start experimenting you should set a spare key (such as 0 on the numeric pad) in the following manner:

```
KEY 128,"MODE 1:INK 0,1:INK 1,24:PAPER 0:PEN 1:CLS"+CHR$(13)
```

If you create small windows or strange colour combinations you will find it difficult to read (and therefore check) commands that you give the computer. By pressing key-pad 0, or f0 on the 6128, normality will be restored.

Printing messages, prompt, and calculation results onto screen windows rather than onto the screen as a whole is a sensible way to use a computer and the visual appearance of any program is likely to be enhanced by the use of just a few windows. Technically, windows are important because they stabilise the screen memory addresses and this is of special importance when it comes to a machine code programming because it makes the task of writing code that much more simple. A window can set aside a portion of screen which Basic does not use and so prepare the way for a machine code routine to handle that part of the screen memory directly.

The windows share the screen with each other, but they also share the screen with the graph paper. It is essential to understand how the graphic's screen windows interact, but now that windows are no longer a mystery, a short program can illustrate the workings of the graphic screen.

Type in Listing 1 exactly as it is shown, with the REM keywords 'protecting' certain instructions. With the listing in the computer (and saved), type MODE 1 directly, followed by RUN. A triangle is partly drawn and the screen coordinates of the various points of the triangle are printed, causing both the text and the graphics to scroll upwards. Two more RUNs make the screen very messy. Now LIST and retype line 110 without its REM keyword. Type MODE 1 and RUN directly and observe the result. The text scrolls but the triangle stays where it is. Further RUNs do not affect the position of the triangle, because the WINDOW statement of line 110, now active, has confined the text to the left-hand side of the screen. CLS, acting on the window, will clear the text but not the graphics. CLG will clear the graphics screen, which covers the whole screen so the triangle and the text will both disappear.

Listing 1

```
100 REM LISTING 1 (P1L1)
110 REM WINDOW 1,20,1,25
120 LOCATE 1,24
130 ORIGIN 400,200
140 TAG:PRINT "0,0";:TAGOFF
150 PLOT 0,0
160 PRINT XPOS;YPOS
170 DRAW 100,100
180 PRINT XPOS;YPOS
190 DRAWR 0,-200
200 PRINT XPOS;YPOS
210 DRAW -50,-50
220 PRINT XPOS;YPOS
230 STOP
```

If you want the graphics screen to be a different colour from the standard blue background, then CLG 3 will set the graphics paper to ink number three, which is normally red. In fact, you can think of the graphics screen as yet another window. It has no channel number attached to it, and it is permanently set to full-screen size. How then, do you push the graphics into just one part of the screen? Normally, the base coordinates of the graphics screen are X=0, Y=0, fixed to the bottom pixel on the left-hand side of the screen. The triangle on the right of the screen is occupying pixel positions in the X=450 Y=250 area of the screen, yet the printed reports given by XPOS and YPOS are much lower figures. What has happened is that the program has set a new base for the graphics screen. Line 130 really means 'Take postion 400,200 of the whole screen and make it position 0,0 of a new graphic screen'. All subsequent PLOTs, MOVEs and DRAWs take place with reference to the new origin. With coordinates 0,0 in their new position, it is possible to plot using minus X and minus Y values, as the program illustrates, but positive values keep the graphics over in the right-hand side of the screen.

Window Design

Listing 2 is a Basic program which demonstrates the way in which windows can be used to enhance a screen display. Figure 1 is the screen dump of the program which illustrates the GEM-like qualities which can be achieved even on the MODE 1 Amstrad screen. For the record, WINDOW 0 is the whole screen, WINDOWS 1 to 4 are the four shadow-outline boxes, WINDOW 6 is the 'Hello World' box, WINDOW 5 is, by design, invisible and carries the INPUT prompts. The remaining window is 7 which carries the headings of the 'pull-down' menus. The program is a demonstration only, but is nevertheless instructive. The information for the eight windows is carried in the DATA statements of lines 1380-1450 and is later transferred to an array and also the variable W,E,N and S. The relationship for window 0 is shown here:

```
1380 DATA
2 : z(1) : Paper is ink 2
1 : z(2) : Pen is ink 1
1 : z(3) : W : west edge col.1
40 : z(4) : E : east edge col.40
1 : z(5) : N : north edge row 1
25: z(6) : S : south edge row 25
```

