IMAGEM Graphic Video

Programming Manual

Imagem Graphic Video

Programming Manual

Publication No. T390

IMAGEM GRAPHIC VIDEO PROGRAMMING MANUAL

CONTENTS

CEGELEC INDUSTRIAL CONTROLS Ltd,
Kidsgrove, Stoke-on-Trent, Staffordshire ST7 1TW, England.

Telephone: Stoke-on-Trent (0782) 783511 Telex: 36293/4 CICKID G

(International 44 782 783511) Fax (Group 3): Stoke-on-Trent (0782) 776 329

Holding Company: CEGELEC CONTROLS Ltd.

T390 Issue 5 03.92

INTRODUCTION AND OVERVIEW

This page Left Intentionally blank

INTRODUCTION

This manual defines how an IMAGEM Graphic Video system can be programmed to produce the required displays.

The IMAGEM Graphic Video system consists of a single module. One or more modules can be plugged into the central highway of a suitable GEM80 Controller (such as a GEM80.310 series). The Graphic Video system accesses the controller's data tables both for data for display and for control information which determines the required display. The display is specified by user-defined formats which contain textual, numeric, presentation and control information. Up to four formats can be displayed on each IMAGEM video system independently of any displayed on other IMAGEM video systems in the same subrack. The formats themselves are held in a battery-supported memory or EPROM on the central highway, and are common to all IMAGEM video systems in that subrack. The system is designed to be self-contained, and needs no support other than a standard GEM80 programming unit (either a Portable Programmer or a System Programmer).

An alternative programming system is available, known as GEMESYS, which comprises a software disc and a cable to connect a p.c. to the host controller.

This manual describes, in detail:

- The language specifying a format.
- The operation of the Format editor used to enter this format into the system.
- The operation of the Character editor, which allows the in-built default character set to be extended or modified.
- Support facilities for archiving (tape/disk load.dump.verify).
- Documentation (format and character set print-out).
- Internal administration (system set-up and format copy.delete).
- The background and user-initiated test facilities.

OVERVIEW

Pixels

The display area of a Video Monitor screen when used with an IMAGEM Graphic Video system is divided into over a quarter of a million picture cells, known as ***pixels***. These are arranged in an array 560 pixels wide by 448 pixels high.

To reference a particular pixel, its X and Y co-ordinates must be given. The origin is normally the pixel at the bottom left-hand corner of the screen with co-ordinates 0,0. The co-ordinates of the pixel at the top right-hand corner of the display area are therefore 559,447. However, reference can be made to notional pixels which are outside the screen area using co-ordinates in the range up to -32 768 to 32 767.

Colours

Colours can be set up to be any one of 4096 possible combinations of shades of the primary colours red, green and blue. Colours may also be set up to flash, or to run through a repetitive sequence. On any single display, up to 16 colour definitions may be used at the same time. These colour definitions can also be varied dynamically by the program running in the controller.

Formats

Any picture displayed on the Video Monitor screen can be built up from up to four ***formats***. When programming the definition of a format, details which are outside the screen area can also be included. By using a WINDOW command, a particular rectangular area can be selected from what has been defined to appear on the Monitor screen.

When displaying more than one format on the screen at once which happen to overlap, any format called by a higher L-table number will overwrite any format called by a lower L-table number where it overlaps. Refer to Section 3, System Interface.

If the formats being displayed at any particular time do not fill the complete area of the screen, the unfilled area continues to display whatever was previously called up. This could, when first using the system, be part of the GEM80 default format.

Using easily remembered English keywords, the IMAGEM Display Language (see Section 4) allows you to define formats. Apart from instructions affecting the presentation (e.g. sizes and colours), the formats normally consist of text (including both letters and numbers), and graphic constructions.

Characters

Text can be constructed from up to 224 different types of character. Of these, 96 are already set up to the ASCII (ISO-7) set, and the remaining 128 are blank. The character editor (See Section 7) can be used to add personal special characters and, if required, to re-define any of the standard set.

The SIZE command allows you to display any of the 224 characters available in enlarged sizes. For SIZE 1 (the default condition), characters are made up from an array of pixels, 7 wide by 14 high. Using the SIZE command, they can be displayed several times as large. If desired, the size in the X and Y directions can be increased independently.

Graphic constructs

Using the DRAW command, straight lines, circles, and arcs of circles can be drawn. If a closed outline of some special shape is created, the FILL command can be used to colour it, cross hatch it, etc.

Functions

Where a particular shape or symbol occurs several times in a format, or is repeated in different formats, instead of writing out the same section of program several times, it can be defined as a ***function***.

Any function set up must have a name. For example, a function called VALVE could be defined to draw a valve symbol; a function called HOPPER to draw a hopper symbol; and so on. Pre-defined functions included in the language are TRIANGLE and RECTANGLE.

Control

Conditional operators can be used in format and function definitions. For example, using the IF..THEN..ELSE construct, the colour of a bar in a bar chart can be arranged to change when a number exceeds a given value, or change the colour of a valve depending on whether it is turned on or off.

Mathematics

As well as allowing simple decision making, IMAGEM Display Language allows you to perform mathematics. Values taken from the main GEM80 Controller's data tables can therefore be suitably manipulated before being displayed, or manipulated values can be used in decisions affecting the presentation of the display.

In the same way that functions can be written for repeated graphical constructions, mathematical functions can also be written for calculations which otherwise would have to be written out several times. Pre-defined functions included in the language are SIN and COS. If necessary, functions can be defined which include both mathematics and graphics constructions.

Other features

The SCALE command allows the overall scale of formats to be altered, so that zooming in can be effected to enlarge particular areas of the display. SCALE interacts with the SIZE command for text, so that text can be made invisible at reduced scale, but visible when the scale is increased to show more detail.

By re-defining which pixel is the origin of a format with the ORIGIN command, the display can be shifted so that something that was previously off the edge of the screen can be made to appear.

This page left intentionally blank

GETTING STARTED

This page left intentionally blank

As this manual is about programming, it is assumed that the IMAGEM Video system has been correctly installed and connected to a suitable colour Monitor. It is also assumed that Video number 0 (zero) is to be used. If this is not so, references to L0 should be interpreted as L4, L8 or L12 corresponding to Videos 1, 2 or 3.

On power up, a default format consisting of GEM80 and a rotating coloured vector should be displayed. If this is not so, a hardware or installation fault is indicated and this must be corrected before any further progress can be made. In addition, the default format displays the Video number, which should match the Video in use, in the bottom left-hand corner.

