21 VDU codes

Introduction

The statement **VDU X** is equivalent to **PRINT CHR$(X);** and the statement **VDU X,Y,Z** is equivalent to **PRINT CHR$(X);CHR$(Y);CHR$(Z);**

However the **VDU** statement finds most common use when generating ASCII control codes and a detailed description of the effect of each control code is given in this chapter. The control codes are interpreted by part of the machine operating system called the VDU driver.

The VDU drivers interpret ail 32 ASCII control character codes. Many of the ASCII control codes are followed by a number of bytes. The number of bytes which follow depends on the function to be performed. The VDU code table summarises all the codes and gives the number of bytes which follow the ASCII control code.

Detailed description

4 This code causes text to be written at the text cursor, ie in the normal fashion. A MODE change selects **VDU 4**, normal operation.

5 This code causes text to be written where the graphics cursor is. The position of the text cursor is unaffected. Normally the text cursor is controlled with statements such as

```
PRINT TAB(5,10)
```

and the graphics cursor is controlled with statements like

```
MOVE 700,450
```

Once the statement VDU5 has been given only one cursor is active (the graphics cursor). This enables text characters to be placed at any position on the screen. There are a number of other effects: text characters overwrite what is already on the screen so that characters can be superimposed; text and graphics can only be written in the graphics window and the colours used for both text and graphics are the graphics colours. In addition the page no longer scrolls up when at the bottom of the page. Note however that **POS** and **VPOS** still give you the position of the text cursor.

VDU code table

Decimal	Hex	CTRL	Ascii abbreviation	Bytes extra	Meaning
0	0	@	NUL	0	Does nothing
1	1	A	SOH	1	Reserved
2	2	B	STX	0	Reserved
3	3	C	ETX	0	Reserved
4	4	D	EOT	0	Write text at text cursor
5	5	E	ENQ	0	Write text at graphics cursor
6	6	F	ACK	0	Enable VDU drivers
7	7	G	BEL	0	Make a short beeb
8	8	H	BS	0	Backspace cursor one character
9	9	I	HT	0	Forwardspace cursor one character
10	A	J	LF	0	Move cursor down one line
11	B	K	VT	0	Move cursor up one line
12	C	L	FF	0	Clear text area
13	D	M	CR	0	Move cursor to start of current line
14	E	N	SO	0	Page mode on
15	F	O	SI	0	Page mode off
16	10	P	DLE	0	Clear graphics area
17	11	Q	DC1	1	Define text colour
18	12	R	DC2	2	Define graphics colour
19	13	S	DC3	5	Define logical colour
20	14	T	DC4	0	Restore default logical colours
21	15	U	NAK	0	Disable VDU drivers or delete current line
22	16	V	SYN	1	Select screen mode
23	17	W	ETB	9	Re-program display character
24	18	X	CAN	8	Define graphics window
25	19	Y	EM	5	**PLOT K,x,y**
26	1A	Z	SUB	0	Restore default windows
27	1B	[	ESC	0	Reserved
28	1C	\	FS	5	Define text window
29	1D	]	GS	5	Define graphics origin
30	1E	^	RS	0	Home text cursoer to top left
31	1F	_	US	2	Move text cursor to x y
127	7F		DEL	0	Backspace and delete

6 **VDU 6** is a complementary code to **VDU21**. **VDU21** stops any further characters being printed on the screen and **VDU6** re-enables screen output A typical use for this facility would be to prevent a pass-word appearing on the screen as it is being typed in.

7 This code,which can be entered in a program as **VDU7** or directly from the keyboard as **CTRL** G, causes the computer to make a short 'beep'.

8 This code (**VDU8** or **CTRL** H moves the text cursor one space to the left. If the cursor was at the start of a line then it will be moved to the end of the previous line. It does not delete characters – unlike **VDU 127**.

9 This code (**VDU 9** or **CTRL** I) moves the cursor forward one character position.

10 The statement (**VDU 10** or **CTRL** J) will move the cursor down one line. If the cursor is already on the bottom line then the whole display will normally be moved up one line.

11 This code (**VDU11** or **CTRL** K) moves the text cursor up one line. If the cursor is at the top of the screen then the whole display will move down a line.

12 This code clears the screen – or at least the text area of the screen. The screen is cleared to the text background colour which is normally black. The BASIC statement **CLS** has exactly the same effect as **VDU12** or **CTRL** L. This code also moves the text cursor to the top of the text window

13 This code is produced by the **RETURN** key. However its effect on the screen display if issued as a **VDU13** or **PRINT CHR$(13);** is to move the text cursor to the left hand edge of the current text line (but within the current text window, of course).