The border colour, ink colours and heights of the window at the top of the screen (which carries the names of the pull-down menus) all find their way into the Z(n) array which carries a total of 54 pieces of information. If you don't want to type in the whole program, you might like to try the key subroutine at lines 1120 - 1340 and 1520 - 1630. The former routine places the window (and its shadow) on the screen, while the latter forms the special symbols required for the shadowing, and the symbols needed for the large arrow ICON and the hand ICON. The few lines of Basic needed to run the two subroutines are as follows:

```
10 GOSUB 1520
12 INK 0,1:INK 1,24:PAPER 0:PEN 1
14 MODE 1:DIM Z(55):Z(54)=0:P=0
15 LOCATE 1,1:PRINT SPACE$ (24)
16 PRINT SPACE$(24):LOCATE 1,1
17 INPUT #W,E,N,S,-;W,E,N,S
20 IF E=40 OR S=25 THEN GOTO 15
22 GOSUB 1130:GOTO 15
```

Listing 2

```
100 REM LISTING 2
110 REM WINDOWING DEMONSTRATION
112 REM P1L2
120 GOSUB 390:REM INITIALISE EVERYTHING
130 :
140 REM DEMO PROGRAM STARTS HERE
150 :
160 M(2)=Z(44):M(3)=Z(44):M(4)=Z(44):M(1
    )=2:GOSUB 650:B=1:GOSUB 720
170 INPUT #5," WHAT NOW";X$
180 IF X$="R" OR X$="r" THEN GOTO 250
190 IF X$="B" OR X$="b" THEN WINDOW 1,40
    ,1,25:INK 0,1:PAPER 0:INK 1,24:PEN 1
    :CLS:STOP
200 IF X$="C" THEN ICOPY,2:GOTO 160
210 B=4:M(B)=2:GOSUB 650:GOSUB 980
220 INPUT #5," PRESS ENTER";X$
230 GOTO 160
240 :
250 M(1)=Z(44):M(2)=Z(44):M(3)=Z(44):M(4
    )=Z(44)
260 B=3:M(3)=2:GOSUB 650:GOSUB 900
270 INPUT #5," WHICH MENU";B
280 M(1)=Z(44):M(2)=Z(44):M(3)=Z(44):M(4
    )=Z(44)
290 M(B)=2:GOSUB 650
300 IF B=1 THEN GOTO 160
310 IF B=2 THEN GOSUB 820
320 IF B=3 THEN GOSUB 900
330 IF B=4 THEN GOSUB 980:GOTO 220
340 IF B=6 THEN GOSUB 1050
350 GOTO 160
360 :
370 REM END OF DEMO PROGRAM
380 :
390 MODE 1
400 REM READ SELECTED WINDOW DATA
410 RESTORE 1380
420 DIM Z(55)
430 FOR N=1 TO 54
440 READ S:Z(N)=S:NEXT
450 INK 0,Z(50):INK 1,Z(51):INK 2,Z(52):
    INK 3,Z(53)
460 BORDER Z(49)
470 A$="     MENU "
480 B$=" 2:MENU2  "
490 C$=" 3:MENU3  "
500 D$=" 4:MENU4  "
510 GOSUB 1530
520 WINDOW Z(3),Z(4),Z(5),Z(6):PEN Z(2):
    PAPER Z(1):CLS
530 WINDOW #6,Z(39),Z(40),Z(41),Z(42)
540 WINDOW #5,Z(33),Z(34),Z(35),Z(36)
550 PAPER #5,Z(31):PEN #5,Z(32):CLS #5
560 WINDOW #7,Z(45),Z(46),Z(47),Z(48)
570 M(1)=Z(44):M(2)=Z(44):M(3)=Z(44):M(4
    )=Z(44):GOSUB 650
580 B=4:GOSUB 980:B=3:GOSUB 900:B=2:GOSU
    B 820:B=6:GOSUB 1050
590 RETURN
600 REM END OF INITIALISATION
610 :
620 :
630 REM WINDOWING SUBROUTINES
640 REM SET TOP WINDOW COLOUR NORMAL OR
    HIGHLIGHT
650 PAPER #7,Z(43):PEN #7,M(1):PRINT #7,
    A$;
660 PEN #7,M(2):PRINT #7,B$;
670 PEN #7,M(3):PRINT #7,C$;
680 PEN #7,M(4):PRINT #7,D$;
690 RETURN
700 :
710 REM DRAW WINDOW 1
720 W=Z(9):E=Z(10):N=Z(11):S=Z(12):P=Z(7
    ):GOSUB 1130
730 REM PRINTED PROMPTS
740 LOCATE #1,2,2:PRINT #1,"L LOAD":PRIN
