USER MANUAL

Graphics ROM

The graphics extension ROM for the BBC micro

COMPUTER CONCEPTS

CHANGES TO THE LATEST VERSION

In response to a large number of requests we have changed the operation of one of the commands in the GRAPHICS EXTENSION ROM.

On all versions prior to 2.00 the GRAPHICS ROM had a command to disable it when required. This command has now been changed so that the GRAPHICS ROM is DISABLED until you issue a command to ENABLE it.

TO ENABLE THE GRAPHICS ROM ENTER -

```
*FX162
```

TO DISABLE THE GRAPHICS ROM ENTER -

```
*FX162,128
```

This means that before using any of the commands listed in this manual the user must enter *FX162. This command may be incorporated into a BASIC program before any GRAPHICS ROM commands are used in that program.

Once the ROM has been enabled or disabled with this command it will remain in that state, even after a hard-BREAK, until the opposite command is issued.

Therefore any other software that would normally clash with this ROM (it may have the same command names or use the same memory areas) can be used with the GRAPHICS ROM disabled.

CONTENTS

It is vital that the registration card supplied with THE GRAPHICS EXTENSION ROM is filled with your name and address and returned to us. The card is postage paid for the U.K. If for any reason a registration card is not supplied, you must contact the dealer from whom this package was purchased. The serial number on the registration card should be printed inside this manual. This number must be quoted in any correspondence with regard to this ROM.

Due to increasing software piracy, a reward of up to £500 is offered to anyone providing information leading to a successful settlement against any dealer, school, individual etc. making copies of this or any other COMPUTER CONCEPTS software package.

Computer Concepts cannot be held responsible for any loss of data due the use of this ROM.

INTRODUCTION

This GRAPHICS EXTENSION ROM contains a large number of useful graphics related commands. The manual is split into three distinct sections. First, an introduction to SPRITE graphics followed by a detailed explanation of the commands associated with SPRITES. Secondly a section that deals with LOGO 'turtle' graphics and the related commands. The third section details the remaining general purpose commands.

All of the commands, with the exception of one, are star commands and therefore they must be preceded by an asterisk whenever they are used. They can be included in BASIC programs in exactly the same way as all other Operating System commands. (See the User Guide page 416.) This does mean however that, like other O.S. commands, there must be no other BASIC statements following the star command.

The operation of these O.S. commands is quite simple. Whenever BASIC comes across a command that starts with an asterisk, it passes the whole of the remaining line over to the Operating System software. The O.S. then offers this command line to each ROM in turn, starting at ROM number 15. If a ROM recognises the command line it will act on it there and then, otherwise it will pass it on to the next ROM down the line. Once the command has been dealt with, control is returned to BASIC which will carry on with the next line in the BASIC program. This explains why there can be no more BASIC statements after the star command.

If there are a large number of ROMs before the GRAPHICS EXTENSION ROM, then the time wasted asking all the ROMs can affect the speed of execution of the graphics commands. Therefore it is best to have the graphics ROM fitted to the highest priority socket, number 15. On a normal BBC this is the ROM socket closest to the edge of the circuit board. However if a ROM extension board has been fitted it is usually one of the sockets on this board.

Most of the commands in this ROM expect some arguments to follow. For example, the *CIRCLE command expects three numbers, the X and Y position on the screen, and the radius of the circle. These arguments would normally be numbers.

*CIRCLE 640,512,100 <RETURN> would draw a circle in the middle of the screen with a radius of 100. It is not normally possible to pass variables to O.S. commands. However the GRAPHICS EXTENSION ROM does allow integer variables (A% to Z%) to be included in the arguments. If a command requires more than one argument then these should be separated either by spaces or commas in the normal O.S. fashion. All of the following are legal -

```
*CIRCLE 500,500 100
*CIRCLE X% Y% R%
*CIRCLE 500 500 R%
```

The *PRINT command is the only one in this ROM that expects a string to follow. As normal, the quotes around the string are optional. However they are usually recommended to avoid ambiguity. If the string contains a comma or a space then they are compulsory.

***HELP**

The GRAPHICS EXTENSION ROM contains an extensive help menu. This can be listed at any time by typing -

```
*HELP GRAPHICS     <RETURN>
```

or any abbreviation usually down to -

```
*H.G.    <RETURN>
```

The help menu should look something like this -

```
*HELP GRA.

      GRAPHICS EXTENSION 2.00
        RESERVE <adr> <end adr>
        DATA (<ID>)
        DESIGN <ID> <Xs> <Ys>
        FILM <ID> <anm>
        ALTER <ID> (<to ID>)
        RESET <ID> (<to ID>)
        PUT <fsp> <ID> (<to ID>)
        GET <fsp>
        IN <ID> (<X,Y>) (<frame>)
        OUT <ID>                                              continued.
```

```
IMAGE <ID> <X,Y>
TURTLE (<op>) (<Xs,Ys>) (<col>)
POS <X,Y> (<angle>)
PENDOWN
PENUP
LEFT <angle>
RIGHT <angle>
FORWARD <dist>
BACKWARD <dist>
SCALE <X,Y> <X,Y>
ROTATE (<angle>) (<X,Y>)
PIXEL <X,Y> (<Xs,Ys>)
PRINT <str> <X,Y> <Xs,Ys>
CIRCLE <X,Y> <radius>
PATTERN <op> <X,Y> <Xs,Ys> (<step>)
ARC <op> <X,Y> <Xs,Ys> <stt,end>
FILL <X,Y> <col>
PLOT <op> <X,Y,Z>
GFX <fn> (<arg>) (<arg>)
```

OS 1.20

This represents a list of all the commands that the ROM understands. Like the manual they are split up into three sections. The syntax for each command is displayed showing the arguments required to follow, separated by angle brackets. All arguments that are optional are in round brackets, and these, if not included, will default to sensible values. For example the *ROTATE command will assume a rotation of zero unless a value follows.

ABBREVIATIONS

It is possible when using any of the commands in this ROM to abbreviate the command word by ending it in a full stop. The actual minimum abbreviation will depend on the other ROMs fitted to the machine, but the general rule is that it should be long enough to distinguish it from any other commands with a similar name. For example the command *FOR. is sometimes used as an abbreviation for the disc command *FORMAT. This however could be confused with the *FORWARD command in this ROM. When including any star commands in BASIC programs is it safest to use the full command.

The meaning of the syntax given for some of the commands in the help menu may not be obvious to the newcomer to this ROM. There follows a list explaining the less obvious. For more details see under the appropriate command name.

<adr> refers to a memory address. Given in HEX.
<ID> refers to the ID number for a SPRITE or FILM.
<anm> refers to the animation sequence.
<fsp> in the normal manner refers to the filename.
<op> refers to the plotting option. See page 319 of the User Guide.
<Xs,Ys> refers to the X and Y size in normal BBC co-ordinates.
<col> refers to the logical colour. 0 to 15.
<X,Y> refers to the X,Y position on the screen.
<angle> is an angle in degrees 0 to 360.
<dist> is the distance to move in normal BBC co-ordinates.
<str> refers to a string of characters, usually surrounded by quotes.

ERRORS

If a command has been entered incorrectly then an error will be produced. This can be quite useful, for example if you are not sure of the syntax of any command or perhaps you cannot remember all the arguments, then simply typing the command and pressing <RETURN> will usually produce an error and list the syntax of the command.

*CIRCLE <RETURN>

will tell you that this command expects the X,Y position and the radius to follow. e.g.

Syntax: *CIRCLE <X,Y> <radius>

Error codes and messages:

Message	Code	Reason
Bad ID	129	An attempt was made to perform some SPRITE based command with an illegal ID number outside the range 0 to 31.
No room	130	An attempt was made to DESIGN or GET a SPRITE or make a FILM without enough space in memory. This may be because the *RESERVE command had not been used or not enough memory was reserved. This error will also be given if an attempt is made to reserve an area in memory which is not allowed, e.g. *RESERVE 0,100. (See *RESERVE on page 13).
Bad number	131	An illegal character was given in a command line when it was expecting a numerical argument. This may be due to using non percentage variables or typing errors, e.g. *IN 1,A,C or *IN 1,%.
ID used	133	An attempt was made to load in or design a SPRITE or make a FILM using an ID number that already existed.
Bad mode	134	An attempt was made to use or design SPRITES when not being in graphics mode 0, 1 or 2.

Message	Code	Reason
Bad size	135	An attempt was made to design a SPRITE with a size not in the range from 1 to 3.
Too big	136	A number was entered that was too big for the GRAPHICS EXTENSION ROM to handle, such as any number greater than 65535. This error would also be given if a line of text in a command line was greater than 248 characters, or if an attempt is made to design a FILM with more than 47 frames.
Syntax:	137	If a command is entered with an incorrect number of arguments, either too few or too many, then the correct syntax of the command will be given.
Bad scale	139	An attempt was made to scale the screen either too large or too small, such as *SCALE 0,0,0,0. (See *SCALE on page 45).
Bad data	141	An attempt was made with *GET to load a file from tape or disk which did not consist of SPRITE or FILM images. This would occur, for example, if a BASIC program is loaded using *GET.
Corruption	142	If a FILM sequence or vital information at page &C is disturbed then when an attempt is made to use the information this error will be given and the command being executed will be halted.
Bad sprite	143	This error will occur if an attempt is made to do something with a SPRITE that has not been defined. This error will also occur if an ID of a FILM is given in a FILM sequence or if an attempt is made to alter an ID that is not a SPRITE.

SPRITES

The BBC Micro supports a user definable character set that enables the user to alter or create his own definitions of the shape of any printed character. While this is quite useful, it does have limitations - characters are always two colour, usually black and white, and the size is limited to an 8 by 8 grid, so that each character can only be a maximum of 8 dots across by 8 dots down. A SPRITE is much like the user definable character except that it can contain any number of colours. It can be much larger in size, up to 3 characters across by 3 characters down, i.e. 24 dots across and down. Another disadvantage of user definable characters is that it is a complex operation to define the shape of the character and store this is memory. A large number of utility programs have been published to help in this definition process. However the GRAPHICS EXTENSION ROM includes its own definition routine, called by typing *DESIGN, that displays a grid on the screen. The user simply fills in the dots with the appropriate colours.

It is becoming increasingly common to find computers that support SPRITE graphics. The BBC Micro however does not have SPRITES as standard, although in other respects its graphics abilities are unsurpassed. Most other computers that do support SPRITES, sometimes called MOBs (Movable Object Blocks), have special hardware facilities that allow very fast and easy movement of these SPRITES on the screen. The GRAPHICS EXTENSION ROM attempts to simulate these effects in software. The advantage of the 'soft' approach is that it is much more flexible. This ROM allows up to 32 SPRITES, of varying sizes, to be defined. These may be put anywhere on the screen and may be part of an animated sequence of different 'frames'. On the other hand, the advantage of having hardware controlled SPRITES is that they are usually faster and suffer less from flicker.