14 This code makes the screen display wait at the bottom of each page. It is mainly used when listing long programs to prevent the listing going past so fast that it is impossible to read. The computer will wait until a **SHIFT** key is pressed before continuing. This mode is called 'paged mode'. Paged mode is turned on with the **CTRL** N and off with **CTRL** O.

15 This code causes the computer to leave paged mode. See the previous entry (14) for more details.

16 This code (**VDU 16** or **CTRL** P) clears the graphics area of the screen to the graphics background colour and the BASIC statement **CLG** has exactly the same effect. The graphics background colour starts off as black but

may have been changed with the **GCOL** statement. **VDU 16** does not move the graphics cursor – it just clears the graphics area of the screen.

17 **VDU 17** is used to change the text foreground and background colours. In BASIC the statement **COLOUR** is used for an identical purpose. **VDU17** is followed by one number which determines the new colour. See the BASIC keyword **COLOUR** for more details.

18 This code allows the definition of the graphics foreground and background colours. It also specifies how the colour is to be placed on the screen. The colour can be plotted directly, ANDed, ORed or Exclusive ORed with the colour already there, or the colour there can be inverted. In BASIC this is called **GCOL**.

The first byte specifies the mode of action as follows:

0 Plot the colour specified
1 OR the specified colour with that already there
2 AND the specified colour with that already there
3 Exclusive-OR the specified colour with that already there
4 Invert the colour already there

The second byte defines the logical colour to be used in future. If the byte is greater than 127 then it defines the graphics background colour (modulo the number of colours available). If the byte is less than 128 then it defines the graphics foreground colour (modulo the number of colours available).

19 This code is used to select the actual colour that is to be displayed for each logical colour. The statements **COLOUR** (and **GCOL**) are used to select the logical colour that is to be used for text (and graphics) in the immediate future. However the actual colour can be re-defined with **VDU 19**. For example

```
MODE 5
COLOUR 1
```

will print all text in colour 1 which is red by default. However the addition of

```
VDU 19,1,4,0,0,0    or    VDU 19,1,4;0;
```

will set logical colour 1 to actual colour 4 (blue). The three zeros after the actual colour in the **VDU 19** statement are for future expansion.

In MODE 5 there are four colours (0,1,2 and 3). An attempt to set colour 4 will in fact set colour 0 so the statement

```
VDU 19,4,4,0,0,0    or    VDU 19,4,4;0;
```

is equivalent to

```
VDU 19,0,4,0,0,0    or    VDU 19,0,4;0;
```

We say that logical colours are reduced modulo the number of colours available in any particular mode.

20 This code **VDU20** Or **CTRL** T sets default text and graphic foreground logical colours and also programs default logical to actual colour relationships. The default values are:

Two colour modes
0 =black
1 =white

Four colour modes
0 =black
1 =red
2 =yellow
3 =white

Sixteen colour modes
0 =black
1 =red
2 =green
3 =yellow
4 =blue
5 =magenta
6 =cyan
7 =white
8 =flashing black/white
9 =flashing red/cyan
10 =flashing green/magenta
11 =flashing yellow/blue

12 =flashing blue/yellow
13 =flashing magenta/green
14 =flashing cyan/red
15 =flashing white/black

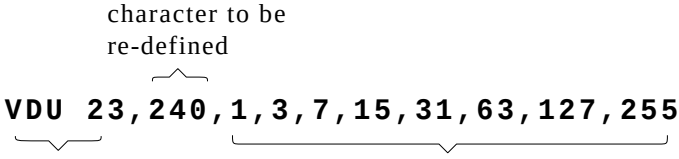
21 This code behaves in two different ways. If entered at the keyboard (as **CTRL** U) it can be used to delete the whole of the current line. It is used instead of pressing the **DELETE** key many times. If the code is generated from within a program by eithev **VDU21** or **PRINT CHR$ (21):** it has the effect of stopping all further graphics or text output to the screen. The VDU is said to be disabled. It can be 'enabled' with **VDU6**.

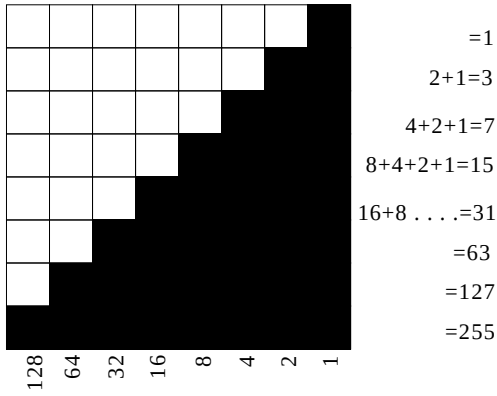
22 This VDU code is used to change MODE. It is followed by one number which is the new mode. Thus **VDU22,6** is exactly equivalent to **MODE6** (except that it does not change **HIMEM**).