If the display appears correctly but rolls up or down, it is an indication that the field rate expected by the monitor is incompatible with that being generated by the IMAGEM video. Check the compatibility of the monitor. (Field rate 75Hz)

After a few seconds, the video system will attempt to display the formats specified in the controller locations L0 to L3. This can produce one of three effects:

a) The default format continues to be displayed
This indicates that L0 to L3 are all zero.

b) A display appears on the Monitor
This indicates that a valid format has been specified in L0 to L3.

c) INVALID FORMAT flashes on the Monitor
This indicates that L0 to L3 contain the number of one or more formats to which the video can find no reference.

In the case of b) or c), connect the Programmer to the controller, halting it if necessary, and write zero to L0 to L3. Condition a) should now apply.

Now connect the Programmer to the front panel port of video number zero and initiate the display of the video main menu. To do this, follow the screen prompts to get to the Programmer's initial menu, if it is not already there, then press:

The space bar and the '7' key on a Portable Programmer
The '2' and '4' keys successively on a System Programmer.

The IMAGEM Graphic Video main menu, as shown below, should be displayed on the Programmer:-

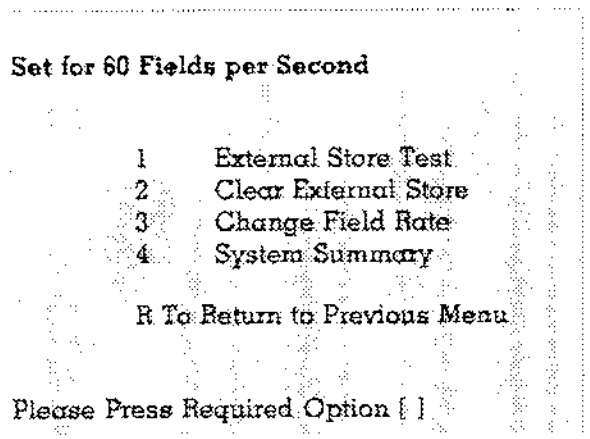
```
IMAGEM - GEM80 Graphic Video - Main menu

        1    Self Test and Set Up
        2    Edit Format or Function
        3    Edit Character Set
        4    Copy Format or Function
        5    Delete Format or Function
        6    Print Character Set, Format or Function
        7    Load from Tape or Disk
        8    Dump to Tape or Disk
        9    Verify Tape or Disk

        R To Return to Previous Menu

Please Press Required Option [ ]
```

This offers a choice of nine options. Press key '1' to select the 'Self-test and Set-up' option. A further menu is displayed as follows:-



Note...This menu displays at the top, the field rate for which the Video system is currently set (60 fields per second in the above example). This message is for compatibility with earlier Imagem systems and does not have any specific meaning. The field rate is set at 75Hz and cannot be changed by the user.

Pressing '4' (System Summary) will display a list of the formats stored in the system. This list could:

a) be empty
b) have a number of formats listed
c) have many error messages displayed, or the video monitor could display a solid magenta screen.

In the event of (c), the external store has not been configured for use with Graphic Video. To do this, press the space bar. This will display the previous menu, from which option '2' (Clear External Store) should be selected. The system asks for confirmation of the choice, which is done by pressing the 'Y' key. Verify that the store has been cleared successfully by pressing '4' to again display the list of stored formats, which should now be empty.

From the list of stored formats, mentally select an unused format number (in the range 1 to 32 767) and then press the space bar, to terminate the summary listing, followed by 'R' to Return to the Video main menu.

The system is ready to accept a trial format. To do this, press '2' (Edit Format or Function). The monitor display clears and is replaced with a simple IMAGEM message. The next prompt asks, via the Programmer, for a format number or function name to be entered.

Note...Requests to 'Enter this or that' are made in many places in the Graphic Video system. To do this, type in the answer terminated by a 'Carriage-return' (CR) or 'Line-feed' (LF). Prior to either of these terminating characters, characters in error can be deleted by using the 'Rub-out' key.

Type in the mentally selected format number, followed by CR or LF. This should result in the message "NEW FILE" followed by the Editor prompting character (>).
Now type in:

```
I
FOREGROUND RED
BACKGROUND CYAN
WINDOW 0,0,559,447
SIZE 2
"HELLO WORLD"
```

Terminate this with a double press of the 'Escape' (ESC) key (which will be shown on the screen as two '$' characters). A further prompting character should be displayed on the Programmer screen, and 'HELLO WORLD' should appear on the Monitor, together with a cursor made of two crossing dotted lines positioned at the end of the text.

Note...The cursor is not part of the format, but is generated by the Format editor as an aid to editing more complex formats.

This format can now be stored for future use by pressing the 'E' key, followed by a double press of the ESC key. The display now returns to the Video main menu on the Programmer, and the initial default format is displayed on the monitor. Pressing 'R' terminates the video editing session and returns to the Programmer's own menus. By moving the Programmer s lead to the controller's front panel port, it should now be possible to display the format by writing its number to L0, and clear it again by writing zero to L0.

Further trial formats can be generated in a similar way by entering:

- different text strings
- different values (in the range 1 to 15) following the SIZE command
- different colours from the default set (BLACK, RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, WHITE or GREY).

If problems are encountered while within the editor then, in response to the prompting character, press the 'Q' key, followed by a double press of the ESC key. Confirmation of Quit action will be requested. A response of 'Y' returns the display to the video main menu. If the menu structure becomes difficult to follow, a specific choice is offered which enables a return to the next higher level menu, or the ESC key can be used to abort any 'Enter this or that' message.

It should not be necessary but, as a last resort, the programming lead can be disconnected. The IMAGEM Graphic Video system monitors the activity on the front panel port, and automatically resets itself if disconnected from the Programmer for more than one second. By re-inserting the programming lead and following the Programmer prompts, it is possible re-enter the video main menu.

This page left intentionally blank

SYSTEM INTERFACE

This page left intentionally blank

MEMORY

Video formats are stored in battery-supported memory or EPROM modules on the central highway. This memory will be referred to as the *external store*. When a format is called up for display or for editing, it is copied from external store into memory on IMAGEM, this on-board memory being referred to as the *internal store*.

The external store is divided into separate *pages*, each capable of storing 16K bytes. Anything from 1 up to 34 pages will be available depending on the particular controller being used. When a format is called up, it is copied from the external store into the internal store within the IMAGEM video.

CALLING UP FORMATS FOR DISPLAY

Controllers capable of supporting Video facilities contain an L-table whose size matches the number of separate IMAGEM modules in the system, with four locations allocated to each module as follows:-

Video number	First format	Second format	Third format	Fourth format
0	L0	L1	L2	L3
1	L4	L5	L6	L7
2	L8	L9	L10	L11
3	L12	L13	L14	L15

To display a format, its number should be placed in one of the L-table locations allocated to the video on which the format is to be displayed.