To use SPRITES to their full advantage, the user should really have a good grasp of the memory map of the BBC machine and have some idea of the various memory areas and their uses. The *RESERVE command expects the addresses to be given in HEX so that a good knowledge of HEX notation is also an advantage. However, having said that, the novice can still use SPRITES and should have no problem in getting to grips with the ideas involved.

Like user defined characters, SPRITES once defined need to be stored in memory. In fact, a SPRITE needs much more memory than a normal character as it can be much larger and can contain any colour. Therefore the first thing to do

when wanting to use SPRITES is to reserve a section of memory for the SPRITE definitions. This is done with the *RESERVE command giving the start and end addresses of the section of memory to be reserved. Since SPRITES can only be used in screen modes 0, 1 and 2, the last possible address that can be reserved is 2FFF. A good start address might be 2000 - this will allow a large number of mode 2 sprites to be defined. There is a table of memory usage for the different sprites given under the *RESERVE command. This will allow the more experienced user to calculate the exact memory requirements for his specific application.

It doesn't really matter if you do not understand the numbers involved at this stage. Simply remember that before any SPRITES are used, the *RESERVE command must be issued. For experimental purposes the command -

*RESERVE 2500 2FC0 <RETURN>

will reserve an adequate section of memory in a sensible place.

Now it is possible to design the SPRITE shape. Not surprisingly this uses the *DESIGN command. This command requires three numbers to follow it. The first is the number of the sprite being defined. This will often be referred to as its ID number in the help menu and in the manual, and may be in the range 0 to 31. The following two numbers specify the size of the SPRITE. Once entered, this command will display a grid on the screen and it is a simple matter to fill in the relevant squares with any of the allowable colours. Once the SPRITE shape has been designed, pressing ESCAPE will store the design in memory in the reserved area.

There are two commands that will actually display the SPRITE on the screen. These are *IN and *IMAGE. The IN command is used when the SPRITE is to be moved about the screen. Whenever *IN is used the SPRITE is removed from its previous place and re-plotted at the new specified position. Therefore to move SPRITE 1 up the centre of the screen the following very simple routine may be used -

```
10 FOR Y%=0 TO 1000
20 *IN 1 500,Y%
30 NEXT
```

Y% goes up in steps of one in the above example but, as usual, a step of 4 would be faster and smoother as each vertical pixel on the screen represents 4 points in the BBC co-ordinates system. Similarly, horizontal movement is fastest with a step of 16 as each position along the X axis represents 16 points in the normal BBC co-ordinates.

The *IMAGE command simply leaves an image or a copy of the specified SPRITE on the screen. The *OUT command is the opposite of the *IN command and will remove any SPRITE from the screen. Once designed it is a simple matter to alter the SPRITE shape. Again, not surprisingly, the *ALTER command allows the SPRITE specified to be altered.

FILMS

As previously mentioned a SPRITE can have any ID number between 0 and 31. In fact, each ID number can be associated with a SPRITE or a FILM. A FILM is simply a sequence of frames, each frame consisting of a SPRITE.

For example, SPRITES 1 to 5 may be defined as 5 separate stages of a man taking one step, i.e. he moves forward slightly in each frame. This is exactly like each frame of a real photographic film of a walking man. Now we can tell the computer that ID number 6 represents the sequence of SPRITES 1 to 5. The command would be -

```
*FILM 6 1,2,3,4,5    <RETURN>
```

This assigns the film to ID number 6, consisting of 5 separate frames being SPRITES 1,2,3,4 and 5. *IN is also used to display FILMS. The only difference is that every time the *IN command is used it will remove the last frame from the screen and plot in the next frame in the sequence. The following short program would move him across the screen.

```
100 FOR X%=0 TO 1000 STEP 16
110 *IN 6 X%,500
120 K=INKEY(25)
130 NEXT
```

Because ID 6 has been defined as a FILM and not a SPRITE, line 110 will not put the same SPRITE in every time but a different SPRITE as defined in the FILM sequence.

There are a few more commands related to SPRITES and FILMS. The *DATA command will tell the user which ID numbers have been used and whether it is a SPRITE or FILM. This will also display how much of the reserved memory has been used. The *RESET command should be used to delete any SPRITE or FILM from memory.

Once the SPRITES have been defined they can be saved onto tape or disc by using the *PUT command. This allows the user to save any one, or a number of SPRITES together. The *GET command does the opposite, namely to re-load the ID definitions back into the reserved area.

SPRITES are EXCLUSIVE OR plotted on the screen. This method enables SPRITES to move over other objects already on the screen without destroying them. This does have one side effect, namely that if one SPRITE is plotted exactly on top of a similar SPRITE they will both cancel each other and no image will appear. It also means that when putting a SPRITE on top of a coloured area the SPRITE colours will change, as it moves over the area.

There now follows a complete explanation of all SPRITE related commands.

***RESERVE** <adr> <end adr>

It is essential that this command is used prior to loading or designing SPRITE or FILM definitions. This command must be used to reserve a single block of memory for SPRITES and FILM sequences. The block of memory should be put away from the user's program otherwise destruction could occur. A reasonably safe place will usually be from about &2500 to &2FC0 although sometimes more memory may be needed. For those unfamiliar with the BBC memory map see page 69.

If memory has already been reserved and SPRITES or FILMS have been defined then no further reservation of memory is allowed.

Only areas between &E00 to &2FFF are valid for reservation.

If when designing SPRITES or FILMS the error message 'No room' occurs, this means that the reserved block of memory has been filled. At this stage two things may be done.

1) Use the *RESET command to get rid of any unwanted SPRITE/FILM definitions.
2) Save all the SPRITES and FILMS on tape or disk using *PUT. Then enter *RESET 0,31 (this enables more memory to be reserved) and then reserve a larger block of memory than before and get the files back from tape or disk using *GET. (See example).

Arguments:

<adr> The start address of the block of memory for SPRITE definitions.

<end adr> The end address of the block of memory.

Examples:

*RESERVE 2500 2FC0
(will reserve a block of memory from &2500 to &2FC0)

An example of what to do if you run out of memory.

*PUT "ITEMS",0,31
Save all existing SPRITES on tape or disk.

*RESET 0,31
Reset or delete all SPRITE definitions to enable the *RESERVE command to work.

*RESERVE 1500,2FC0
Reserve a larger section of memory.

*GET "ITEMS"
Get the SPRITE and FILM definitions back from tape or disk.

Or, if you have SPRITES you no longer require, for example 4,5,6 and 7 you could enter *RESET 4,7, so giving more room for the design of new SPRITES.

SPRITE MEMORY USAGE

To calculate exactly how much memory a SPRITE will use, look up how many bytes are used by a one character SPRITE (1*1) in that mode and multiply by the number of characters in the SPRITE. For example a MODE 2 SPRITE, 2 by 3 in size (6 characters) is six times 32. i.e. 192 bytes. A MODE 1 SPRITE, 3 across by one down, would be 3 times 16 i.e. 48 bytes.

MODE 2 1*1=32 bytes
3*3=288 bytes

MODE 1 1*1=16 bytes
3*3=144 bytes

MODE 0 1*1=8 bytes
3*3=72 bytes

***DATA** (<ID>)

This command provides data about selected or all ID's. It tells the user which ID's have been used and whether they are SPRITE definitions or FILMS. It also tells the user the memory location where SPRITE/FILM definitions are stored and in the case of SPRITES the screen mode and X by Y character size. If the user wishes to be able to use the information in a program then use the *GFX command (see page 64).

Argument:

(<ID>) OPTIONAL. If this argument is given then specific information about the stated ID will be given. If the ID is a FILM then the FILM sequence will be listed for possible editing. If the argument is omitted all the defined ID's will be listed with other information.

Examples:

```
*DATA
 0 SPRITE XRES 3 YRES 3 MODE 2 ADR 2000
10 SPRITE XRES 2 YRES 2 MODE 0 ADR 2120
31 FILM ADR 2140
USED FROM 2000 TO 2170 LEAVING 0100
```

This is the general format for information given with *DATA. The first number is the ID number. It then tells you if the particular ID number is a SPRITE or FILM. If it is a SPRITE then the next two numbers indicate its size followed by the screen mode in which the SPRITE was defined. The ADR is the memory address in hex where the SPRITE is stored. In this example ID 0 is a SPRITE with width and height of size 3, it was designed in MODE 2, and is at &2000 in memory.

Information on memory usage is given at the end. In this example memory has been used from &2000 to &2170 and it tells us that we have &100 left.

To get more information about a FILM then type the FILM ID number on its own after the *DATA command. e.g.

```
*DATA 31
*FILM 31 1 2 3 4 5
```

***DESIGN** <ID> <Xs,Ys>

Before using this command, be sure to read the section on SPRITES. This command can be used to design SPRITES (multi sized and coloured characters.) However before the SPRITE can be designed two things must first be done:

1 - The graphics mode in which the SPRITE is to work must be entered into by using either the BASIC MODE or VDU 22 command (see pages 301 and 383 in the User Guide). SPRITES may only be designed in graphics modes 0,1 and 2.

2 - Memory must be set aside for the SPRITE definition. This is done using the *RESERVE command.

When this command is used a grid will appear on the screen. The SPRITE will be designed in the grid. Above the grid is a palette of colours and below is the ID number of the SPRITE currently being defined.

Two crossed cursors are used in the program. One cross is used in the grid and its position denotes which particular cell is to be coloured. The other cross is used to select the (logical) colour to be used in designing. When the <RETURN> key is pressed then the selected colour will be put in the grid cell. As well as an enlarged picture of the SPRITE appearing in the grid (as it is designed) an actual sized SPRITE will also be shown in the top right hand corner of the screen.

This whole command is probably best explained by using the command in trial runs to become familiar with it.

Arguments:

<ID> The ID number or identification number of the SPRITE to be designed.

<Xs,Ys> The size of the SPRITE. The <Xs> is the width of the SPRITE and <Ys> is the height of the SPRITE. They both have a range from 1 to 3. The actual sizes for these values is best seen by example.

CONTROL KEYS FOR THE GRID POSITION CROSS

'X' ------- Move grid cross right
'Z' ------- Move grid cross left
':' ------- Move grid cross up
'/' ------- Move grid cross down

CONTROL KEYS FOR THE COLOUR PALETTE

'>' ------- Move palette cross right
'<' ------- Move palette cross left

<RETURN> -- Puts into the selected grid square the chosen palette colour.
<SHIFT> --- Puts black (logical 0) into the selected grid square, ie.it deletes the colour.

Once the SPRITE has been designed either the <ESCAPE> key or the <TAB> key may be pressed.

<ESCAPE> -- This causes the SPRITE definition to be downloaded into the reserved memory. Then the DESIGN command will be exited leaving a clear screen.