23 This code is used to re-program displayed characters. The ASCII code assigns code numbers for each displayed letter and number. The normal range of displayed characters includes all upper and lower ease letters, numbers and punctuation marks as well as some special symbols. These characters occupy ASCII codes 32 to 126 If the user wishes to define his or her own characters or shapes then ASCII codes 224 to 255 are left available for his purpose. In fact you can re-define any character that is displayed. but extra memory must be set aside if this is done, and this is explained in appendix D).

ASCII codes 0 to 31 are interpreted as VDU control codes and this chapter is explaining the exact functuim of each. 'Thus the full ASCII sel consists of all the VDU control codes, all the normal printable characters and a user defined set of characters.

For example if the user wishes to define ASCII code 240 to be a small triangle then the following statement would have to be executed.



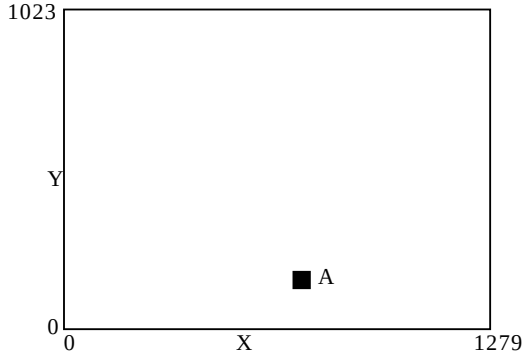


As explained above the user may define any ASCII code in the range 224 to 255. To display the resultant shape on the screen the user can type

PRINT CHR$ (248) or
VDU 240

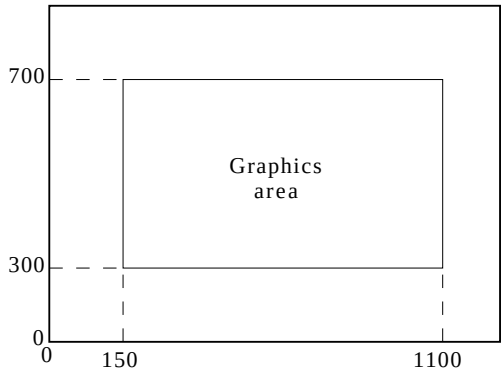
In the unlikely event of the user wishing to define more than the 32 characters mentioned above (ASCII 224 to 255) it will be necessary to allocate more RAM for the purpose.

24 This code enables the user to define the graphics window – that is, the area of the screen inside which graphics can be drawn with the **DRAW** and **PLOT** statements. The graphics screen is addressed with the following coordinates.



Thus the coordinates of A would be approximately 1000,200.

When defining a graphics window four coordinates must be given; the left, bottom, right and top edges of the graphics area. Suppose that we wish to confine all graphics to the area shown below.



The left hand edge of the graphics area has an X value of (about) 150. The bottom of the area has a Y value of 300. The fight hand side has X=1100 and the top has Y=700 The full statement to set this area is

```
VDU 24,150;300;1100;700;
```

Notice that the edges must be given in the order left X, bottom Y, right X, top Y and that when defining graphics windows the numbers must be followed by a semi-colon.

For those who wish tv know why trailing semi-colons are used the reason is as follows: X and Y graphic coordinates have to be sent to the VDU software as two bytes since the values may well be greater than 255. The semi-colon punctuation in the VDU statement sends the number as a two byte pair with low byte first followed by the high byte.

25 This VDU code is identical to the BASIC **PLOT** statement. Only those writing machine code graphics will need to use it. **VDU25** is followed by five bytes. The first gives the value of A referred to in the explanation of **PLOT** in the BASIC keywords chapter. The next two bytes give the X coordinate and the last two bytes give the Y coordinate. Refer to the entry for **VDU24** for an explanation of the semi-colon syntax used. Thus

```
VDU 25,4,100;500;
```

would move to absolute position 100,500.