Up to four formats can be requested for simultaneous display by one video. These formats may or may not overlap, depending on their sizes. If more than one format is called up, the one requested in L0 (or L4, L8 or L12 for Videos 1, 2 or 3) will be built up first, overlayed by that requested in L1 (or L5, etc.), further overlayed by that in L2 and then by that in L3. If any L-table location contains zero, it will be ignored. Only if all four locations associated with a given video contain zero is the default format displayed.

HOW DISPLAYS ARE BUILT UP

The IMAGEM Video system contains two screen buffers. One is used for the current display output, while the other is used for building up the next display. Once all formats for a given Video have been processed, then the two buffers are interchanged in the next interval between screen refreshes, thus giving a clean picture update.

Whenever the two buffers are interchanged, the IMAGEM Video system increments the contents of L16, L20, L24 or L28 (corresponding to Videos 0, 1, 2 or 3 respectively), to enable the controller to monitor the activity of the video. In addition, the state of the 'Active' light on the front panel of the IMAGEM module is changed, giving a visual indication of the display update rate.

To build up the next display, the Video system works as follows:-

First, it reads the lowest L-table location for the particular video (e.g. L0 for Video number 0) to find the first format number required. It copies the format definition of this number from external store into internal store, and processes it. Working through this definition from beginning to end, IMAGEM sets up data in the buffer memory for the colour of every pixel referenced by this definition. The data for any pixels that are unaffected by the format definition remain unchanged.

Next, the process is repeated for the second, third and fourth formats for the particular video.

Consequently, while a display is being built up, the data for the colour of any pixel can be altered several times, as the data may be overwritten by:

- Later commands in any format definition overwriting data set up by earlier commands.
- Later format definitions overwriting data set up by earlier format definitions.

The fact that, while format definitions are being processed, pixel data is being altered several times makes no difference to the display. Only when the last format definition has been processed and the whole of the display has been built up are the buffers interchanged. The pixel colour data used for the display on the video monitor therefore only consists of the final versions of data after all processing is complete.

PICTURE FREEZE

Picture updating can be frozen by writing -1 to L0, L4, L8 or L12 as appropriate. When this is done, options 2 to 9 inclusive cannot be selected from the IMAGEM video main menu. To re-request updating, a positive value must be written to the appropriate L-table location. When this is done, the contents of L16, L20, L24 or L28 as appropriate will be re-set to zero.

SELF-TEST

On the IMAGEM module front panel, a second light (the 'O.K.' light) is normally illuminated. If the internal self-test routines detect a problem, a fault number is written to the Video's F1 location, and the light is extinguished. For further details, see Section 11, Test Facilities.

IMAGEM DISPLAY LANGUAGE INTRODUCTION

This page left intentionally blank

OVERVIEW

This Section describes the IMAGEM Display Language. It gives various examples of programming format and function definitions, and introduces the various commands.

Once some of the examples have been tried and a little experience gained in programming the IMAGEM Graphic Video system from reading this section, Section 5 will be found useful. Section 5 is a reference section, giving details of every keyword in the Display Language, along with notes on topics such as Separators', arranged in alphabetical order for easy access.

INTRODUCTION

Look again at the simple example given in Section 2, "Getting Started".

```
I
FOREGROUND RED
BACKGROUND CYAN
WINDOW 0,0,559,447
SIZE 2
"HELLO WORLD"
```

Before this could be keyed in, Option 2, Edit Format or Function, had to be selected from the Graphic Video menu to select the Format Editor. Full details of all the Format Editor commands are given in Section 6 of this manual. The 'I' in the first line is one of these commands. It tells the Format Editor that what follows will be Display Language commands which are to be Inserted into the memory on the IMAGEM module. This is referred to as 'internal store' so as not to confuse it with the memory on the separate RAM memory modules in the subrack that provide the 'external store'. See Section 3.

After the Insert command to the Format Editor, the next five lines in the example consist of Display Language commands.

Note...Every line keyed in must terminate with a Line Feed (LF) or Carriage Return (CR). It is assumed that each line is terminated in this way, and the LF or CR keystrokes will not be shown in any of the examples.

Each of the first four lines of program consists of a ***keyword*** followed by a name or some figures. The keywords in this example are therefore the words FOREGROUND, BACKGROUND, WINDOW and SIZE. These words can be keyed in as either upper case (capital letters) or lower case (small letters). However, in this section they will always be shown in upper case so that they stand out from any descriptions of their use.

The first line of program is a FOREGROUND command. This sets the colour of any characters, lines or other graphic figures that commands later on in the program call up. In this example, these will be RED, one of the named colours in the default set. A list of the names of the default set of colours is given in Section 5 under the COLOUR command.

The second line of program is a BACKGROUND command. This sets the background colour of the window defined in the next line as CYAN which is the name of another colour out of the default set of colours.

The third command uses the WINDOW keyword. This command sets the size of the area of the screen into which the format is to fit. Positions on the screen are defined by X,Y co-ordinates. The bottom left-hand corner is the origin with co-ordinates 0,0, and the top right-hand corner has co-ordinates 559,447. With a WINDOW command, the co-ordinates given must be of two diagonally opposite corners of the window area (which is always rectangular). In the example, this line of program sets the size of the format to be equal to the whole area of the screen.

The fourth line of program is a SIZE command. SIZE 2 means that any characters displayed will be of double size, i.e. double height and double width.

The fifth and last line of this short format definition has no keyword. It just consists of text within quotes. Any characters appearing in the program within quotes are displayed on the video monitor. As there is no command to tell the system whereabouts it must appear, the text in this instance will appear in the top left-hand corner of the screen.

Having completed this simple format definition, the Format Editor must be told that command insertion is complete. This is done pressing the 'Escape' (ESC) key twice. The result of the format definition is now displayed on the monitor screen. Also, another editor prompt symbol ('>') is displayed on the Programmer, showing that it is ready to accept further Format Editor commands.

KEYING IN FORMAT AND FUNCTION DEFINITIONS

In general, long keywords can be abbreviated to the first three characters. Exceptions are the keywords FOREGROUND and GETCHAR, which must not be shortened to less than FORE and GETC because the Display Language also contains separate keywords FOR and GET. However, if a format definition that has been saved in external store is subsequently recalled, it will be displayed with the keywords spelt out in full.