<TAB> ----- This causes the SPRITE definition to be downloaded into the reserved memory as for <ESCAPE>. It also causes the ID number to be increased until it reaches an undefined ID number. Then definition of this new SPRITE may continue with the same SPRITE image on the screen. This is essential in designing SPRITES in FILM sequences and can be used to make copies of SPRITES.

Examples:

```
MODE 2
*RESERVE 2000,2FFF
*DESIGN 0,2,3
```

This will enter MODE 2, then reserve a block of memory from &2000 to &2FFF. Then the SPRITE design program will be entered. The ID number of the SPRITE being designed is 0 and its size will be 2 by 3.

***FILM** <ID> <anm>

This command is used to make animation sequences or FILMS. Each FILM consists of a series of ID numbers relating to SPRITE characters to be in the FILM and must be given in the order required for showing. To 'project' the FILM simply use the *IN command, each time this command is used the next frame of the sequence is shown. When the end of the sequence is reached it will start from the beginning again.

The *IN command also allows the FILM to be shown in a random order using the <frame> argument, although this is for the more experienced users.

If the FILM sequence contains an ID of another FILM, and not a SPRITE, then an error will be produced when that frame is displayed.

Arguments:

<ID> The ID number of the film.

<anm> The animation sequence. This consists of up to 47 ID numbers of SPRITES in the film.

Example:

*FILM 31 1 2 3 4 5 6
This would define ID 31 to be a FILM consisting of the SPRITES 1,2,3,4,5 and 6.

***ALTER** <ID> (<to id>)

This is used to edit SPRITE definitions (NB. not FILM sequences, see *DATA for film editing). When editing a SPRITE it will be loaded into a grid and using the same controls as for *DESIGN the character can be 'altered' or edited. Before editing can begin the screen mode in which the SPRITE was designed must be entered. This is because a long BASIC program could be corrupted if *ALTER were to perform the mode change automatically. To find out which mode a SPRITE was defined in either use the *DATA or *GFX 0 command.

Arguments:

<ID> The ID number of the SPRITE to be edited.

(<to ID>) OPTIONAL. Once the SPRITE has been altered it will be given the new identity of (<to ID>). If this argument is omitted then the altered SPRITE will keep its original ID number.

Examples:

MODE 2
*ALTER 5
Like *DESIGN this will display a grid except that, instead of it appearing blank, an image of sprite 5 will be displayed, enabling it to be altered. All the control keys are identical to those in *DESIGN.

MODE 1
*ALTER 5,9
This will do the same as above except that after alteration the SPRITE is given ID number 9. This SPRITE 9 is the altered version of SPRITE 5.

***RESET** <ID> (<to id>)

This command may be used to 'reset' or delete ID numbers to give more space for new SPRITES or FILMS.

WARNING: When an ID is reset the SPRITE or FILM definition that had the ID will be lost for ever. So take care!

Arguments:

<ID> This is the initial ID to be reset. If only one argument is given then only this particular ID will be reset.

(<to ID>) OPTIONAL. This is the final ID to be reset. If this argument is given then all the ID's from <ID> to <to ID> inclusive will be reset.

Example:

*RESET 0,31
Will reset all 32 ID numbers. This example will always enable memory to be reserved.

*RESET 5
Will reset ID 5 only.

*RESET 9,12
Will reset ID's 9 to 12 inclusive.

***PUT** <fsp> <ID> (<to ID>)

This command saves SPRITE or FILM definitions onto the current filing system. If only two arguments are given, then only one SPRITE/FILM definition will be saved. If all three arguments are given then a range of SPRITES and/or FILMS will be saved. If an attempt is made to save an ID that has not been defined then nothing will be saved.

Arguments:

<fsp> FILE SPECIFICATION. The name of the file.

<ID> The first ID to be saved.

(<to ID>) OPTIONAL. The last ID to be saved. If this argument is given then all the SPRITES in the range <ID> to <to ID> will be saved.

Examples:

*PUT "JIM",12

This will (assuming ID 12 exists), save the SPRITE or FILM with an ID of 12 on the current filing system.

*PUT "FRED",0,31

This will try and save all 32 possible ID numbers on the current filing system from 0 to 31. If any of the ID's have not been used they will not be saved.

***GET** <fsp>

This command gets SPRITE definitions or FILM sequences from the current filing system, usually disk or tape. Before this can be done memory must be reserved using the RESERVE command.

The file on tape or disk must already have been put there using the *PUT command. The file may consist of one or several SPRITES/FILMS. When the SPRITES are loaded in they will still have the same ID numbers they had when saved. If an ID number has already been used when loading then the error 'ID used' will be given.

If the error message "Channel" is given this means that the given filename could not be found.

Arguments:

<fsp> FILE SPECIFICATION. The name of the file to be loaded.

Examples:

*GET "FRED"
This will load in SPRITES or FILMS from a file called "FRED" from the current filing system.

***IN** <ID> <X,Y> (<frame>)

This command can be used to place a predefined SPRITE or FILM on the screen. If an attempt is made to plot a SPRITE either totally or partially off the screen the SPRITE/FILM will not be shown. It is possible to show any frame of a FILM using the <frame> argument.

Arguments:

<ID> ID number of the SPRITE or FILM to be placed on the screen.

<X,Y> X and Y co-ordinates for the position of the SPRITE/FILM on the screen. NOTE: If the screen has been scaled or rotated using *SCALE or *ROTATE then the SPRITE position will be affected by those commands.

(<frame>) OPTIONAL. When using FILMS it is possible to select any frame in that FILM by giving the (<frame>) argument. Thus the FILM may be shown in any random order.
This argument will have no effect if the selected ID is one of a SPRITE. The value must not exceed the length of the film strip. (The FILM strip length may be found out using a *GFX 0).

Examples:

*IN 31,200,300
Will put in ID 31, (whatever it may be), at a screen co-ordinate of (200,300).

*IN 22,100,100,23
Put in the 23rd frame of a FILM with ID of 22 at (100,100).

***OUT** <ID>

This command removes a SPRITE/FILM from the screen. If the SPRITE was not on the screen initially then nothing will happen.

Argument:

<ID> The ID number of the FILM/SPRITE to be removed from the screen.

Example:

*OUT 5

Will remove SPRITE/FILM number 5 from the screen, assuming it was on the screen to begin with.

***IMAGE** <ID> <X,Y>

This function is very similar to *IN except instead of the SPRITE actually moving, a copy or 'image' of the SPRITE is made.
NOTE: It is not possible to make an image of a FILM.

Arguments:

<ID> The ID number of the SPRITE to be copied.

<X,Y> The (X,Y) co-ordinate on the screen where the image is to be placed. If the point is off the screen then it will not be copied.

Example:

*IMAGE 4,100,100

Will put an image of sprite 4 at position (100,100).

SPRITE GRAPHICS EXAMPLES

There follows a complete example of how to design SPRITES and FILMS and how to save and load them from disk or tape. Before attempting to enter any of this example be sure to read everything in the SPRITE section.

Only enter the commands not enclosed in brackets.

EXAMPLE:

Before we can design SPRITES or FILMS we must first reserve some memory for them.

*RESERVE 2800,2FFF
This will reserve a single block of memory from &2800 to &2FFF. Do not worry if you do not understand the numbers.

MODE 2
*DESIGN 0,1,1
When this is entered a grid of 8 by 8 blocks will be displayed on the screen in mode 2, as specified. The size of the grid is one character across and down. The top of the screen will show a palette of 16 colours. There will be a cross in the red box which means the selected colour for plotting in the grid will be red. A larger cross will be in the grid down in the bottom left corner. This is used to position the selected colour in the grid. The grid itself represents the SPRITE character to be designed. Just under the palette it says 'SPRITE 0' This is the ID number currently being designed. The two crosses are moved using keys on the keyboard. (See page 17 for the keys used.)

Now enter the following sequence of keys on the keyboard to design a SPRITE character. As the keys are entered various grid squares fill up with selected colour and an image of a simple man will appear. Notice also that the actual SPRITE is being plotted in the top right hand corner of the screen.

NOTE: <RETURN> has been abbreviated as R and <SHIFT> has been abbreviated as S. '>' means right cursor key. '<' means left cursor key.

Now enter carefully the following key entries.

R:XR:XR:XR/XR/XR/XR::::ZZZ>R:R:>RXR:RZRZR/R//>>RZRZRXXXXRXRXR

After typing these keys a picture of a man should be in the grid. Now press the <TAB> key. The character you have just designed will now be downloaded into the reserved block of memory. Pressing <TAB> also causes the SPRITE number under the palette to increase to 1. This means the SPRITE ID number you are now designing is 1, and you can now change the first SPRITE you had, i.e. ID 0, to something else and download this new character as ID 1. Thus after entering the next sequence of keys you will have two totally separate SPRITES.

Carefully enter the following.

SZS/RX/RZZZZZZR:XR:SZS

By entering these keys you will see that the right arm of the character was deleted using the <SHIFT> and was replaced lower down. The operation was repeated for the left arm. Now press <ESCAPE>. This will cause the second SPRITE with ID of 1 to be downloaded into the reserved area. We now have two SPRITES, one of a man with his arms out straight, and one of a man with his arms by his sides. The screen should be clear at this point as the SPRITE definition process has been stopped. Now enter in the following.

*IN 0,640,512
The first SPRITE you designed ie.that of a man with his arms out-stretched, will now appear on the screen in the centre.

*IN 1,640,700
The second SPRITE you designed will now appear above the first. In total we should now have two pictures of a man, one above the other.

*OUT 1
The top man should now disappear.

*OUT 0
The man in the centre, (the one with his arms out-stretched), should now vanish leaving a clear screen.

*FILM 2,0,1
This designs a FILM sequence. The ID number of the FILM is 2, (the first argument). The second two arguments are the ID numbers of SPRITES to be in the FILM. In this case ID's 0 and 1. (You may have up to 47 ID numbers.) Thus in doing this we have designed a FILM of a man moving his arms up and down.

Enter the following.

*IN 2,640,512
The first man should appear in the centre of the screen.

*IN 2,640,512
The man should now move his arms down. In reality the first SPRITE has been replaced by the second.

*IN 2,640,512
*IN 2,640,512
*IN 2,640,512
*IN 2,700,700
The man flaps his arms up and down. In the last one he also moves.

*OUT 2
The screen should now be clear.

Until now we have been moving a character from one place to another. However we can put copies of a SPRITE on the screen using the *IMAGE command.

*IMAGE 1,200,200
*IMAGE 1,700,900
*IMAGE 0,300,300
We now have three 'images' or copies of the man on the screen at the selected positions. NOTE: You CANNOT make images of FILMS, e.g. *IMAGE 2,500,500 would not do anything.

Having designed the FILMS and SPRITES it may be desirable to save them on tape or disk for future use. We use the *PUT command for this.

*PUT MAN,0,2
This will save on tape a file called "MAN" consisting of ID numbers from 0 to 2 inclusive, i.e. 0,1 and 2. Thus the two pictures of a man and the FILM of the man will be saved. The man may be re-loaded using *GET MAN.