The above is completely equivalent ta

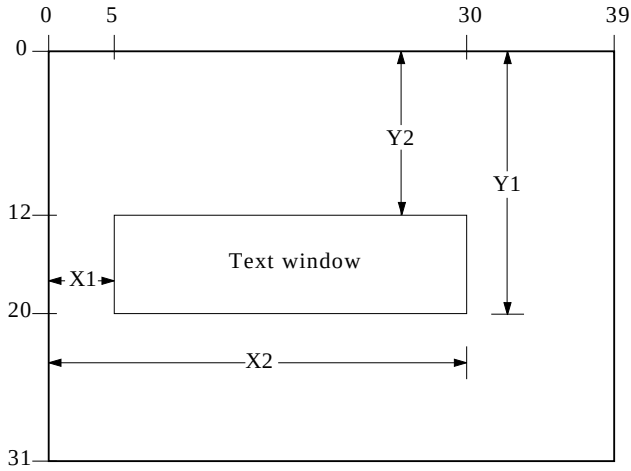
```
VDU 25,4,100,0,244,1
          X       Y
```

26 The code VDU 26 (CTRL Z) returns both the graphics and text windows to their initial values where they occupy the whole screen. This code re-positions the text cursor at the tap left of the screen, the graphics cursor at the bottom left and sets the graphics origin to the bottom left of the screen. In this state it is possible to write text and to draw graphics anywhere on the screen.

28 This code (**VDU28**) is used to set a text window. Initially it is possible to write text anywhere on the screen but establishing a text window enables the user to restrict all future text to a specific area of the screen. The format of the statement is

```
VDU 28,leftX,bottomY,rightX,topY
```

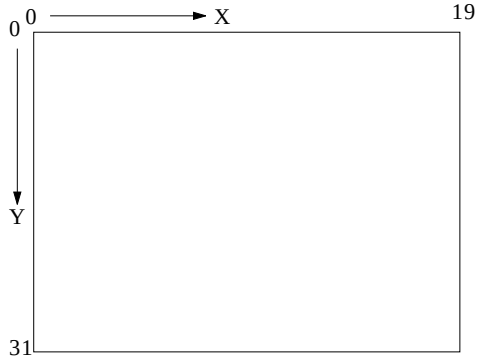
where **leftX** sets the left hand edge of the window
bottomY sets the bottom edge
rightX sets the fight hand edge
topY sets the top edge



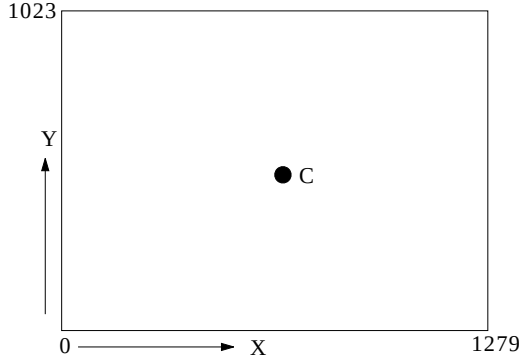
For the example shown the statement would be

```
VDU 28,5,20,30,12
```

Note that the units are character positions and the maximum values will depend on the mode in use. The example above refers to MODE1 and MODE4. In MODES 2 and 5 the maximum values would be 19 for X and 31 for Y since these modes have only 20 characters per line.



29 This code is used to move the graphics origin. The statement VDU29 is followed by two numbers giving the X and Y coordinates of the new origin. The graphics screen is addressed



Thus to move the origin to the centre of the screen the statement

```
VDU 29,640;512;
```

should be executed. Note that the X and Y values should be followed by semi-colons. See the entry for **VDU24** if you require an explanation of the trailing semi-colons. Note also that the graphics cursor is not affected by **VDU29**.

30 This code (**VDU 30** or **CTRL** ^) moves the text cursor to the top left of the text area.

31 The code **VDU31** enables the text cursor to be moved to any character position on the screen. The statement **VDU31** is followed by two numbers which give the X and Y coordinates of the desired position.

Thus to move the text cursor to the centre of the screen in MODE 7 one would execute the statement

```
VDU 31,20,12
```

Note that the maximum values of X and Y depend on the mode selected and that both X and Y are measured from the edges of the current text window not the edges of the screen.

32-126 These codes generate the full set of letters and numbers in the ASCII set.

127 This code moves the text cursor back one character and deletes the character at that position. VDU 127 has exactly the same effect as the **DELETE** key

128-223 These characters are normally undefined and will produce random shapes.

224-255 These characters may be defined by the user using the statement VDU23. It is thus possible to have 32 user defined shapes such as

```
♣VDU 23,224,8,28,28,107,127,107,8,28
♦VDU 23,225,8,28,62,127,62,28,8,0
♥VDU 23,226,54,127,127,127,62,28,8,0
♠VDU 23,227,8,28,62,127,127,127,28,62
```

Try typing each of the lines above, remembering to press the **RETURN** key after each definition. To display any of the new definitions, type in the appropriate VDU code. For example, to display the heart, type

VDU 226 RETURN

Character defmitions 224 to 255 are stored in a block of memory reserved for them in the computer. If however, you need more characters, or you want to re-define some of the keyboard characters, the best way to do this is to tell the computer to set aside extra memory to store them. (If you don't, you may run into problems). The operating system call ***FX20** described in Appendix D enables you to do this.