CORTEX BASIC

The BASIC used on the Cortex contains many statements which will be unfamiliar to readers who are used to Microsoft. Beginning this month, we'll be taking a brief look at the keywords and their functions. This month: graphics.

BASIC on the Cortex is a derivative of Texas Instruments' Power BASIC, with some additional keywords necessary to make use of some of the features of the Cortex. Some of these involve the graphics commands which we are making the basis of this article.

The video display processor in the Cortex is, in fact, capable of four display modes, but only two of these are implemented by the BASIC. The two types of display are accessed by the TEXT and GRAPH commands.

TEXT

The TEXT mode provides 24 40-character rows in two colours and is intended to maximise the capacity of the TV screen to display alphanumeric characters. A diagram of the screen in this mode is shown in Fig. 1. The character cell number is equal to the horizontal position (0 to 39) plus 40 times the vertical position (0 to 23). The only items that may be displayed in this mode are the alphanumerical character set, which are defined on a six by eight grid of pixels (six pixels across, eight pixels down). There are a possible 256 patterns that can be displayed and on power-up these are defined by the BASIC in the EPROM. Examining the characters shows that the first 32 are symbolic representations of the corresponding ASCII control codes, the next 64 are standard upper case ASCII, the next 32 are 'small capitals' rather than lower case ASCII (these small capitals are used in the error messages), and the remaining 128 characters are not assigned any meaningful pattern.

However, any or all of these character patterns may be changed by using the CHAR command. This has the format

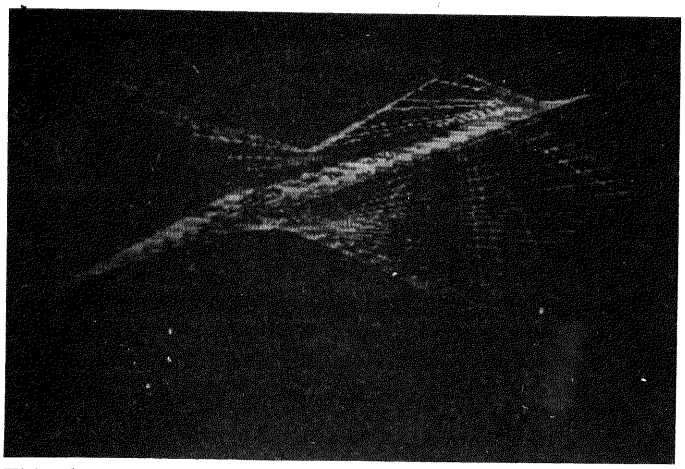
CHAR arg1, arg2, arg3, arg4

where arg1 is the number of the character to be changed (0 to 255), and arg2, arg3 and arg4 define the new bit pattern for the character. These arguments are 16-bit numbers and define the character row by row from top to bottom, with a 1 producing the foreground colour and a 0 producing the background colour. For example, if you type CHAR32,20,20,20: TEXT then your screenful of 'spaces' suddenly develops freckles! (Incidentally, executing TEXT clears the screen and homes the cursor).

Another fun thing to do (although completely pointless!) is to scramble the character set using random numbers. Try

```
10 FOR I = 0 TO 255
20 CHAR I, RND*255,RND*255,RND*255
30 NEXT I
40 TEXT
```

and then type in LIST after running the program. Not so easy to read, eh? To get back the original patterns, just execute a reset with the switch on the rear and the Cortex will re-load the character table from the EPROM.



This photograph shows lines plotted in GRAPH mode. The pixel resolution is 256 by 192 but the limitation on colour means some areas (noticeably to the left of the shape) get 'blocked in' colour.

GRAPH

In GRAPH mode the screen dimensions change to that shown in Fig. 2, a grid of cells 32 by 24. The character cell number is given by the horizontal position (0 to 31) plus 32 times the vertical position (0 to 23). In addition, the screen may also be considered to consist of individual pixels (256 across by 192 down). Thus, each character cell in this mode is eight by eight pixels in size, offering a better pattern resolution than the six by eight pixels of TEXT mode.

As in TEXT mode, executing the GRAPH statement will clear the screen and home the cursor, which in graph mode is an invisible pixel cursor. An alternative method of clearing the screen is to use the program statement

PRINT "<0C>"

which has the advantage of wiping off any text messages or plotted lines but leaving sprites unaffected. The reason why this statement works will be covered in a future article: suffice it to say that the statement executes the ASCII control code for Clear Screen.

Pixels are numbered from 0 to 255 horizontally and from 0 to 191 vertically. The origin is at the top left-hand corner of the screen as shown in Fig. 2.

PLOT AND UNPLOT

The PLOT statement is used to turn on individual pixels on the text/graphic plane. The basic format is

PLOT arg1,arg2 TO arg3,arg4

By leaving out various parts of the statement, different actions can be performed. If the entire statement is executed, then a line is drawn in the current foreground colour from the pixel co-ordinates given by arg1 and arg2 (arg1 = horizontal, arg2 = vertical) to the pixel co-ordinates given by arg3, arg4. The in-

0	1	2	3	4	5		37	38	39
40	41	42	43	44	45		77	78	79
80	81	82	83	84	85		117	118	119
880	881	882	883	884	885		917	918	919
920	921	922	923	924	925		957	958	959

Fig. 1 Screen position map for the Cortex in TEXT mode. Here the screen is divided into a 40 by 24 grid which can only display the character set — sprites are not possible in this mode.