We give below an example program using the designed man. When this program is RUN a man should move across the screen moving his arms up and down. Lines 40 and 50 are only there to slow down the movement. Try changing the 500 to something smaller for faster movement.

```
10 MODE 2
20 FOR A%=0 TO 1000 STEP 16
30 *IN 2,A%,500                    Put in the FILM sequence at A%,500.
40 FOR T%=0 TO 500                 Wait a while to slow it down.
50 NEXT
60 NEXT
```

Try changing line 30 to *IMAGE 1,A%,500 and re-run the program. Notice that part of the character disappears. This is due to the way the SPRITES are placed on the screen. (See section on SPRITES).

Examples of SPRITES in programs:

(1)

This program demonstrates how a program that uses SPRITES may load them into the computer from tape or disk. This particular example will try and load a file called "MAN", which was saved in the last example.

In some instances you may already have some valuable SPRITES/FILMS in memory which you want to keep. In this case simply omit lines 10 to 30.

NOTE: When developing your program you may wish to avoid having to load the SPRITES/FILMS in each time it is run. Thus it would not be unusual to have a temporary GOTO at the beginning of the program to miss out these instructions. In this example it could be '1 GOTO 60', and when the program is complete these lines may be removed.

```
10 *RESET 0,31                  This resets any existing SPRITES
20                              in memory.
30 *RESERVE 2800,2FFF           This reserves a block of memory from
40                              &2800 to &2FFF.
50 *GET MAN                     Get the file called "MAN" from tape/disk.
60 etc...
```

The SPRITES/FILMS have now been loaded. The rest of your program can continue from here.

(2)

Using *IMAGE to create patterns. Try changing the STEP 3 to something else.

```
 80 *RESET 0,31              Reset all 32 ID numbers to enable
 90                          memory to be reserved.
120 *RESERVE 2900,2FFF       Reserve a block of memory from
130                          &2900 to &2FFF
160 MODE 2                   Enter MODE 2 to design a SPRITE.
170 *DESIGN 0,3,3            Design a SPRITE as some simple pattern.
260 FOR Z%=0 TO 359 STEP 3
270 A%=640+400*SINRAD(Z%)    Get co-ordinates for a circle.
280 B%=512+400*COSRAD(Z%)    It would be faster to use *GFX 3 and 4.
290 *IMAGE 0,A%,B%           Use *IMAGE to make multiple copies of
295                          your SPRITE to make a large pattern.
300 NEXT
```

TURTLE GRAPHICS

The LOGO language is becoming increasingly popular in schools, and one of the major reasons for this is its use of 'turtle' graphics. The original 'turtle' was a mechanical device that crawled along the floor. This had a pen that could be raised or lowered onto the floor, and when lowered it would leave a trail behind. The turtle was remotely controlled from a computer and could be given commands telling it to move forward or backward a certain distance, or to turn left or right through a certain angle.

Most recent versions of LOGO represent this mechanical device as a small triangle on the screen, and this can be given commands in exactly the same manner as its mechanical brother.

The GRAPHICS EXTENSION ROM adds 'turtle' graphics to BBC BASIC. Most of the commands are fairly obvious in use. The *TURTLE command controls the turtle's size and shape, its colour, and the type of trail it leaves behind. Normally the trail would be a line, but it is possible to have triangles, dotted lines etc. The *POS should be used to position the 'turtle' on the screen at the required place and angle. For most applications this will be *POS 640,512,0.

The *FORWARD, *BACKWARD, *RIGHT and *LEFT commands are used to move the turtle around the screen. For example, to draw a square just move forward and turn right 90 degrees, four times. A procedure to do this could be something like -

```
1000 DEFPROCSQUARE(S%)
1010 FOR C%=1 TO 4
1030 *FORWARD S%
1040 *RIGHT 90
1050 NEXT
1060 ENDPROC
```

Whenever PROCSQUARE is called, a square will be drawn with sides S% long. This clearly shows the advantage of turtle graphics over the normal MOVE and DRAW technique, and is especially true when you realise that the above procedure will draw a square with the turtle pointing in any direction.

It is possible to enter the commands in direct mode i.e. typed in directly at the keyboard and not in a program. Thus a semi-LOGO can be formed. First the screen must be set up as follows -

```
MODE 1
VDU 28,0,31,39,26
CLG
*POS 640,512,0
```

With the screen set up, any of the turtle graphics commands can be used very much like many LOGOs, e.g. enter the following -

```
*FORWARD 100
*RIGHT 45
*FORWARD 200
*RIGHT 45
*FORWARD 100
*RIGHT 45
*FORWARD 200
*LEFT 90
*BACKWARD 100
*PENUP
*BACKWARD 400
*PENDOWN
*FORWARD 100
```

With a little practice in using these commands, even the most inexperienced computer users should be able to draw simple shapes. For the slightly more advanced these commands can be incorporated in BASIC programs to produce complicated patterns and shapes, as in the examples at the end of this section.

The next few pages describe each of the commands in more detail. This is followed by some simple demonstration programs.

***TURTLE** <op> (<Xs,Ys>) (<col>)

Before using any of the these commands it is essential to read the previous section on 'turtle' graphics.

This command defines the size, shape, colour and plotting option of the turtle. This basically consists of a triangle which can be made various sizes and colours.

If no *TURTLE command is used then a default turtle will be assumed as: *TURTLE 5,70,100,1

Arguments:

<op> The trail left by the moving turtle may be composed of lines, dots, or triangles. The <op> specifying which of these is to be plotted. The codes are identical to the BASIC PLOT codes (see page 319 in the User Guide).

(<Xs,Ys>) OPTIONAL. These arguments specify the width <Xs> and height <Ys> of the logo turtle. If the arguments are not given then no turtle will be used enabling shapes to be drawn much faster.

(<col>) OPTIONAL. This is the (logical) colour of the turtle. (See page 222 in the User Guide for colour numbers.) If this argument is omitted then colour 0 will be assumed - usually black.

Example:

*TURTLE 5,30,40,3
This will define a turtle 30 wide and 40 high. 5 is the plotting option which means that the turtle will draw lines, and 3 is the colour of the turtle which may vary depending on the screen mode.

Example program on next page:

(1)

```
   10 REM To demonstrate the various
   20 REM types of turtle available.
   30 REM By P.Hiscock
   40
   50 MODE2
   60
   70 REM Position the turtle in the
   80 REM centre of the screen with
   90 REM an angle of 0,ie.facing up.
  100 *POS 500,500,0
  110
  120 FORX%=10 TO 400 STEP20
  130
  140  REM Choose a random colour for
  150  REM the turtle
  160  C%=RND(7)
  170
  180  REM Set the turtle to draw lines
  190  REM with the 5. The size of the
  200  REM turtle is X%,X%. (These need
  210  REM not be the same). The colour
  220  REM is C%.
  230  *TURTLE 5,X%,X%,C%
  240
  250  REM Wait for a key press
  260  G=GET
  270 NEXT
```

***POS** <X,Y> (<angle>)

This is used in conjunction with the TURTLE graphics. The command positions the turtle (if it exists) at a specified position on the screen at a specified angle. The angle is taken relative to the vertical like a compass bearing, i.e. 0 is facing up the screen.

Arguments:

<X,Y> The (X,Y) co-ordinate where the turtle is to appear on the screen.

(<angle>) OPTIONAL. If this argument is given then the turtle will be displayed at an angle given by <angle> in degrees, otherwise the original rotation of the turtle will be assumed.

Examples:

*POS 400,400,45
Will place the turtle, assuming it has been defined, at screen co-ordinate (400,400) at an angle of 45 degrees to the vertical.

*POS 200,200
Will place the turtle at position (200,200) with the same angle of rotation it had previously.

***PENUP**

When the turtle moves it will normally draw lines, dots or triangles. This command stops the turtle from drawing or plotting anything - identical to the LOGO command PENUP.

Example:

```
*PENUP
```

***PENDOWN**

This command performs the opposite function to *PENUP and is also identical to the LOGO command PENDOWN. It basically restores the plotting option to the turtle, thus re-enabling it to leave dots, lines or triangles as was defined by the *TURTLE command. Since the 'pen' is normally down, the *PENUP command must have been used previously before this command will have any effect.

Example:

```
*PENDOWN
```

***LEFT** <angle>

This command will rotate the turtle in an anti-clockwise (left) direction by a specified number of degrees. It is identical the LOGO command LEFT.

Argument:

<angle> This is the angle in degrees through which the turtle will rotate.

Example:

*LEFT 12
Will rotate the LOGO turtle through an angle of 12 degrees in an anti-clockwise direction.

***RIGHT** <angle>

This command rotates the LOGO turtle in a clockwise direction by a specified angle. Identical to the LOGO command RIGHT.

Argument:

<angle> This is the angle in degrees through which the turtle will rotate.

Examples:

*RIGHT 45
(Will rotate the turtle through 45 degrees in a clockwise direction.)

*RIGHT R%

***FORWARD** <dist>

This command moves the turtle forwards by a specified distance in the direction the turtle is facing. If the turtle has been asked it will leave a line, dot or triangle depending on the *TURTLE setting. If a *PENUP has been used prior to the use of the *FORWARD command then no trail will be left.

Arguments:

<dist> This is the distance that the turtle will move forwards in normal BBC co-ordinates.

Example:

*FORWARD 100
Will move the turtle (if defined) forwards by 100.

***BACKWARD** <dist>

This command moves the turtle backwards by a specified distance in the opposite direction to the way the turtle is facing. If the turtle has been asked it will leave a line, dot or triangle depending on the *TURTLE setting. If a *PENUP has been used prior to the use of the *BACKWARD command then nothing will be left.

Arguments:

<dist> This is the distance the turtle will move backwards in normal BBC co-ordinates.

Example:

*BACKWARD 421
Will move the turtle backwards by 421.