More than one command can be keyed in on a line. However, on recalling a saved format definition, it will be displayed with one command per line. Therefore the previous example could have been keyed in as follows:-

```
IFORE RED BAC CYAN WIN 0,0,559,447 SIZ 2 "HELLO WORLD"
```

but when recalled it would give a display on the Programmer as listed previously (except, of course, for the leading I, as this was a format editor command and not part of the format definition).

CURSOR MOVEMENT

In the example, the text enclosed in quotes in the program was displayed in the top left-hand corner of the video monitor screen. Unless instructions in the program are given to the contrary, the system always starts any text in this position.

However, the writing position can be moved on the screen with the MOVE command. The MOVE command requires X,Y co-ordinates to place the current writing position wherever it is required on the screen. This X,Y position corresponds to the bottom left-hand pixel of the first character. If the program is modified as below:-

```
FOREGROUND RED
BACKGROUND CYAN
WINDOW 0 0 559 447
MOVE 205 210
SIZE 2
"HELLO WORLD"
```

the 'HELLO WORLD' text will appear approximately in the centre of the screen.

Note... In the previous example, commas were shown between the co-ordinates in the WINDOW command, whereas this time spaces are shown, and also a space between the X,Y co-ordinates of the MOVE command. In fact, the system accepts either and semi-colons can also be used. For more details, see the item on Separators in Section 5.

EDITING FORMATS

So far, only setting up a new format has been described. What it is obviously necessary to know now is how to modify an existing format, e.g. how to insert the extra command 'MOVE 205 210' into the original example format.

As this section is primarily concerned with the Display Language, editing will be covered only briefly. Full details of the Format Editor and the commands available are given in Section 6, which may be referred to as necessary.

The W, (Window Mode) command provides a more convenient mode of editing with System Programmers only. With the exception of this command all the format editor commands can be used in the same way for both Portable Programmers and System Programmers

With either type of programmer, select Option 2 (Edit Format or Function) from the video main menu. Next, key in the number of the format to be edited, followed as always by a CR or LF (carriage return or line feed). After this, the format editor prompt character ('>') is displayed.

EDITING WITH THE SYSTEM PROGRAMMER IN WINDOW MODE

Press W then press ESC (Escape) twice to enter the Window Mode. The format definition will be displayed on the Programmer screen. If it is required to insert an extra command, using the vertical and horizontal arrow keys, move the current writing position to the required position. Then simply type in the extra command. Use the RUB OUT key when correcting any errors. When inserting is complete, press ESC once.

If the insertion has been carried out correctly, the results of the revised format definition will be displayed on the Video Monitor screen If an error has been made so that the compiler cannot understand what has been keyed in, an error message is displayed on the programmer screen. In this case, press the space bar, continue editing, and then press ESC again. Once the format definition is displayed without any compiler error messages, press ESC again.

EDITING WITH THE PORTABLE PROGRAMMER

As mentioned above, editing can also be carried out in this way with a System Programmer, but the Window Mode of editing is generally more convenient.

Once the editor prompt character is displayed, the pointer must be set to the place in the program where an insertion is to be made. Usually, the quickest way to move the pointer is to tell the editor to find the last command, or section of command, immediately prior to where the insertion is to be made, using the F (Find) command.

Taking the original example, to insert the MOVE command, the editor command F447 could be used, followed by pressing ESC twice. This would tell the editor to find the characters '447' at the end of the WINDOW command (since 447 does not appear anywhere else in the program), and the pointer would be left at the end of the line. To insert the extra command, use an I (Insert) command thus: IMOVE 205 210, followed by an ESC.

If everything is satisfactory, the results of the modified format definition will appear on the Video Monitor screen. If, after the ESC, an error message is displayed, something unintelligible to the compiler has been keyed in. In this case, press the space bar to continue. In order to see where the mistake has occurred, key B100T followed by two ESCs. This tells the editor to Type (T) to your screen (i.e. list it out as a display) up to 100 lines of program, starting from the Beginning (B). The program can then be inspected to locate the error. Use the F (Find) command again to get the pointer to the right place for inserting or deleting, and use the I or D commands respectively. See Section 6 for more details.

Once the format is displayed correctly without any compiler error messages, press ESC again. Another editor prompt character ('>') should be displayed on the programmer screen.

SAVING THE REVISED FORMAT DEFINITION

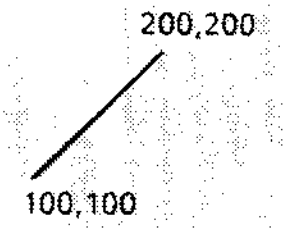
With either type of programmer, press E (for Exit) then press ESC twice. This causes the revised version of format to be saved in external store, where it overwrites the previous version of the format definition of the same number.

If a copy of the particular format is required, do an E followed by a new format number then press ESC twice. This saves the modified version under the new number, leaving the format definition of the original number unchanged.

DRAWING LINES

To draw a line, start by using a MOVE command to set the current writing position to where the line is to start. Then, use the keyword DRAW followed by the X,Y co-ordinates of where the line is to finish. For instance:

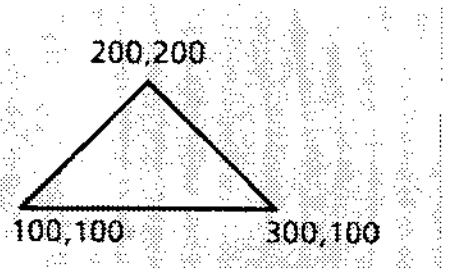
```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
DRAW 200 200
```



A line will be drawn at 45 degrees from pixel 100,100 to pixel 200,200, with a single pixel thickness. If a thicker line is required, use the SIZE command, which has already been mentioned for text character sizes. Thus, if a line thickness of 5 pixels is required, put in a SIZE 5 command before the DRAW.

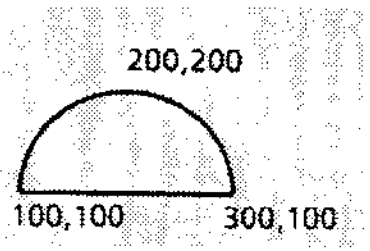
Once the line is drawn, the current writing position will be set at the end of the line ready for drawing another line. Thus, if a triangle outline is required with line thickness 5 pixels, the instructions could be:

```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
SIZE 5
DRAW 200 200
DRAW 300 100
DRAW 100 100
```



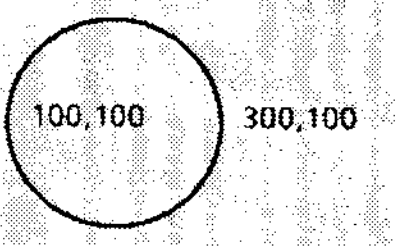
As well as drawing straight lines, DRAW can also be used to produce circles or arcs of circles, using the keyword VIA followed by the X,Y co-ordinates of a point to be passed through. The following commands would produce a semicircle on the same base as the previous triangle:

```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
SIZE 5
DRAW 300 100 VIA 200 200
DRAW 100 100
```



