Pegasus Programmable Character Graphics

This documentation assumes that the Graphics option has been correctly installed and tested through your Pegasus dealer.

THEORY OF OPERATION

Pegasus uses a technique called programmable character generation. On the Pegasus screen there are sixteen lines of thirty-two characters each. This means that there are 512 bytes of video RAM in the system, from $BE00 to $BFFF.

When graphics is in effect, each character position on the screen is replaced with a programmable character, each of which is sixteen dots high by eight dots wide. This gives an effective graphics resolution of 256x256 points on the screen.

There are 128 programmable characters in 2K of graphics RAM. Any of these may be programmed with the pattern of dots that you specify in an 8x16 format. To program a character, we make use of the ability of each language operating on the Pegasus to print out an 8 bit character. When printing normal characters we can specify the character as a number between 0 and 255, of which seven bits specify the character, and the most significant bit activates inverse video for that character. As an example of this, here are three programs in BASIC, Forth and Pascal that will output the letter 'A', given that it is character ASCII 65.

BASIC:
```
10 PRINT CHR(65),
20 REM Comma supresses CR LF
```

FORTH:
```
DECIMAL 65 EMIT
```

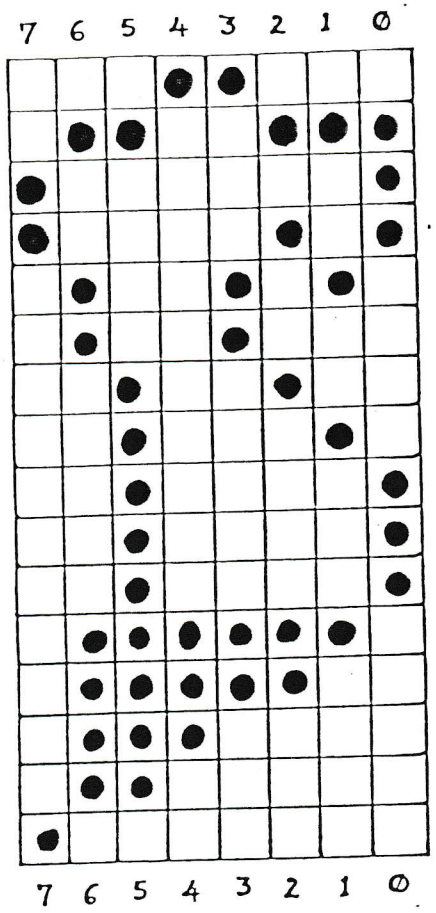
Pascal:
```
PROGRAM FRED;
VAR C:CHAR;
BEGIN
  C:=65;
  WRITE(C)
END.
```

A summary of the ASCII character set is given in the section on the system monitor in the Pegasus user manual. Either the decimal or hex versions may be used, as long as the language will handle them correctly and without confusion.

To program a programmable character, we print, emit or write a series of 18 bytes or characters, without any intervening new lines or other characters.

The first byte is the escape code, ESC, decimal 27, hex $1B. This is a special code which tells the computer that a graphics sequence is to follow. The second byte is the actual character to be programmed. Since there are 128 of them, then this character should have a value in the range of 0 to 127, which uses only seven bits of the character.

The following 16 bytes gives the actual bit pattern for the graphics character. All 8 bits in each byte is used, which gives you the 8 x 16 resolution. The character is seen as a grid, starting from the top, with the leftmost dot being the most significant bit of the byte. Below is an example of the grid required, and the translation to the numbers used for printing.



BINARY	HEX	DECIMAL
00011000	18	24
01100111	67	103
10000001	81	129
10000101	85	133
01001010	4A	74
01001000	48	72
00100100	24	36
00100010	22	34
00100001	21	33
00100001	21	33
00100001	21	33
01111110	7E	126
01111100	7C	124
01110000	70	112
01100000	60	96
10000000	80	128

Once this manual encoding step is completed, the next phase is to print the 16 bytes, starting always from the top. Thus, the numbers printed would be 24,103,129,133,74 etc, except the character form of the numbers given would need to be printed.

Once you have defined yourself a set of programmable characters, you may then use them. This requires that the computer be set into graphics mode, which can be done in two ways - either type control-G at the keyboard, or

PRINT CHR(7)

Note that if you use control-G from the keyboard, then that character should be removed from the input buffer of BASIC or Forth using the BACKSPACE key - for Pascal or the System Monitor this does not matter so much. To put the computer back to its normal mode of operation, use control-C, or

PRINT CHR(3)

Again, the same applies to control-C as to control-G. Note that the graphics characters appear instantly on the screen when you enable graphics mode. They will stay there until you change them or until you disable graphics. Note that each character on the screen in normal mode has a graphics equivalent in graphics mode - although when no shape is specifically programmed in, they may appear as a random collection of lines and dots.

To cause your programmed graphics to appear on the screen, simply ensure that the computer is in graphics mode, and then place on the screen the characters that correspond to the byte that was second in the 18 byte sequnce you provided for each programmable character. There are two ways of placing characters on screen: either by printing them, after moving the cursor to where you wish it to appear (with the square brackets in BASIC, for instance), or by poking their equivalent directly into video RAM in the range of $BE00 to $BFFF. If you are poking directly, there are two things to note: (1) Turn the cursor off if you are going to be poking anywhere near it, and (2) Remember that a byte with MSB clear is inverted, while MSB set gives normal characters. Programmable characters can be inverted, so this can lead to some interesting screen effects if you are imaginative.

Careful use of the graphics can give you animation capabilities, and if you get tired of ASCII then you can design your own character set. This form of graphics is ideally suited to video games, so experiment with writing your own Space Invaders games, for example. If you have any problems with your graphics, simply let us know, and we will endeavour to help you.

Control Y, the read graphics function, will read the graphics data for the following character into Ram locations $BDA0 - $BDAF. USER programs must then get this data directly from here. eg PRINT CHR(25),CHR($20) This will read from the graphics Ram the shape of the space character (as it is currently defined) in memory starting at $BDA0

Graphics characters can be read back form the programmable character generator RAM by sending a control Y function code followed by the ASCII code you wish to read. The data for the character's shape will be depoisted in memory at locations BPAØ to BDAF, where it can be read by USER programs. Basic for instance would usually transfer the data into the @ array for processing.

As an example, the following program examine the shape of the "A" programable character and put the data into @ (Ø) to @ (15).

```
10 PRINT CHR (25), "A",
20 FOR I = Ø TO 15
30 @ (I) = PEEK ($BDAØ+I)
40 NEXT I
```