Turtle graphics examples

(1)

This program demonstrates all the logo graphics commands in The Graphics Extension ROM.

```
 10 REM A lesson in logo by P.H.
 20
 30 MODE1
 40
 50 *TURTLE 5,70,100,1
 60 PRINT"The TURTLE is normal size and red.":G=GET
 70 *POS 640,212,0
 80 PRINT"TURTLE at POSition 600,200 angle 0":G=GET
 90 *TURTLE 5,30,40,2
100 PRINT"The TURTLE is now yellow, & smaller":G=GET
110 *FORWARD 150
120 PRINT"The TURTLE has moved FORWARD by 150":G=GET
130 *RIGHT 90
140 PRINT"TURTLE has rotated RIGHT by 90 degrees":G=GET
150 *BACKWARD 150
160 PRINT"TURTLE moved BACKWARDs by 150":G=GET
170 *PENUP
180 PRINT"TURTLE has its 'PENUP', no more lines":G=GET
190 *LEFT 45
200 PRINT"TURTLE has rotated LEFT by 45 degrees":G=GET
210 *FORWARD 200
220 PRINT"TURTLE moved FORWARD 200 with no lines":G=GET
230 *PENDOWN
240 PRINT"TURTLE has its 'PENDOWN', can draw again";:G=GET
250 *RIGHT 270
260 PRINT"TURTLE has rotated RIGHT by 270 degrees":G=GET
270 *BACKWARD 200
280 PRINT"TURTLE moved BACKWARDs by 200":G=GET
290 *TURTLE 5
300 PRINT"TURTLE off screen"
```

(2)

This program draws random shapes and makes them into patterns.

```
 10 MODE2
 20 I%=700
 30 J%=500
 40
 50 *POS J%,I%,90                  Position turtle at I%,J% at 90 degrees.
 60 *POS J%,I%,90                  It has been done twice since triangles
 70                                may be used to draw with.
 80 IF RND(2)=1 O%=85 ELSE O%=5    Choose either lines or triangles to
 90                                draw with.
100 *TURTLE O%                     As only one argument has been given
110                                no turtle will be used, giving faster
115                                graphics. The O% is the plot option.
120 T%=RND(50)                     Choose random values for the random
130 U%=RND(50)                     shape
140 V%=RND(50)+40
150 W%=RND(50)+40
160 X%=RND(10)*20
170 F%=RND(20)*10
180 G%=RND(20)*20
190 H%=RND(20)*5
200
210
220 REPEAT
230 GCOL 0,3
240
250 *FORWARD X%                    Go forward a distance X%
260
270 *RIGHT T%                      Turn clockwise an angle T%
280
290 GCOL 0,2
300 *FORWARD F%
310 *RIGHT U%
320 GCOL 0,1
330 *FORWARD G%
340 *RIGHT V%
350 GCOL 0,6
360 *FORWARD H%                    continued over..
```

```
370 *RIGHT W%
380
390 *GFX 2                          This command returns in A% and B% the
395                                current position of the turtle in A%
396                                and B%. See *GFX command.
400 UNTILA%>J%-8 AND A%<J%+8 AND B%>I%-8 AND B%<I%+8
405                                Keep on drawing the random shape until
406                                the pattern meets up at the start.
410 I=INKEY(100)                   Wait one second.
420 RUN                            Draw another shape.
```

(3)

Symmetrical patterns using turtle graphics.

```
 10 MODE2
 20 *TURTLE 5                      De-select the turtle and draw lines.
 30                                Try using *TURTLE 5,30,40,1 instead.
 40 C%=0
 50 FOR A%=30 TO 175 STEP 6
 60 CLS
 70
 80 *POS 640,512,0                 Position turtle in screen centre.
 90
100 FORO%=0 TO 600 STEP 4
110
120 C%=C%+1                        Select a colour to plot.
130 IF C%>6 C%=1
140 GCOL 0,C%
150
160 *RIGHT A%                      Turn the turtle right by A% degrees.
170
180 *FORWARD O%                    Move the turtle forward a distance O%.
190 NEXT
200
210 FOR W%=0 TO 1000               Wait a while.
220 NEXT
230 NEXT                           Draw the next shape.
```

GENERAL PURPOSE COMMANDS

This section deals with the more general graphics commands. Two of these, *ROTATE and *SCALE have no function of their own but they do affect all subsequent plot commands.

The *ROTATE will rotate all subsequent plotting commands around any specified point. Both the SCALE and ROTATE will affect all normal plotting going through the BBC VDU channel, such as PLOT, MOVE and DRAW. It will not affect printing (except *PRINT) and it will not affect graphics that poke directly to the screen memory, such as many 'arcade' type games. The positioning of SPRITES is affected but the actual SPRITE orientation will always remain the same.

The *PATTERN, *ARC and *CIRCLE commands are all related. It is possible to draw circles with all three commands, but the *CIRCLE is the simplest. The former two commands are quite complex and can have a large number of arguments following each. The best way to find out more about these is to experiment and try the examples given with each command.

The BBC Micro has a very fast triangle fill command. However it becomes difficult to fill shapes other than triangles, especially irregular polygons. The *FILL command will fill ANY shape on the screen, including spirals and other shapes that may contain corners, U-bends etc.

The *GFX command is a general purpose function command. It is unusual in that it normally returns a value to the calling program rather than the other way round. The first number after the *GFX controls its action. The results are always returned in the integer variables A% to E%.

3D GRAPHICS

The normal BBC co-ordinate system allows the user to specify any point on the screen by giving its X and Y positions on the respective axis. (See page 56 of the User Guide if you are unsure about co-ordinates). In real life, of course, all solid objects have three dimensions, sometimes measured as the height, width and depth.

The *PLOT commands allows the user to give 3 co-ordinates for any specified point, representing the width, height and depth of that point. The command will then display that point with the correct perspective - giving the illusion of depth. Since the screen is only two dimensional, this routine is really a 3D to 2D converter.

Of the three points given (X,Y,Z), the X and Y are exactly equivalent to the X and Y co-ordinates that would normally be given for plotting on the screen. The Z co-ordinate is the distance into the screen. A Z value of 1024 represents the screen position from the eye. This means that a line 200 long would appear to be 200 long with Z=1024: If Z is less, then the line is nearer the eye and so longer, and if Z is larger then the line will appear smaller.

The *PLOT command accepts any of the normal PLOT numbers so it is possible to plot lines, triangles, etc. in 3D.

The following program draws a square edge on. This square can then be moved around in space by using the 4 arrow keys to move it left, right, up and down. The 'F' key will move it further into the distance and the 'N' key will bring it closer. Line 20 puts the origin in the middle of the screen as this usually gives the best effect for 3D graphics.

When typing in this program remember to omit the comments to the right.

```
 10 MODE4
 20 VDU29,640;512;            Puts the origin in the centre
 30 *FX4 1
 40 X%=-100:Y%=100:Z%=1024
 50 M=50                      The square moves by this amount
 60 REPEAT
 70 C=GET                     Gets the number of the key pressed
 80 IF C=136 THEN X%=X%-M     If left arrow then decrease X%
 90 IF C=137 THEN X%=X%+M     If right arrow then increase X%
100 IF C=138 THEN Y%=Y%-M     If down arrow then decrease Y%
110 IF C=139 THEN Y%=Y%+M     If up arrow then increase Y%
120 IF C=70 THEN Z%=Z%+M      If 'F' then increase Z%
130 IF C=78 THEN Z%=Z%-M      If 'N' then decrease Z%
140 PROCSQUARE
150 UNTIL FALSE               Do loop again
160 END
170
180 DEFPROCSQUARE             Draws a 3D square
190 CLS
200 A%=X%+200
210 C%=Z%+200
220 *PLOT 69,X%,Y%,Z%         Moves to the first point
230 *PLOT 5,A%,Y%,Z%          Draws the 4 sides
240 *PLOT 5,A%,Y%,C%
250 *PLOT 5,X%,Y%,C%
260 *PLOT 5,X%,Y%,Z%
270 ENDPROC
```

It would be an easy matter to plot another shape instead of a square. Any shape that was made up of points relative to X%,Y%,Z% and incorporated into PROCSQUARE would work, and would move around with the keys.

***SCALE** <X,Y> <X,Y>

When this command is used all future PLOT, *PRINT, *PIXEL, *ARC, *CIRCLE and *PATTERN commands will be affected. The screen co-ordinates normally start at (0,0) in the bottom left and finish at (1279,1023) in the top right. The *SCALE command enables the user to change these two screen co-ordinates to make a screen of almost any size and all points in between will be scaled to that screen size. Thus, without changing a program, the finished result of a graphics display can be made larger or smaller as required using *SCALE. This is more easily seen by examples.

The total size of the screen must be greater than 12 by 12 and less than 31000 by 31000.

If, once the screen has been scaled, no future scaling is required then either enter into a new screen mode using MODE, or enter *SCALE 0,0,1279,1023, or use a *FX 162, (See page 72.)

NOTE: Once the screen has been scaled the BASIC POINT command will not work as its co-ordinates do not pass through the scaling routine. So instead of using POINT the user should use a GFX 6 to obtain the colour of a pixel at a scaled screen co-ordinate. (See page 66 for GFX6.)

Arguments:

<X,Y> The new bottom left co-ordinate of the screen.

<X,Y> The new top right co-ordinate of the screen.

Example:

```
MODE 1
```
Enter mode 1.

```
PLOT 69,100,100
```
Plot a dot in the bottom left of the screen.

Continued -

*SCALE 0,0,200,200
Scale the screen from (0,0) to (200,200)

PLOT 69,100,100
Using the same PLOT command the dot appears in the centre of the screen, since the screen ranges from (0,0) to (200,200) and therefore 100,100 is in the centre of the new screen.

Example programs:

(1)

```
   10 REM  Using a square to show the
   20 REM  effects of *SCALE.
   30 REM  As the scale increases
   40 REM  numerically, the size
   50 REM  of the square decreases.
   60 REM  NOTE: The square is drawn
   70 REM  the same size each time.
   80
   90 MODE1
  100 FORX%=200 TO 3000 STEP 10
  110 CLS
  120 *SCALE 0,0,X%,X%
  130 PROCSQUARE
  140 NEXT
  150 END
  160
  170 DEFPROCSQUARE
  180 MOVE 0,0
  190 DRAW 200,0
  200 DRAW 200,200
  210 DRAW 0,200
  220 DRAW 0,0
  230 ENDPROC
```

(2)

```
  10 MODE1
  20
  30 PROCGRAF                    This displays the graph.
  40
  50 PRINT"Here the graph takes up little"
  60 PRINT"of the screen. Using the *SCALE"
  70 PRINT"command and replotting the graph"
  80 PRINT"we can make it fill the whole screen."
  90 PRINT"(Press key to cont.)"
 100 G=GET
 110
 120                            Change the scale so the screen ranges from
 130 *SCALE 0,0,230,400         (0,0) in the bottom left to (230,400)
 140                            in the top right corner.
 150
 160 PROCGRAF                   Plot same graph with new scale.
 170
 180 PRINT"The graph now fills the screen"
 190 PRINT"The *SCALE command can also be used"
 200 PRINT"to make things smaller"
 210 PRINT"(Press key to cont.)"
 220 G=GET
 230
 240 *SCALE 0,0,2000,2000       Scale the screen to make the graph smaller.
 250
 260 PROCGRAF                   Plot the graph again with new scale.
 270 END
 280
 290 DEFPROCGRAF                This is the graph plotting routine.
 300 CLS
 310 MOVE 0,200
 320 FORX=0 TO 200 STEP4
 330 DRAW X,200+X*SINRAD(X)+X*.5*SINRAD(X*3)
 340 NEXT
 350 ENDPROC
```

***ROTATE** (<angle>) (<X,Y>)

When the rotate command is used all future PLOT, *PRINT, *ARC, *IN, *PATTERN and *PLOT commands will be affected. Anytime one of these commands is used after using a *ROTATE, the position of the item being plotted will be rotated about the screen co-ordinate (X,Y) by an angle given by <angle> in degrees. The angle represents a clockwise rotation.