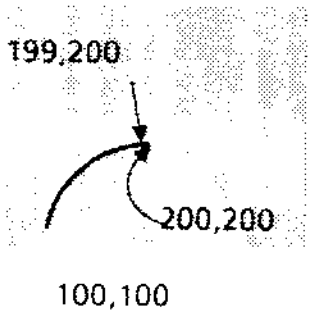
To produce a complete circle, the end point of the line given in the DRAW command needs to be the same as the starting point, and the pixel the line goes through must be a diametrically opposite point. Thus, to turn the semicircle of the previous example into a full circle, the following could be written:-

```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
SIZE 5
DRAW 100 100 VIA 300 100
```



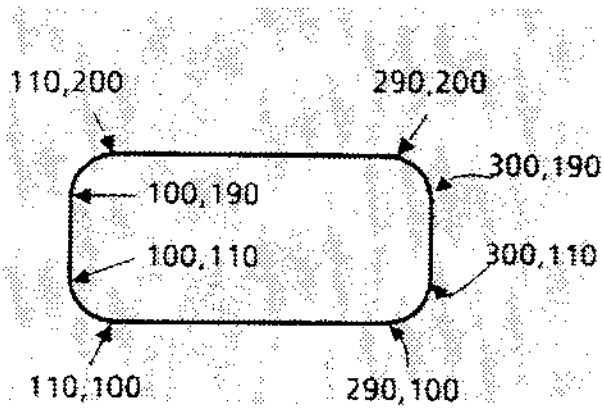
Consider a quarter circle. The problem is that, without doing a few calculations, the point through which the arc should pass will not be apparent. The method is to specify two adjacent pixels at the end. Theoretically, there will be fractionally more than a quarter of a circle, but in practice the difference will not be noticeable. Thus, to get a quarter circle which follows, to all intents and purposes, the same line as the circle of the previous example, the following could be written:-

```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
SIZE 5
DRAW 200 200 VIA 199 200
```



Thus, to draw a rectangular outline with rounded corners, the following could be written:-

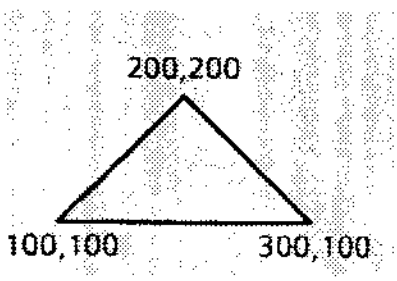
```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 110
SIZE 3
DRAW 100 190
DRAW 110 200 VIA 109 200
DRAW 290 200
DRAW 300 190 VIA 300 191
DRAW 300 110
DRAW 290 100 VIA 291 100
DRAW 110 100
DRAW 100 110 VIA 100 109
```



RELATIVE MOVE AND DRAW

In some cases, it may be more convenient to use the relative move and draw commands RMOVE and RDRAW. These work in exactly the same way as MOVE and DRAW, except that the figures given with these commands are not X,Y co-ordinates, but changes in co-ordinates. Thus, another way of drawing the triangular outline shown on the previous page would be with the format definition below:-

```
FOREGROUND RED
BACKGROUND BLUE
WINDOW 0 0 559 447
MOVE 100 100
SIZE 5
RDRAW 100,100
RDRAW 100,-100
RDRAW -200,0
```



Think of the figures after an RDRAW or RMOVE as meaning 'Right,Up". Thus, in the last command, the current writing position has moved 200 left (because of the minus sign) and kept on the same level (Up nothing).

Note...Commas or semi-colons must be used as separators, not spaces, between figures when the second one is negative.

COLOURS

Other colours, listed in Section 5 under COLOUR (DEFAULT COLOURS), can be used in place of the RED, BLUE and CYAN used so far. These include flashing colours. The colours listed are known as the 'default set', which is already set up.

However, other colours can be defined with their own names, e.g.:

```
COLOUR ORANGE 100,50,0
```

where the figures indicate the intensities of the primary colours red, green and blue (in spectrum order), as percentages of their maximum brightnesses.

For flashing colours, a sequence of colour compositions can be strung together, but each, having a fourth figure - the duration in tenths of a second. If the default colour REDFLASH flashes faster than desired, it could be re-defined as, for example:

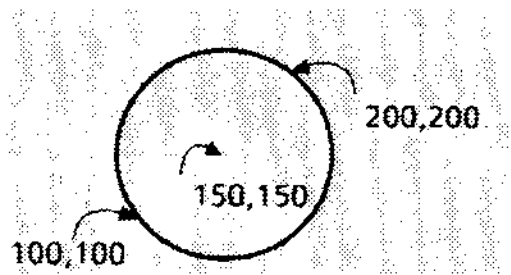
```
COLOUR REDFLASH 100,0,0,10;50,0,0,10
```

which would cause REDFLASH to give full brightness red for 1 second (i.e. 10 tenths of a second) followed by half brightness red for 1 second, the cycle then being repeated. Although COLOUR commands can be placed anywhere in the format definitions, it is recommended to place them at the start. For more details of colours and the COLOUR command, refer to the item in Section 5.

DRAWING SOLID SHAPES

If any closed outlines have been drawn, such as those in the examples above on drawing lines, they can be coloured solid by using the FILL command. This will block them in, in the current FOREGROUND colour. Before doing a FILL, the current writing position must be placed, with a MOVE, somewhere inside the outline. For instance, taking the example of a complete circle:

```
COLOUR ORANGE 100,50,0
FOREGROUND ORANGE
BACKGROUND GREEN
WINDOW 0 0 559 447
MOVE 100 100
DRAW 100 100 VIA 200 200
MOVE 150,150
FILL
```



Note...The outline must be closed. If there is a gap in the outline of only one pixel, the colour will 'leak out' and flood the screen. Try the example above again with the DRAW command altered to:

```
DRAW 100 98 VIA 200 200
```

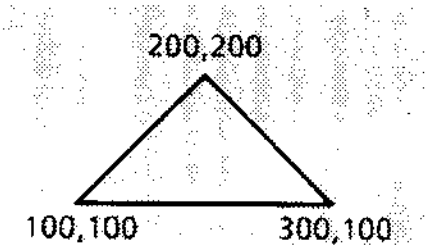
Since the circle is incomplete (pixel 100,99 is missing), the FILL will leak out.