    T #1
750 PRINT #1," S SAVE":PRINT #1
760 PRINT #1," B BASIC":PRINT #1
770 PRINT #1," R RUN":PRINT #1
780 PRINT #1," Press ";CHR$(250);CHR$(25
    1):PRINT #1:PRINT #1," L,S,B,or R "
790 RETURN
800 :
810 REM DRAW WINDOW 2
820 W=Z(15):E=Z(16):N=Z(17):S=Z(18):P=Z(
    13):GOSUB 1130
830 REM SCROLLING PATTERNS
840 FOR C=1 TO 8
850 FOR A=211 TO 255
860 PRINT #2,CHR$(A);:NEXT:NEXT
870 RETURN
880 :
```

```
890 REM DRAW WINDOW 3
900 W=Z(21):E=Z(22):N=Z(23):S=Z(24):P=Z(
    19):GOSUB 1130
910 REM PRINTED PROMPTS
920 PRINT #3:PRINT #3," PRESS ";CHR$(244
    );CHR$(245)
930 PRINT #3,"          ";CHR$(247)
940 PRINT #3," A NUMBER    1,2,3,4,6"
950 RETURN
960 :
970 REM DRAW WINDOW 4
980 W=Z(27):E=Z(28):N=Z(29):S=Z(30):P=Z(
    25):GOSUB 1130
990 REM PRINTED PROMPTS
1000 PRINT #4:PRINT #4," SORRY, BUT  ONL
     Y <RUN>  IS WORKING  ON THIS     DE
     MO":PRINT #4
1010 PRINT #4," PRESS":PRINT #4," <ENTER
     >"
1020 RETURN
1030 :
1040 REM DRAW LARGE COLOUR WINDOW (#6)
1050 W=Z(39):E=Z(40):N=Z(41):S=Z(42):P=Z
     (37):GOSUB 1130
1060 REM FILL WITH TEXT
1070 FOR A=1 TO 57
1080 PRINT #6,"HELLOWORLD";:NEXT A
1090 RETURN
1100 :
1110 REM WINDOW SHADOWING ROUTINE
1120 REM FIRST CREATE WINDOW SIZE,PAPER
     INK &PEN INK
1130 WINDOW #B,W,E,N,S:PEN #B,0:PAPER #B
     ,P:CLS #B
1140 REM NOW CREATE BOTTOM & RIGHT SHADO
     WS
1150 LOCATE W,S+1:PRINT CHR$(248);
1160 FOR P=W+1 TO E:PRINT CHR$(131);:NEX
     T
1170 PRINT CHR$(129);
1180 LOCATE E+1,N:PRINT CHR$(249);
1190 FOR P=N+1 TO S
1200 LOCATE E+1,P:PRINT CHR$(133);:NEXT
1210 REM CREATE LEFT SIDE UNLESS AGAINST
      LEFT OF SCREEN
1220 IF W<2 THEN GOTO 1270
1230 FOR P=N TO S
1240 LOCATE W-1,P:PRINT CHR$(254);:NEXT
1250 LOCATE W-1,S+1:PRINT " ";
1260 REM CREATE TOP SIDE UNLESS AGAINST
     TOP OF SCREEN
1270 IF N<2+Z(54) THEN RETURN
1280 LOCATE W,N-1
1290 FOR P=W TO E:PRINT CHR$(253);:NEXT
1300 PRINT " ";
1310 REM FINISH OFF
1320 IF W<2 THEN RETURN
1330 LOCATE W-1,N-1:PRINT CHR$(252);
1340 RETURN
1350 :
1360 REM WINDOW & COLOUR DATA
1370 REM PAPER/PEN/W/E/N/S
1380 DATA 2,1,1,40,1,25
1390 DATA 2,0,2,13,3,15
1400 DATA 2,0,11,22,5,17
1410 DATA 2,0,19,30,7,19
1420 DATA 2,0,28,39,9,21
1430 DATA 2,1,1,40,24,25
1440 DATA 3,0,6,35,4,22
1450 DATA 0,1,1,40,1,1
1460 REM BORDER COLOUR
1470 DATA 13
1480 REM THE 4 INK COLOURS
1490 DATA 0,13,25,22
1500 REM HEIGHT OF TOP BOX
1510 DATA 1
1520 REM SYMBOL DATA
1525 SYMBOL AFTER 244
1530 SYMBOL 244,0,1,127,127,0,3,0,0
1540 SYMBOL 245,0,255,255,255,15,255,15,
     255
1550 SYMBOL 247,15,127,0,0,0,0,0,0
1560 SYMBOL 248,&FF,&7F,&3F,&1F,0,0,0,0
1570 SYMBOL 249,&80,&C0,&E0,&F0,&F0,&F0,
     &F0,&F0
1580 SYMBOL 250,24,56,127,255,127,56,24,
     0
1590 SYMBOL 251,15,31,254,252,254,31,15,
     0
1600 SYMBOL 252,0,0,0,0,0,0,0,1
1610 SYMBOL 253,0,0,0,0,0,0,0,&FF
1620 SYMBOL 254,1,1,1,1,1,1,1,1
1630 RETURN
```