NOTE: If a scaled screen is to be used as well as a rotated screen then the scaling should always be done first, since the position about which the rotation will occur must also be scaled.

The effects of *ROTATE can be stopped simply by entering *ROTATE with no argument.

Arguments:

(<angle>) OPTIONAL. The angle in degrees by which the graphical display will be rotated. If the argument is not given then no rotation will occur and none of the above commands will be affected.

(<X,Y>) OPTIONAL. The (X,Y) co-ordinate about which the rotation is to be made. If they are omitted then a rotation about the origin will be assumed.

Examples:

*ROTATE 45,640,512
This will rotate anything plotted on the screen by an angle of 45 degrees about the centre of the screen, assuming no scaling has been done.

*ROTATE 67
This will rotate all plotted points by an angle of 67 degrees about the origin.

*ROTATE
This will prevent any further plot commands from being rotated.

Example program:

The effects of a *ROTATE on a square and a point. NOTE: The point and the square are always plotted with the same co-ordinates. The *ROTATE makes it move round. Try removing the CLS.

```
  5 MODE 1
 10 FORA%=0 TO 3600 STEP 4
 20 CLS
 30 *ROTATE A%,640,512          Rotate all future PLOT etc. commands
 40                             about the centre by A% degrees
 50
 60 PROCSQUARE                  Plot the same square with the same
 70                             co-ordinates on the screen.
 80 PLOT 69,400,0               See how the screen co-ordinate 400,0
 90                             changes position with the rotation.
100 NEXT
110 DEFPROCSQUARE               Draws a square.
120 MOVE 540,412
130 DRAW 740,412
140 DRAW 740,612
150 DRAW 540,612
160 DRAW 540,412
170 ENDPROC
```

Patterns with *ROTATE.

```
 10 MODE1
 20 FORG%=200 TO 880
 30 IF N%=1 N%=2 ELSE N%=1       Alternate between red and yellow.
 40 GCOL 1,N%                    Use OR condition to get patterns.
 50 *ROTATE G%,G%,G%             Rotate the screen about G%,G% with
 55                              an angle G%!
 60 MOVE G%+100,G%+100           Draw a line. Try different numbers.
 70 DRAW G%-100,G%-100
 80 NEXT
```

***PIXEL** <X,Y> (<Xs,Ys>)

This command can be used to plot multi-sized pixels anywhere on the screen. It also works with the *ROTATE command in producing rotated pixels as appropriate.

Arguments:

<X,Y> The (X,Y) co-ordinate where the pixel is to be placed.

(<Xs,Ys>) OPTIONAL. These arguments represent the width and height of the pixel to be plotted respectively. If omitted then a normal sized pixel will be printed for the current screen scaling, e.g. if the screen is scaled (using the *SCALE command) from (0,0) to (20,20) then each pixel plotted by the *PIXEL command will be 1/20 of the width and height of the screen.

Examples:

*PIXEL 500,500,A%,B%
This will draw a pixel at position 500,500 with a width of A% and height B%.

*SCALE 0,0,20,20
*PIXEL 4,4
Will scale a screen so that there are only 20 by 20 pixels and put a correctly sized pixel (i.e. 1/20th the width and height of the screen) at position 4,4.

Program:
This program will draw random sized pixels at random orientations.

```
10 MODE 1
20 A%=RND(1280)              Choose random X position
30 B%=RND(1024)              Choose random Y position
40 C%=RND(200)               Choose random width
50 D%=RND(200)               Choose random height
60 E%=RND(360)               Choose random orientation
70 *ROTATE E%,A%,B%          Rotate pixel
80 *PIXEL A%,B%,C%,D%        Plot pixel
90 GOTO 20
```

***PRINT** <str> <X,Y> (<Xs,Ys>)

This command can be used to print multi-sized, multi-patterned and multi-coloured characters anywhere on the screen. The colour of the text printed will be the current graphics foreground colour as set by GCOL. The multi-colours and patterns (as seen on the box) use unusual GCOL statements (see User Guide page 262 for GCOL), e.g. GCOL 31,1. Many different effects can be achieved using random GCOL's (see example).

This command can also be used with *ROTATE to obtain slanted text or with *SCALE to produce scaled text. In some instances the text may be very large and take quite a while to print so the <ESCAPE> key may be pressed to stop further printing.

*PRINT will only work in graphics modes.
NOTE: Remember string variables such as A$, CAT$ etc. cannot be used. If variables must be used then see example (3).

Arguments:

<str> The string or text to be printed. If spaces or commas are to be included in the string then the whole string must be enclosed in quotes.

<X,Y> The (X,Y) co-ordinate position from where the text string is to be printed.

(<Xs,Ys>) OPTIONAL. This is the width and height of each pixel used to make up the character. Since each character is made up of 8 X 8 pixels then the actual size of the printed character will be 8 times the given X and Y sizes. If negative numbers are given for the size, then inverted and reflected text may be printed.

Examples:

```
MODE 2
GCOL 31,1
*PRINT HELLO,100,100,16,50
```

MODE 2

*ROTATE 48,640,512

Rotate the screen by 48 degrees about the point 640,512.

GCOL 0,3

Select yellow to print text.

*PRINT HELP,400,400,-16,16

Print the word 'HELP' at position 400,400 with a size of 16 by 16. The -16 means print it up-side-down.

Programs:

***PRINT demonstration.**

```
 40 MODE2
 50
 80 *PRINT "LARGE TEXT",0,850,16,16       Will print 'LARGE TEXT' at (0,850).
 90
110 GCOL 0,2                              Select green plotting colour.
120
150 *PRINT "ANY COLOUR",0,760,16,8        Text half as high as before.
160
190 GCOL 31,1                             Non-standard GCOL obtains patterns.
200 *PRINT " PATTERNS",0,600,16,16
210
240 GCOL 8,89                             Another pattern.
250 *PRINT "MULTIPLE SIZES",0,200,11,40   Print tall text in unusual pattern.
260
290 *ROTATE 355                           Puts following text at 5 degrees.
291
320 GCOL 30,89                            Yet another pattern using GCOL.
330 *PRINT "ANY ANGLE",100,0,16,16
```

Using negative sizes to produce inverted text.

```
  40 MODE1
  50 *PRINT HELLO 500,500,8,16        Normal.
  60 *PRINT HELLO 500,500,-8,16       Backwards.
  70 *PRINT HELLO 500,500,-8,-16      Backwards, up-side-down.
  80 *PRINT HELLO 500,500,8,-16       Up-side-down.
```

Using *PRINT with variables. When using variables it is essential to have the procedure from 110 lines to 160 and also line 10 at the beginning. The variable in this case is TEXT$ but anything can be used. The '0,0,16,16' is the size of the text. These too may be variables.

If you have BASIC II in your machine then the procedure at lines 110 to 160 may be replaced with the single line:

```
OSCLI("PRINT"""+TEXT$+""",0,0,16,16")
```

```
  10 DIM A 100                        Reserve space for variable.
  20
  30 MODE1
  40
  50 INPUT "TEXT :"T$                 Get text from input.
  60 CLS
  70 PROCPRINT(T$)                    Will print the contents of T$.
  80 GOTO50
  90
 100
 110 DEFPROCPRINT(TEXT$)              Procedure for using variables.
 120 $A="PRINT """+TEXT$+""",0,0,16,16"    Store command with variables at A
 130 X%=A MOD 256                     Set X% to lobyte.
 140 Y%=A DIV 256                     Set Y% to hibyte.
 150 CALL &FFF7                       Call OSCLI machine code routine.
 160 ENDPROC
```

***CIRCLE** <X,Y> <radius>

A command to draw circles. The colour for the circle will be the current graphics foreground colour as set by GCOL.

Arguments:

<X,Y> The (X,Y) co-ordinate where the centre of the circle is to be.

<radius> The radius of the circle.

Examples:

```
*CIRCLE 100,100,50
(Will draw a circle at (100,100) with radius of 50)
```

Programs:

(1)

```
  10 MODE 0                        Uses Exclusive Or to produce patterns
  20                               Try changing the STEP to 8.
  30 GCOL 3,1
  40 FOR R%=0 TO 840 STEP 4
  50                               640,512 is the centre of the circle.
  60 *CIRCLE 640,512,R%            R% is the radius.
  70 NEXT
  80 END
```

(2)

```
  10 MODE1                         Concentric circles.
  20
  30 FOR R%=0 TO 500 STEP 8
  40 GCOL 0,RND(3)
  50                               Draw a circle at centre of
  60 *CIRCLE 640,512,R%            screen with radius R% and random colour.
  70
  80 NEXT
  90 END
```

***PATTERN** <op> <X,Y> <Xs,Ys> (<step>)

This command enables multi-sized circles, spirographs or patterns to be drawn on the screen in a variety of colours. The patterns may be drawn with lines, dots or triangles and with any unusual GCOL's (see *PRINT). They need not be round but can be elongated and twisted using the *ROTATE command. The pattern will be drawn in the current graphics foreground colour as set by GCOL.

Arguments:

<op> The plot option. It enables either dots, lines or triangles to be plotted. (see User Guide page 319)

<X,Y> The centre co-ordinate of the pattern on the screen.

<Xs,Ys> The Xs is the width of the pattern. The Ys is the height of the pattern. This feature enables elongation or compression of patterns.

(<step>) OPTIONAL. This is the step taken around a circle in degrees to where the next point in the pattern is to be. If after going round the circle once, it does not meet up at zero then more of the pattern will be plotted until it does. This is how spirographs or solid rings are produced. If the argument is omitted then a step of 1 will be assumed.

Examples:

*PATTERN 5,640,512,400,400,58
Will draw a pattern with lines at co-ordinate 640,512 (centre of screen) with a width of 400 and height of 400. The step rotation is 58 and, since 360 is not exactly divisible by 58, then some form of spirograph will be plotted.

*ROTATE 6,300,300
Will rotate the pattern by 6 degrees about the point 300,300.

*PATTERN 85,300,300,100,100,120
Will draw a pattern in triangles at position 300,300 rotated by 6 degrees with a radius of 100. Since the step is one third of 360, this will draw a triangle.