0,0

									255,0
0	1	2	3	4	5		29	30	31
32	33	34	35	36	37		61	62	63
64	65	66	67	68	69		93	94	95
704	705	706	707	708	709		733	734	735
736	737	738	739	740	741		765	766	767

0,191 255,191

Fig. 2 The screen position map for the Cortex in GRAPH mode. The grid is now 32 by 24 squares and each square may contain members of the character set or the shape table. In addition, up to 32 sprites may be displayed using the shape table patterns, and individual pixels may be set or reset.

visible pixel cursor is left at arg3, arg4 (horizontal, vertical).

If arg1, arg2 are omitted, ie PLOT TO arg3,arg4, then a line is drawn from the current graphic cursor position to the co-ordinates given by arg3, arg4. If the TO arg3,arg4 part of the statement is omitted, ie PLOT arg1,arg2, then the single pixel specified by the co-ordinates arg1, arg2 is set to the current foreground colour. The UNPLOT statement has the same format and variants as the PLOT statement except that the line or pixel is removed instead of being plotted.

COLOUR, COL

Colours may be set up in TEXT or GRAPH mode by means of the COLOUR statement. The format for this is

COLOUR foreground colour, background colour

The two colour arguments can take the values 0 to 15, the corresponding colours being given in Table 1, Cortex Part 1, November 82 issue. If the foreground colour only is given, eg COLOUR 6, then the current background colour is used.

Two colours only are allowed in TEXT mode. Executing a COLOUR statement in a program or in immediate mode will recolour the entire display. By contrast, all 16 colours may be displayed at once in GRAPH mode, with the limitation that each horizontal line of eight pixels (ie one character cell width) can only have one foreground colour and one background colour. Try this program to see what this means:

```
10 COLOUR 4,7: GRAPH
20 COLOUR 1,13: PLOT 0,0 TO 255,191
```

The pixels in the text/graphic plane can be tested for their colour by reading the code into a variable using the COL function. The format is

var = COL arg1,arg2

where arg1, arg2 are the horizontal and vertical co-ordinates of the pixel to be tested. The variable var will now have a value equal to the colour code of the pixel.

SHAPE, SPRITE, MAG

The SHAPE statement is used to define one of 256 possible eight by eight pixel shape definitions. The format is

SHAPE arg1,arg2,arg3,arg4,arg5

where arg1 is the shape table entry to use (0 to 255), arg2 is the 16-bit integer pattern of the first and second row of the shape, arg3 gives the third and fourth rows, arg4 gives the fifth and sixth rows and arg5 gives the seventh and eighth rows. For arg2 to arg5 the most significant byte defines the first row and the least significant, the second row. For example, to define a solid block use SHAPE 2, –1, –1, –1, –1.

Once shapes have been defined they can be displayed on screen using the SPRITE command. Each sprite plane can hold one sprite, giving a maximum of 32 on screen at once, and if a sprite on a plane is rewritten into a new position the old one is automatically erased. The format for the statement is

SPRITE arg1,arg2,arg3,arg4,arg5

where arg1 is the sprite plane to hold the sprite (0 to 31), arg2 is the horizontal co-ordinate of the sprite's top left pixel, arg3 is the vertical co-ordinate of the sprite's top left pixel, arg4 is the shape number to use for the pattern (0 to 255) and arg5 is the sprite colour (0 to 15).

There are two limitations to the use of sprites. One is that only four sprites at a time may be displayed on a given horizontal line: an attempt to add a fifth will make the overlapping portion invisible. Try this program:

```
10 COLOUR 1,15: GRAPH: MAG 1,0
20 SHAPE 10, –1, –1, –1, –1
30 FOR T=1 TO 14
40 FOR I=1 TO 100
50 SPRITE T, I–T*2, I–T*2, 10,T
60 WAIT 1
70 NEXT I: NEXT T
```

The second limitation is that you can only use a sprite plane if all the ones above it have been used. Hence you must place your first sprite on plane 0, your second on plane 1 and so on. Of course, once a plane has been initialised in this way you can wipe it if necessary by setting its sprite to an all-zeros shape or setting its colour to transparent.

The MAG statement defines both the size and the definition of the sprites. The format is

MAG sprite magnification, sprite definition size

If the sprite magnification is 0 every bit in the shape definition used for the sprite will be displayed as one pixel. If the magnification is non-zero then each bit will be displayed as two pixels horizontally and vertically. If the definition size is zero then one shape table entry will be used to build the sprite. If it is non-zero then four entries will be used. These entries are joined in the following way to build a 16 by 16 point sprite: top left, shape n; bottom left, shape n+1; top right, shape n+2; bottom right, shape n+3, where n is the shape number given to the sprite statement. The shape table entries must start on a four entry boundary, so valid values of n are 0,4,8,12 etc.

Note that the SHAPE, SPRITE and MAG statements will only work if the Cortex is in GRAPH mode.