The Real Screen

The real screen is a block of memory residing in one-quarter of the Amstrad's 64K of RAM, and it is actually known as the screen memory. When the Z80 CPU isn't looking (or at least not looking at this particular block of memory), another chip called the VDU controller sends the information in the screen memory to yet another large chip called a Gate Array (where it is mixed with colour-palette information) and hence out to the green or colour monitor. A computer needs one bit of computer memory to store one pixel of VDU image, and more than one bit may be necessary to encode the colour of that pixel.

The first consideration must be to find out how many pixels the Amstrad is capable of showing, and then to see how they correspond to the screen memory. Just as it was easier to deal with Basic windows in MODE 1, so it is easier to explain the real screen with reference to MODE 2, the high resolution mode. In this mode, the screen can display a maximum of 80*25 characters. Each character is made up from 64 pixels so there are 80*25*64 or 128,000 pixels on the screen. Don't at this stage, worry that the graph-paper of the graphics screen appears to have twice as many pixels (640*400 = 256,000): it hasn't. From here on things get a little complicated: 16K of screen memory represents 16*1024*8 bits of information which is 131,076. The spare capacity of approximately 3076 (a few hundred bytes) is in fact not used at all and can be thought of as being in limbo just beyond the edges of the screen.

In MODE 2, a plotted point is represented by a memory bit set to 1 and an unplotted point a memory bit set to 0. A character is screened as an 8*8 matrix of points containing an assortment of 1s and 0s. The 1s take on the pen ink colour and the 0s take on the paper ink colour. Previous commands, stored in the gate array determine which two colours are assigned to the INKS, and only two different colours can be on screen at the same time. Identical colours cause all points

to be invisible - an interesting situation which frequently occurs when MODE 2 is entered after certain colour combinations have been set in MODE 0 or MODE 1.

The screen memory runs from &C000 to &FFFF, and it can be POKED from Basic. It can also be PEEKED from Basic, despite the fact that a ROM also occupies the same part of the computer's memory. Locomotive Basic is clever enough to assume that if you ask it to read an address above &BFFF, it's the screen memory RAM contents you're interested in and not the ROM contents.