If the circle outline is required to be orange but the inside to be blue, the foreground colour must be changed before the FILL, e.g.

```
COLOUR ORANGE 100,50,0
FOREGROUND ORANGE
BACKGROUND GREEN
WINDOW 0 0 559 447
MOVE 100 100
DRAW 100 100 VIA 200 200
MOVE 150,150
FOREGROUND BLUE
FILL
```

For drawing solid triangles and rectangles, the Display Language includes the keywords TRIANGLE and RECTANGLE. IMAGEM executes these faster than using the FILL command (i.e. the display can therefore be updated faster), hence they should be used for preference where possible. For a triangle in the current foreground colour, it is only necessary to specify the co-ordinates of its corners (in any order), e.g.

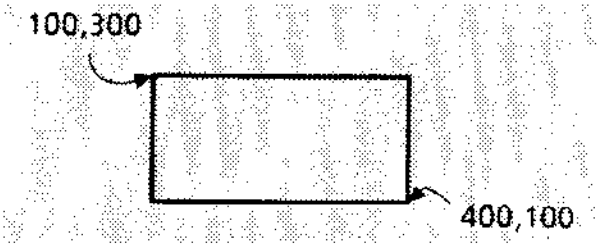
```
FOREGROUND YELLOW
BACKGROUND BLACK
WINDOW 0 0 559 447
TRIANGLE 100,100;300,100;200,200
```



As mentioned earlier, it doesn't normally matter whether spaces, commas or semicolons are used between the figures; see the item on Separators in Section 5 for more details.

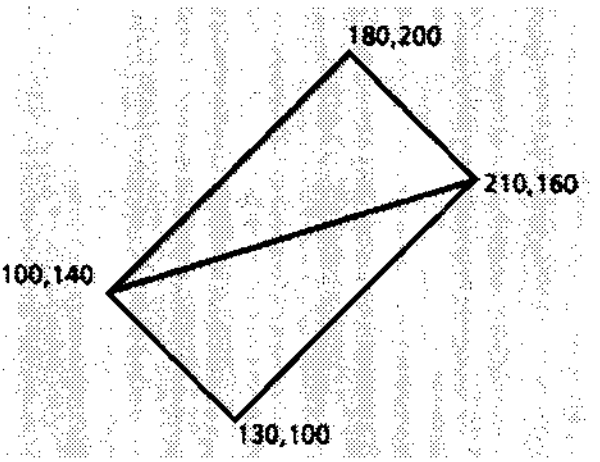
For a rectangle, the co-ordinates of two diagonally opposite corners must be specified. It doesn't matter which two, or in what order the co-ordinate pairs are given. For example:

```
FOREGROUND YELLOW
BACKGROUND BLACK
WINDOW 0 0 559 447
RECTANGLE 100,300;400,100
```



The rectangle command can only be used for rectangles which have sides parallel to the X and Y axes, i.e. parallel to the edges of the screen. If a solid rectangle is required at an odd angle, it is quickest to build it from two triangles, e.g.

```
FOREGROUND YELLOW
BACKGROUND BLACK
WINDOW 0 0 559 447
TRIANGLE 100,140;180,200;210,160
TRIANGLE 100,140;130,100;210,160
```



PATTERNS

For lines and also for colouring TRIANGLES, RECTANGLES, and other shapes blocked in with FILL, selected patterns can be specified. For lines, the item on the DRAW command in Section 5 shows the different sorts of dotted, dashed and chain dotted lines available. To specify a patterned line, simply add the pattern number at the end of the DRAW command, e.g.

```
DRAW 200 250 4
```

or, if drawing a circular arc:

```
DRAW 200 200 VIA 200 250 4
```

In a similar way, a pattern number can be added to a TRIANGLE, RECTANGLE or FILL command at the end of the command line. The list of pattern numbers and pictures is given in the item on FILL in Section 5.

COMMENTS

Within a format definition, comments can be included for personal information by using the symbol pairs '/*' and '*/'. Thus, if you want a reminder about what parts of your format definitions are meant to be doing, comments can be included, such as:

```
/* Solid semicircle */
```

There are very few restrictions on where comments can be placed within a format. See Section 5, Comments.

SEVERAL FORMATS

So far, the definition of a single format has been considered. However, a picture can be composed of up to four formats.

Considering video number 0, the controller can call up whichever formats it wants displayed at any time by the format numbers its program places in table locations L0 to L3, see Section 3, System Interface. The format called up by the contents of L0 is processed first. The format called up by L1 is processed next, and may overwrite some or all of the first format called up by L0. Similarly, the format called up by L2 can overwrite the previous two formats, and finally that format called up by L3 can overwrite anything already called up.

To demonstrate this, set up four formats as follows:-

```
/* First format */.
BACKGROUND RED
WINDOW 0 0 559 447
/* Second format */
BACKGROUND YELLOW
WINDOW 140 112 559 447
/* Third format */
BACKGROUND BLUE
WINDOW 280 224 559 447
/* Fourth format */
BACKGROUND WHITE
WINDOW 420 336 559 447
```

If they are called up in the above order from the controller, calling up the first format in L0, the second in L1 etc., a display like that shown below will be obtained:-

However, if they are called up in a different order, at least one of them will be entirely overwritten by a format called later. If the first format above is made to be the one that is called up last (by L3), nothing of the other formats will be seen at all.

In a practical case, it is therefore a good idea to use the last format called up to provide the operator with priority information, e.g. alarm conditions, as this last format will always overwrite any formats called up earlier.

DISPLAYING NUMBERS

All the examples of format definitions so far have been basic shapes, whereas the practical purpose of a video display is to provide information for an operator of the plant that the GEM80 controller is controlling. It is therefore necessary to know how to extract data from the controller's data tables, and how to display it as part of the picture on the video monitor screen.

The simplest way of displaying data is in the form of numbers. The Display Language has three keywords for this: DECIMAL, UNSIGNED and HEX. Because it is the most commonly used, DECIMAL will be dealt with here. The other two keywords are very similar, and details of them are given in Section 5.

A typical DECIMAL command looks like this:

```
DECIMAL ^^^#.## G[100]
```

This command displays the data held in the controller's table location G100. In Display Language, the table location number must be included in square brackets as shown. The row of symbols following the DECIMAL keyword is known as the FIELD DESCRIPTOR, and specifies how many digits are to be displayed before and after the decimal point. The hash ('#') symbols indicate significant digits, i.e. digits will always be displayed in these positions. The caret ('^') symbols indicate leading figures which, if zero, will display as spaces. One caret must be allowed for a minus sign if the value to be displayed could go negative. Thus, if G100 contained the value 1234, the display would be:

```
12.34
```

and it if contained -12345, the display would be:

```
-123.45
```

See Section 5, DECIMAL for further examples, and for where the leading zeros are to be displayed.