Example programs:

(1)

Making complex patterns using *PATTERN in conjunction with *ROTATE.

```
 10 MODE 0
 20
 30 *SCALE -500,-500,500,500         Scale the screen with origin at centre.
 40
 50 FOR R%=0 TO 840 STEP 4
 60 *ROTATE R%                       Rotate the pattern by R% degrees.
 70
 80 *PATTERN 5,0,0,R%,R%,45          Draw a pattern with lines, at the
 90                                  origin with width and height of R%
100 NEXT                             and step of 45. Try another step.
```

(2)

Making multi-sided shapes using *PATTERN.

```
 10 MODE1
 20 FOR N%=3 TO 20                   Draw shapes with sides from 3 to 20.
 30
 40 S%=360/N%                        Get required step for amount of sides.
 50
 60 IF S%<>360/N% THEN 170           Do not draw patterns if uneven
 70                                  amount of sides.
 80
 90 PRINT"Step in *PATTERN = ";S%
100 PRINT"Amount  of sides = ";N%
110
120 *PATTERN 5,640,512,400,400,S%    Put the pattern in the centre of the
130                                  screen. With width and height of 400.
140                                  S% is the step of rotation.
150 G=GET                            Wait for key press.
160 CLS
170 NEXT
```

***ARC** <op> <X,Y> <Xs,Ys> <stt,end>

A command to draw arcs. These will be drawn in the current foreground colour and may be drawn in lines, dots or triangles specified by <op>. The arc is formed from part of an oval. The actual part of the oval to be drawn is defined by <stt,end> and the size and position of the oval are given by <X,Y> and <Xs,Ys>.

NOTE: Arcs will only work in graphics modes. The position and rotation of the arc will be affected by *SCALE and *ROTATE.

Arguments:

<op> This specifies whether the arc is to be drawn in dots, lines or triangles. The op codes are the same as for the BASIC PLOT command. (See User Guide on page 319). For most usual cases <op> will be 5 which draws lines.

<X,Y> The <X,Y> co-ordinate of the centre of an oval used to create the arc.

<Xs,Ys> The width and height of the oval from which the arc is to be formed.

<stt,end> The <stt> is the angle in degrees from the vertical where the arc is to start. The <end> is an angle in degrees where the arc will finish. This is more easily explained by examples.

Examples:

```
MODE 1
*ARC 5,500,500,400,300,90,180
```

This will draw in screen mode 1 a quarter of an oval from 90 degrees to 180 degrees from the vertical. The oval from which it was formed has its centre at 500,500 and is 400 long from the centre and 300 high from the centre. If the 90 and 180 are replaced by 0 and 359 you will see the whole oval from which it was formed.

Example programs:

Patterns created using *ARC. Try adding at line 75 *ROTATE H%,500,500.

```
  10 MODE1
  20
  30 FORH%=0 TO 500 STEP 8           Alter radius of curvature of arc.
  40 FORG%=0 TO 359 STEP 20          G% is the starting angle for the arc.
  50 GCOL0,G%/120+1                  Select plotting colour.
  60 A%=G%+10                        Find the end angle of the arc.
  70
  80 *ARC 5,640,512,H%,H%,G%,A%      Draw the arc from G% to A% from a
  90                                 circle of radius H% at (640,512).
 100 NEXT
 110 NEXT
```

Using *ARC in a simple pie chart program.

```
  10 MODE1
  20 FORG%=0 TO 359 STEP 20
  30 A%=G%+20                        Each segment is 20 degrees.
  40 *ARC 5,640,512,500,500,G%,A%    Put an arc in the screen centre
  50                                 with radius 500 from G% to A% degrees.
  60 DRAW 640,512                    Draw line to centre for sectors.
  70 NEXT
```

***FILL** <X,Y> <col>

This command may be used to fill any shape on the screen but the shape to be filled must have a closed border or the whole screen will be filled in. The fill routine will fill on top of one colour only, this being the colour of the pixel at the point (X,Y). All other colours are considered border colours. The shape will be filled with the colour set by <col>.

The *FILL will fill nearly every shape and even around text. However, since some exceedingly complex shapes require vast amounts of memory, occasionally the filling will stop. Since some very large areas may take quite a while to fill, pressing the <ESCAPE> key will stop the process.

NOTE 1: The *FILL command will not work in text modes.
NOTE 2: Due to the nature of the filling routine, memory must be used for a stack. This has been placed from &A00 to &AFF which means that any cassette or RS 423 files will almost definitely be corrupted. And so it is advisable not to use these during filling.

Arguments:

<X,Y> The (X,Y) co-ordinate of the starting point for filling. The colour of the pixel at this point determines the colour which will be filled over. All other colours will be considered as borders.

<col> The shape will be filled with this colour. If the point at <X,Y> is the same as colour <col> then this command will return immediately.

Example:

*FILL 34,56,2
Will fill from co-ordinate 34,56 in colour 2.

Example programs:

(1)
Filling a maze with *FILL. Try entering at line 45 *SCALE -400,-100,1400,2000 and *ROTATE -46 on line 46.

```
 10 T%=50                          T% is the density of the maze.
 20                                Try changing it to something else.
 30
 40 MODE 2
 50
 60 MOVE 0,0                       Draw the maze.
 70 DRAW 1279,0
 80 DRAW 1279,1023
 90 DRAW 0,1023
100 DRAW 0,0
110 FORG%=0 TO 1300-T% STEP T%
120 FORH%=0 TO 1054-T% STEP T%
130 IF RND(12)>6 THEN 160
140 MOVE G%,H%
150 DRAW G%+T%,H%
160 IF RND(12)>6 THEN 190
170 MOVE G%,H%
180 DRAW G%,H%+T%
190 NEXT
200 NEXT
210
220
230 *FILL 0,0,1                    Fill from the bottom left hand corner of
250                               the maze in red. This point will lie
260                               on the maze wall and thus the maze walls
270                               will change colour.
280 *FILL 504,504,2
290
300 END
```

(2)

Filling random shapes using *FILL.

```
 10 MODE2
 20
 30 COLOUR 7
 40 PRINTTAB(3,10)"Press <ESCAPE>"
 50 COLOUR 6
 60 PRINTTAB(3,15)"To end filling"
 70
 80 GCOL 0,1
 90 FOR N%=1 TO 40                  Draw 40 random lines.
100 DRAW RND(1400)-150,RND(1200)-150
110 NEXT
120
130 REPEAT
140 C%=RND(5)+1                     Choose random colour for filling.
150
160 A%=RND(1279)
170 B%=RND(1023)                    Choose a random start position.
180 *FILL A%,B%,C%                  Fill from (A%,B%) in colour C%.
190 UNTIL FALSE                     Repeat continuously!
200 END
```

***PLOT** <op> <X,Y,Z>

This is a very powerful command and enables complicated 3-dimensional structures and patterns to be drawn easily. (see examples).

NOTE: For most applications the origin should be in the centre of the screen.

Arguments:

<op> specifies the PLOT option (see page 319 in User Guide for PLOT options). Lines=5, dots=69 or triangles=85.

<X,Y,Z> specifies a point in 3-dimensions, X represents how far across the screen, Y how far up the screen and Z how far into the screen. NOTE: Z must be greater than 1 as numbers less than this would be behind or directly in the user's eye. Z will usually range from about 50 to 10000.

Examples:

```
*PLOT 69,100,200,1024
Will plot a dot at 3D co-ordinate (100,200,1024).
*PLOT 5,200,300,2000
Will draw a line to 3D co-ordinate (200,300,2000)
```

Example program:

The moving of a cube around the screen in 3-dimensions. i.e. move left, right, up, down and in and out of the screen.

```
  10 D%=0                          Set up variables.
  20 E%=0
  30 F%=600
  40 *FX 4,1                       Give values to cursor keys.
  50 MODE1
  60
  70 *SCALE -500,-500,500,500      Scale the screen and put origin
  80                               in the centre.
```

```
 90 REPEAT
110 G=GET                          Get control key.
120 IF G=136 D%=D%-100             If left arrow is pressed move left.
130 IF G=137 D%=D%+100             If right arrow is pressed move right.
140 IF G=138 E%=E%-100             If down arrow is pressed move down.
150 IF G=139 E%=E%+100             If up arrow is pressed move up.
160 IF G=73 F%=F%+100              If 'I' is pressed move into screen.
170 IF G=79 F%=F%-100              If 'O' is pressed move out of screen.
180
190 O%=4                           Plot in the cube.
200 RESTORE
210 FORN%=1 TO 16
220
230 READ A%,B%,C%                  Read data for cube from table.
240
250 X%=A%+D%                       Add offset position to data in all 3
260 Y%=B%+E%                       dimensions
270 Z%=C%+F%
280
290 *PLOT O%,X%,Y%,Z%              Plot with plot option O% at (X%,Y%,Z%)
300 O%=5
310 NEXT
320 UNTILO                         Repeat continuously.
330
340 DATA 0,0,0                     Data for the cube
350 DATA 100,0,0
360 DATA 100,100,0
370 DATA 0,100,0
380 DATA 0,0,0
390 DATA 0,0,100
400 DATA 100,0,100
410 DATA 100,0,0
420 DATA 100,0,100
430 DATA 100,100,100
440 DATA 100,100,0
450 DATA 100,100,100
460 DATA 0,100,100
470 DATA 0,100,0
480 DATA 0,100,100
490 DATA 0,0,100
```

***GFX** <fn> (<arg>) (<arg>)

A Graphics Effects (GFX) command. This provides the user with various pieces of information concerning the several graphical routines but, unlike the *DATA command, it stores the results in the resident integer variables A%,B%,C%,D% and E% so that they can be used within BASIC programs. Information can be requested about memory usage, ID numbers, SPRITES, FILMS and the logo turtle. Two of the GFX commands provide the user with fast integer SIN and COS routines. Another performs a POINT command with scaled and rotated screens and returns the actual screen position. GFX5 can be used to enable or disable the definition of user definable characters.

Arguments:

<fn> The function to be performed.

<arg> OPTIONAL. A general purpose argument. If it is omitted then 0 will be assumed.

<arg> OPTIONAL. Same as above.

FUNCTION VALUES

fn=0 - Returns information about SPRITES/FILMS
Details are returned about the ID number given by the first <arg>.
A%=ID type. (A%=0 if not used. A%=1 if a FILM. A%=2 if a SPRITE.)
If the ID number has not been used then B%,C%,D% and E% will equal zero.
If the ID number corresponds to a SPRITE then-
B%=SPRITE condition. (B%=1 if SPRITE is on screen otherwise B%=0.)
C%=Width of SPRITE. i.e. <Xs>
D%=Height of SPRITE. i.e. <Ys>
E%=Graphics MODE in which SPRITE was designed.
If the ID number corresponds to a FILM then-
B%=FILM condition. (B%=1 if FILM is on screen otherwise B%=0.)
C%=Number of frames in the FILM.
D%=The current frame being shown in the FILM.
E%=0

fn=1 - Memory status

A%=Bottom memory address of reserved block.

B%=Current memory address in the reserved block.

C%=Amount of memory left.

D%=Address of SPRITES or FILM with an ID given by 1st <arg>.

E%=Amount of ID's used.

fn=2 - Turtle status

A%=X co-ordinate of turtle.

B%=Y co-ordinate of turtle.