Type in direct mode, CLS:POKE &C000,&FF and a small dash will appear in the top left-hand corner of the screen. The first blow to sanity is the fact that the screen memory moves about. It's always at &C000 to &FFFF, but &C000 is not always the first byte of the screen. Whenever a whole screen scrolls, a fast 'hard scroll' routine is used and screen addresses change, which is disconcerting to say the least. POKE &C000,&FF will probably place the dash in the middle of the screen. However, when windows which are less than full-screen size scroll, a slower 'soft scroll' routine is used and the screen memory addresses are not scrambled.

The third program this month allows you to poke the screen memory in various safe ways to see how the addresses are arranged, how a byte appears when it is placed directly into the screen memory, and how the same bytes created different effects when the screen mode is changed. Windowing is used to form a menu-driven program and to inhibit the hard-scrolls.

Listing 3

```
49 REM LISTING THREE
50 REM SCREEN INVESTIGATOR
52 REM P1L3
100 INK 0,1:PAPER 0:INK 1,24:INK 2,20:IN
    K 3,6:PEN 1:CLS:BORDER 9
110 INPUT "MODE ";M:MODE M
120 PLOT 36,376:DRAWR 508,0:DRAWR 0,-64:
    DRAWR -508,0:DRAWR 0,64
130 PLOT 552,376:DRAWR 80,0:DRAWR 0,-64:
    DRAWR -80,0:DRAWR 0,64
140 PLOT 36,296:DRAWR 596,0:DRAWR 0,-272
    :DRAWR -596,0:DRAWR 0,272
150 IF M=0 THEN L=3:R=19:RR=16
160 IF M=1 THEN L=4:R=39:RR=33
170 IF M=2 THEN L=6:R=77:RR=67
180 WINDOW L,R,8,23
190 A=&C000+&138
200 WINDOW #1,L,RR,3,5
210 REM ----------------------
220 CLS:PRINT " MENU"
230 PRINT:PRINT "1. BOX - AUTO"
240 PRINT "2. BOX - MANUAL"
250 PRINT "3. WHOLE SCREEN"
260 PRINT "4. NEW MODE"
270 PRINT "5. CLS"
280 PRINT "6. BASIC":PRINT
290 INPUT RP:IF RP<1 OR RP>6 THEN GOTO 2
    20
300 CLS:CLS #1:IF RP=1 THEN GOTO 370
310 IF RP=2 THEN GOTO 460
320 IF RP=3 THEN GOTO 510
330 IF RP=4 THEN GOTO 100
340 IF RP=5 THEN MODE M:GOTO 120
350 IF RP=6 THEN MODE 1:STOP
360 REM ----------------------
370 LOCATE #1,1,3:PRINT #1,"AUTO";
380 INPUT "STEP ";S:IF S=0 THEN GOTO 380
390 IF S>99 THEN GOTO 380
400 FOR H=0 TO &FF STEP S
410 GOSUB 630
420 IF S>2 THEN INPUT X$
430 NEXT H
440 GOTO 220
450 REM ----------------------
460 LOCATE #1,1,3:PRINT #1,"&X INPUT";
470 INPUT "&X ";H
480 IF H>&FF THEN GOTO 220
490 GOSUB 630:GOTO 470
500 REM ----------------------
510 CLS #1:LOCATE #1,1,1:PRINT #1,"ADDR
    IS C000";
520 LOCATE #1,1,3:PRINT #1,"INC BY ";
530 CLS:INPUT "INCREMENT ";INC:PRINT #1,
    HEX$(INC,4);
540 INPUT "HOW MANY ";AM
550 AD=49152:NUM=0
560 POKE AD,&C0:LOCATE #1,9,1:PRINT #1,H
    EX$(AD,4);
570 NUM=NUM+1:IF NUM>AM THEN GOTO 600
580 AD=AD+INC:IF AD>49151 AND AD<65536 T
    HEN GOTO 560
590 PRINT "THE NEXT POKE WOULD BE OUT OF
     RANGE":GOTO 610
600 PRINT "ALL DONE":GOTO 610
610 PRINT "PRESS <ENTER>":INPUT X$:GOTO
    220
620 REM POKE S/R ------------------
630 PRINT H;" ";HEX$(H,2)
640 LOCATE #1,1,1:PRINT #1,H;" ";HEX$(H,
    2);"H  ";BIN$(H,8)
650 POKE A,H
660 X=A+4
670 FOR Y=1 TO 8
680 POKE X,H
690 X=X+&800:NEXT
700 RETURN
```