The digits are displayed, as for fixed text, in characters of the current SIZE and in the current FOREGROUND colour. Thus, if a message is to be displayed saying "FEEDWATER TEMPERATURE = nn.nn DEG" where nn.nn represents a number taken from table location W235, it could be written as:

```
FOREGROUND YELLOW
BACKGROUND BLACK
WINDOW 0 0 559 447
SIZE 2
'FEEDWATER TEMPERATURE = '
DECIMAL ^#.## W[235]
' DEG'
```

FEEDWATER TEMPERATURE = 40.32 DEG

Note...Where a number of items of text and/or figures are to be displayed, each item carries on from the end of the previous one. If some further text is to appear on a new line, use the keyword NEWLINE, as shown in the example below. NEWLINE always moves the current writing position back to X co-ordinate 0 (zero) relative to the current WINDOW. This is normally the left-hand edge of the window, but it is shown later how this can be altered. If it is desired to start the text elsewhere, a MOVE can be used after the NEWLINE.

```
FOREGROUND YELLOW
BACKGROUND BLACK
WINDOW 0 0 559 447
SIZE 2
'FEEDWATER TEMPERATURE = '
DECIMAL ^#.## W[235]
' DEG'
NEWLINE
'FEEDWATER PRESSURE = '
DECIMAL ^^#.# W[236]
' PSI'
```

```
FEEDWATER TEMPERATURE = 40.32 DEG
FEEDWATER PRESSURE = 120.4 PSI
```

GRAPHICAL REPRESENTATION OF NUMERIC VALUES

As well as displaying values as numbers, it may be desirable to show them graphically. A very common way is as a bar chart.

Consider the DRAW command for drawing lines. By using the SIZE command, these lines could be made several pixels wide. For a bar chart, bars can be produced using lines a given number of pixels wide using these commands. The question is how to make the lengths variable.

All that is necessary is to replace one of the co-ordinates, attributed with constant values in all the previous examples, by:

- A Controller table location reference (as used with DECIMAL).

- A mathematical expression using a table location reference.

Thus, a horizontal bar could be written as:

```
FOREGROUND BLUE
BACKGROUND BLACK
WINDOW 0 0 559 447
MOVE 0,400
SIZE 7
DRAW W[123],400
```

This example would draw a horizontal bar, starting from co-ordinates 0,400 (towards the top of the left-hand edge of the display), and extending to the right a number of pixels equal to the number contained in the controller's table location W123.

Similarly, a vertical bar could be written as:

```
FOREGROUND BLUE
BACKGROUND BLACK
WINDOW 0 0 559 447
MOVE 20,20
SIZE 7
DRAW 20,W[123]/100+20
```

This example would draw a vertical bar starting near the bottom left-hand corner of the display (co-ordinates 20,20) and draw a bar of height equal to the number contained in table location W123, divided by 100, and offset by 20 in the Y direction (since the bar started 20 pixels up from the bottom of the display).

NUMBERS, VARIABLES AND EXPRESSIONS

The last two examples of single horizontal and vertical bars of variable length indicate the magic of IMAGEM Display Language. Without exception, any number in any of the commands given so far that involved numbers can be replaced by:

- Table locations values.

- Mathematical expressions.

Therefore data taken from the controller s data tables can be used to control:

- Co-ordinates (in MOVE, DRAW, TRIANGLE and RECTANGLE commands).

- Line widths and character sizes (in the SIZE command).

- Colour compositions (in the COLOUR command).

REPEATED OPERATIONS

The previous examples of variable length bars showed only how to draw single bars. For a bar chart, it may be required to show several bars, representing several process measurement values. If these values are stored in adjacent table locations in the controller's data tables, a FOR..TO statement can be used, e.g.

```
FOREGROUND BLUE
BACKGROUND BLACK
SIZE 7
WINDOW 0 0 559 447
FOR A = 1 TO 20
(MOVE 20*A,20
DRAW 20*A,W[200 + A],100 + 20)
```

If the programmer is familiar with the BASIC language, it will be noticed that the FOR..TO statement is similar, except that there is no NEXT. Instead, all the commands to be repeated must be enclosed in parentheses (round brackets). The closing parenthesis acts like a BASIC 'NEXT'.

Another thing to notice in the above example is that it includes a variable denoted by 'A'. This is called a simple variable. Up to 26 simple variables can be used in a format definition, i.e. all the letters of the alphabet. However, of these, three have specific pre-defined usage:

- V This is set to the L-table number that called up the particular format. V is set to this number when the format definition is executed. Subsequently, this value could be overwritten by assigning some other value to V.

- X,Y These two variables hold the X and Y co-ordinates of the current writing position. Their values therefore change as the format definition is being processed.

All the other 23 letters of the alphabet, however, can be used freely when a variable is needed in a format definition.

SWITCHED CONTROL

Extending the bar chart further, it may be required to alter the colour of any bar that exceeded a certain height, i.e. if the value that the bar represented exceeded a certain pre-set value. To do this, IMAGEM Display Language includes the IF..THEN..ELSE construction. The ELSE is optional. It is only necessary to use the ELSE if switching between two alternatives.

Thus, if the colour of a bar is to be changed from BLUE to RED if it exceeds 300 pixels in height, it could be written as:

```
BACKGROUND BLACK
SIZE 7
WINDOW 0 0 559 447
FOR A = 1 TO 20
(MOVE 20*A,20
IF W[200 + A],100 > 300 THEN
FOREGROUND RED
ELSE FOREGROUND BLUE
DRAW 20*A,W[200 + A]/100 + 20)
```

As with most programming languages, the comparators >, <, >=, <=, = and <> can be used. See the item on IF..THEN..ELSE in Section 5. A switch between commands can also be achieved by looking at an individual bit in a table location. For instance, if there was an output from B4.11 from the controller feeding a solenoid valve, it may be required to alter the colour of some lines representing pipework, so as to indicate whether liquid was flowing or was stopped. To do this, a command such as the following could be used:-

```
IF B[4].11 THEN FOREGROUND RED
ELSE FOREGROUND GREEN
```

For this bit-dependent condition, note first that the bit number must be written outside the square brackets as shown above, NOT inside them. Second, note that there is no comparator. The command following the THEN is obeyed if the bit is ON, while any ELSE command (if there is one) is obeyed if the bit is OFF.

WINDOW AND ORIGIN

Except for the example of overlapping formats, in all the other examples above, the WINDOW size has always been set to the area of the whole screen. It may appear that a WINDOW command just sets the area of the video monitor screen in which your format definition is to appear. However, the WINDOW command has a broader application than this.