C%=Current angle of turtle. (This is measured anticlockwise from the horizontal, i.e. 90° points up the screen)

D%=Turtle condition. (Will be 1 if a turtle is being used otherwise 0)

E%=Turtle plot option. (Will be 4 after a *PENUP.)

fn=3 - Fast SIN function

A%=2nd<arg> * SIN(1st<arg>) in degrees.

B%,C%,D% and E% are not altered.

fn=4 - Fast COS function

B%=2nd<arg> * COS(1st<arg>) in degrees.

A%,C%,D% and E% are not altered.

fn=5 - Enabling or disabling the definition of user definable characters. Since THE GRAPHICS ROM uses page &C (see memory map page 69) user definable graphics cannot normally be defined. However, a *GFX 5 command may be issued to re-enable user definable characters. In this case the definable character set would be stored in memory at the current OSHWM, i.e. the current setting of PAGE. This command also sets PAGE to be 256 bytes higher, but this will only take effect after a BREAK or *BASIC.

*GFX5,128 will disable user definable character definition and will set PAGE back to normal (after a *BASIC).

When the GRAPHICS ROM is first enabled (*FX162), characters may not be defined and all user definable characters will be mapped onto the normal ASCII character set. This means that while the ROM is enabled, programs using definable characters will not work until the *GFX 5 command is issued followed by BREAK or *BASIC. Continued..

It is possible to use exploded character definitions (see User Guide page, 427). *GFX5,x may be issued where x is the explosion state required. Alternatively *FX20 may still be used for this exploding or imploding, but this must follow a *GFX5 command. As usual pressing BREAK will implode the character set.
When this command is used the resident integer variables A% to E% will not be altered.

fn=6 - Converts the given (X,Y) co-ordinate at (<arg>,<arg>) before scaling and/or rotating and returns the actual screen co-ordinate and the colour of the pixel at that co-ordinate. If no arguments are given then the current graphics cursor position (after scaling/rotating) will be returned along with the colour at that point.
A%=Actual X co-ordinate.
B%=Actual Y co-ordinate.
C%=POINT(X,Y).
D% and E% are not altered.

Examples:

*GFX 0,12
Will provide the user with all the necessary information on ID number 12.

*GFX 1,12
Will provide information on memory usage and also the memory address where the FILM or SPRITE with ID 12 is situated.

*GFX 2
Will provide the user with the current position, angle, condition and plotting option of the turtle.

*GFX 3,67,145
*GFX 4,88,222
Will return in A% and B% the results of the functions SIN(67)*145 and COS(88)*222 respectively.

*GFX 6,1000,200
(Will return in A% and B% the actual screen co-ordinates before rotation and/or scaling of the co-ordinate (1000,200). The colour of the pixel at this point will be returned in C%.)

MODE 8

This is an extra screen mode. Normally screen modes range from 0 to 7. However when THE GRAPHICS EXTENSION ROM is in the machine, an eighth screen graphics mode exists. This is entered using the BASIC MODE command as MODE 8 or using a VDU 22,8. MODE 8 is a 16 colour text and graphics mode like MODE 2 but, whereas MODE 2 uses 20K of memory, MODE 8 only uses 10K thus leaving 10K bytes extra for programs but still giving 16 colours.

The horizontal resolution of MODE 8 is half the normal MODE 2, i.e. 10 by 32 characters, and 80 by 256 graphics.

MODE 8 will work perfectly for most applications but care must be taken when defining screen windows and any lines or triangles plotted must not go off the screen or wrap-round will occur. The POINT command will not always return -1 if the point being tested is off the screen.

If *FILL is used in this mode care must be taken to ensure that the shape being filled is not bounded by the screen edge.

Example:

```
MODE 8
```

Example program on the next page.

MODE 8 demonstration program.

```
 10 MODE 8                          Enter graphics mode 8
 20
 30 PRINT "16 colours"              Print in 16 colours 10 by 32 text
 40 PRINT '" Only 10K"
 50 FOR G%=0 TO 15
 60 COLOUR G%
 70 PRINTTAB(1,G%+9)"Colour ";G%
 80 NEXT
 90
100 COLOUR 7
110 PRINTTAB(0,31)"Press key";
120
130 G=GET                           Wait for key press
140
150 CLS
160 PRINT"Graphics"                 More wide character printing
170 PRINT'"80 by 256"
180
190 FOR N%=1 TO 12                  Make 12 patterns
200 GCOL 3,RND(7)                   Set random foreground colour.
210 A%=RND(20)*4                    Choose random step for *PATTERN
220 B%=RND(10)*40+50                Choose random radius for *PATTERN
230 *PATTERN 85,640,450,B%,B%,A%    Make a pattern at screen centre
240 NEXT                            with triangles. Radius B%. Step A%
250 END
```

BBC Memory map with GRAPHICS EXTENSION ROM fitted.

The total memory in the BBC is 64K or 65536 memory locations or bytes. Each of these 65536 bytes has its own unique address number ranging from 0 to 65535. The memory in the BBC is split into pages, each page consists of 256 bytes and there are 256 pages. The addresses in memory are not usually specified in decimal but, and more conveniently in hexadecimal. In Hex the memory addresses range from 0 to &FFFF. Each page is &100 locations long.

BASIC programs are stored from the value of PAGE onwards. (See User Guide page 317 for PAGE.)

The memory arrangements in the BBC are split into pages as follows:-

0 - &FF	ZERO PAGE
&100 - &1FF	6502 stack.
&200 - &8FF	Operating system and language workspace.
&900 - &AFF	Cassette and RS 423 workspace.
&B00 - &BFF	Function key definitions.
&C00 - &CFF	The Graphics Extension ROM workspace. (See next page.)
&D00 - &DFF	Operating system usage.
&E00	Default setting of page with tape filing system, unless a *GFX 5 is used.
&F00	PAGE setting after a *GFX 5.
&1900	Default setting of PAGE with disk filing system, unless a *GFX 5 is used.
&1A00	PAGE setting with disk after a *GFX5.
&1A00 - &2FFF	Always free for user programs and SPRITE definitions.
&3000 - &7FFF	Memory used by the screen in modes 0,1 and 2.
&5800 - &7FFF	Memory used by the screen in modes 4 and 5.
&8000 - &BFFF	Usually BASIC language and paged ROMs such as THE GRAPHICS EXTENSION ROM.
&C000 - &FFFF	Operating System ROM.

Detailed Memory Map of THE GRAPHICS EXTENSION ROM workspace.

&60 - &8F Used when a GRAPHICS ROM command is used. These locations will, under normal conditions, be saved and restored before and after the command so that the user's machine code programs will still work.

&60 - &6F When using PLOT commands with a scaled or rotated screen, these locations will be corrupted. Thus the user is advised not to use these locations in machine code programs.

&100 - &140 Used for storing error messages and temporary workspace.

&A00 - &AFF Used for storage of strings in *PRINT, *GET and *PUT. This whole area will also be used as workspace for the *FILL routine. The user is thus advised not to use cassette files or RS 423 while these commands are in use.

&C00 - &CFF The GRAPHICS EXTENSION ROM workspace area.

This area of memory is normally assigned to user definable graphics characters. However when GRAPHICS ROM is in use, character definitions are not allowed in this area and therefore it is not normally possible to define characters. There are two ways of overcoming this problem.

1 - Using a *GFX 5 and entering *BASIC. When this is done memory will be reserved elsewhere (at the old value of PAGE) for the user definable characters so they may now be defined as normal.

2 - Disabling the ROM totally, with *FX162,128. This has a similar effect to removing the ROM from the machine. (See page 72).

If any other ROM in the system has a command with the same name as one of the GRAPHICS ROM commands then the *FX162,128 may be used to temporarily disable the GRAPHICS ROM. When the command is issued it will now "pass over" the GRAPHICS ROM to the other ROM where it would be interpreted as normal. *FX162 would then be used to enable the GRAPHICS ROM again.

&C00 - &CBF Information about all 32 ID's. This area should never be touched. Use the *GFX commands to obtain required information.

&CC0,&CC1 The current position in the reserved block of memory.

&CC2,&CC3 The end address of the block.

&CC4,&CC5 The beginning address of the block.

&CC6 Amount of bytes to go in the intercepted VDU queue.

&CCB The top bit is set if user definable characters are to be allowed.

&CCE The top bit is set if the turtle is on the screen. Bit 6 is set if a turtle is being used.

&CCF The colour of the turtle.

&CD0 The plot option for the turtle.

&CD1,&CD2 The width of the turtle.

&CD3,&CD4 The height of the turtle.

&CD6 - &CDB Temporary store for intercepted PLOT arguments.

&CDD Amount of SPRITES that have been defined.

&CDF The ROM number of the GRAPHICS EXTENSION ROM.

&CE0,&CE1 X factor for screen scaling.

&CE2,&CE3 Y factor for screen scaling.

&CEF Store for turtle plot option after *PENUP.

&CF2,&CF3 The angle of rotation set by *ROTATE.

&CF4,&CF5 The X co-ordinate about which rotation takes place.

&CF6,&CF7 The Y co-ordinate about which rotation takes place.

&CF8,&CF9 The X co-ordinate for the turtle position.

&CFA,&CFB The Y co-ordinate for the turtle position.

&CFC The X co-ordinate fraction byte. i.e. divide by 256 and add to X co-ordinate.

&CFD The Y co-ordinate fraction byte.

&CFE,&CFF The current angle of the turtle in internal co-ordinates.

Problems which may occur as a result of THE GRAPHICS ROM being in the machine.

If the machine will not turn on properly. i.e. it jams, or if none or few of the graphics commands appear to work, then check the following-

1) Make sure all the links in your machine have been changed correctly as specified by the fitting instructions.
2) Make sure the ROM is inserted the correct way round, i.e. notch facing north and that all the legs of the ROM are correctly in the socket and that none have been left out.
3) Make sure nothing has been left in the machine such as a screw-driver etc.
4) It is very unlikely that the ROM is faulty. However if there are still problems contact the dealer from whom you purchased the ROM.

If the GRAPHICS ROM is in the machine and a program or game does not work as it would without the ROM, it may be connected with-

1) The program uses definable characters. As explained before, this is not normally allowed since the GRAPHICS ROM uses this area.
2) The program uses locations &C00 to &CFF illegally. This may happen with some games.

Both of these problems can be solved by disabling the ROM. This has the same effect as removing the ROM from the machine. The command to disable and re-enable the ROM is *FX 162.

TO ENABLE THE GRAPHICS ROM ENTER -

```
*FX 162
```

TO DISABLE IT AFTER BEING ENABLED ENTER -

```
*FX 162,128
```

NOTE: This may be used within a user's program to temporarily stop the screen from being scaled or rotated. Another ROM in the machine may have an identical command to that of the GRAPHICS ROM and if you wished to use that particular command you could disable the ROM, use the other command, and re-enable it again.

It may be that you wish to have user definable characters and still use the GRAPHICS ROM commands. In this case use a *GFX 5 and type *BASIC (see page 65). This will increase the value of PAGE by &100.