When the program is started, you are invited to choose a screen mode and you should choose 2 for the first run since it gives a very clear demonstration, especially on green monitors. The display menu is printed in one of them. Options 4, 5 and 6 merely change screen modes, clear the screen and exit to Basic, but options 1, 2 and 3 allow the screen to be investigated.

1.BOX - AUTO:

A number, starting with zero, is incremented in steps of your choice until 255 is reached, whereupon the routine stops. Each number is POKED directly into the screen memory so you can observe the effect it has on the screen display. When you choose this option, you are first invited to enter a stepping value. Type 32 then ‹ENTER›, ‹ENTER›. The program will increment a byte from 0 to 255 in steps of 32 and POKE the value onto the screen. The two 'enters' cause a window above the menu to display '0 00H 00000000' and then '32 20 H00100000'. In a third, square, window the effect of the poke can be seen both as a single byte and as a stack of 8 contiguous vertical bytes of the same value. Pressing ‹ENTER› continues the process. Being mode 2, there is of course a direct comparison between the binary representation of the byte and the bar which is poked into the square window.

If the step value entered is 1, then all 256 bytes will be poked to the screen. This is done by the program so you need not press ‹ENTER› 256 times. It takes 50 seconds to go through all 256 combinations.

2. BOX - MANUAL:

This option invites you to enter a number of your choice which is then POKED on the screen in the same way as in option one. For example, entering any one of the number types 255, &FF or &X11111111 produces a bar (all 8 pixels illuminated) in the square window. Entering any number greater than 255 will take you back to the menu.

3. WHOLE SCREEN:

This option POKES values to various addresses all over the screen and does not confine its activities to the square window. At the start of the routine 'ADDR IS C000' is written in the top window and an input prompt 'INCREMENT?' is placed in the main window. Sixteen is a suitable increment to start with so type 16 ‹ENTER›. The prompt 'HOW MANY?' appears next because the program is about to make a series of pokes to address &C000 and every sixteenth address after &C000 and it wants to know the total number of pokes it should make before stopping. 100 is sufficient and 100 ‹ENTER› will start the process. Error trapping prevents any poke going to non-screen addresses, so you can play around with this option and try to work out how the screen memory addresses are laid out. When the screen gets too messy, use option 5 and start again.

In these three routines you will find that the results obtained in screen modes 0 and 1 are different from those obtained in MODE 2, but that by staying in MODE 2 it is possible to work out the address-structure, or memory-map, of the screen display.

Next month the screen memory map will be revealed (it is misprinted in the CPC464 Firmware Manual so don't cheat and look it up). Modes and colour will be discussed and a start will be made on some short machine-code routines.