To begin with, the format definition can include commands where the co-ordinates are outside the WINDOW area, or in fact are outside the screen area. Write the following:-

```
FOREGROUND RED
BACKGROUND GREEN
SIZE 5
WINDOW 0 0 559 447
MOVE -10000 112
DRAW 10000 112
FOREGROUND BLUE
MOVE 140, -10000
DRAW 140 10000
```

Here, a red horizontal line and a blue vertical line have been drawn, the first having X co-ordinates extending from -10 000 to 10 000, and the second having Y co-ordinates extending from -10 000 to 10 000. The maximum numbers usable are -32 768 and 32 767. These lines are set a quarter of the way in from the edge of the screen.

Note... Where the lines cross, the blue line overwrites the red line, because the commands for drawing this line appear later in the format definition.

Edit the WINDOW command to:

```
WINDOW 0 0 279 223
```

The WINDOW area is now restricted to the bottom left-hand quarter of the screen. The red and blue lines terminate at the edges of the window, half-way up and half-way across the screen.

Now move the window, modifying the WINDOW command to:

```
WINDOW 280 224 559 447
```

This sets the window to the top right-hand corner of the screen. Notice that the red and blue lines are still shown, in the same places relative to the window as before. This shows that, ***in a format definition, co-ordinates are relative to the WINDOW,*** not to the left-hand corner of the screen.

Unless otherwise specified with an ORIGIN command, as far as the format definition commands are concerned, the bottom left-hand corner of a WINDOW has co-ordinates 0,0.

The ORIGIN command, allows this to be modified. Since the format definition can include commands with co-ordinates in the range -32 768 to 32 767, the format definition can be considered as defining what is drawn on a large sheet of paper, much larger in area than the size of the screen. Without an ORIGIN command, co-ordinates 0,0 of this sheet of paper always line up with the bottom left-hand corner of the window. When the position of the WINDOW on the screen is altered as above, the piece of paper moves with it. Then, only a small rectangle out of the sheet of paper near its origin will ever be displayed.

If an ORIGIN command is included, the sheet of paper can be moved underneath the window so as to display any desired portion of it. For instance, if the given ORIGIN command is inserted before the WINDOW command:

```
FOREGROUND RED
BACKGROUND GREEN
SIZE 5
ORIGIN -100,0
WINDOW 0 0 559 447
MOVE -10000 112
DRAW 10000 112
FOREGROUND BLUE
MOVE 140, -10000
DRAW 140 10000
```

This tells the format definition that the point with co-ordinates 0,0 on the notional sheet of paper must be placed at a point 100 pixels to the left of the bottom left-hand corner of the WINDOW. This causes the vertical blue line to be displayed much nearer to the edge of the window. If the ORIGIN command is altered to:

```
ORIGIN -200,0
```

the blue line will disappear, as, by moving the format definition 200 pixels to the left, the line no longer comes within the WINDOW.

Therefore there is an enormous expanse of format definition available, representing an array of pixels 65 536 square. Using the ORIGIN command, this definition can be moved so that the required portion comes under the WINDOW.

SCALE

With the ORIGIN command, whichever portion of the format definition comes under the WINDOW can be moved. The format definition can also be reduced using the SCALE command. SCALE is a command that allows all co-ordinates to be multiplied by a percentage factor between 10% and 100%. It can only be used to shrink the co-ordinates, not to expand them. If a format definition is written containing lines and shapes and text covering a much larger area than will fit into a WINDOW, it can be reduced using, for example, SCALE 10. This gives an overall view, but then a particular area can be expanded by using SCALE 100 to see the detail full size, using the ORIGIN command to determine the particular part of the format that is to appear on the display.

An exercise for practice is given in Section 5 in the item on the ORIGIN command. This is arranged so that the SCALE can be altered by writing to the Controller's table location G0, the ORIGIN by writing to G1 and G2, and the WINDOW by writing to G3 to G6.

When co-ordinates are multiplied by a SCALE factor, they obviously have to be rounded to the nearest integer so as to line up with a physical pixel on the screen.

When line thicknesses set by SIZE are multiplied by a SCALE factor, they similarly have to be rounded to the nearest integer. If a scaled line thickness comes to less than 0.5, the line will not be displayed.

Likewise, character sizes when scaled could come to less than 0.5, in which case the characters are not displayed. Characters can only display in integer multiples of width and height, e.g. if SIZE 5 characters are scaled down to 75% using the command SCALE 75, they would come out as SIZE 4, 4 being the nearest integer to 3.75.

By careful design of a format definition, some of the fine detail can be caused to disappear from an overall scaled-down display, but this detail will re-appear when the SCALE factor is altered to give a full size display.

FUNCTIONS

Where a section of format definition is repeated several times (e.g. to draw a valve symbol or a hopper symbol), instead of writing the same sequence of commands again, the sequence can be written as a Function.

To set up a function, use the format editor in exactly the same way as if writing a new format, but, instead of giving the function a number, it must be given a name.

When the function has been written, it must be called up the format definition by using a CALL command, followed by the function name, e.g.

```
CALL VALVE
```

To write a function that can be used in various different places in a format definition, it is better not to use fixed co-ordinates, as the particular symbol might be wanted at several different positions on the display. To ease this problem, the Display Language includes the commands RMOVE and RDRAW. These move the current writing position and draw lines to points relative to where the current writing position was last set, and not to absolute co-ordinates. Thus, the command:

```
RMOVE 100,100
```

is an instruction to move the current writing position 100 in the X-direction (i.e. to the right) and 100 in the Y-direction (i.e. upwards) from where it is already.

For those commands that do not have relative equivalents, make use of expressions with the variables X and Y, e.g.

```
TRIANGLE X,Y;X + 100,Y + 100;X + 200,Y
```

will draw a triangle relative to the current writing position X,Y. Formats can also be used where mathematical formulae are to be used several times in the format. More details of functions are given in Section 5, Functions.

Note... Functions can be used by different format definitions. So, where there is some clever graphic or mathematical construction which may be needed on several format definitions, it is worth considering writing it as a function.

This page left intentionally blank

IMAGEM DISPLAY LANGUAGE REFERENCE SECTION

Keywords are listed in upper case, other topics in lower case.

This page left intentionally blank

FUNCTION

To combine two conditions when using an IF..THEN statement.

EXAMPLES

The following command:-

```
IF P>4 AND Q>45 THEN FOREGROUND RED
```

sets the FOREGROUND to RED if both the specified conditions (P greater than 4, Q greater than 45) are met, otherwise no action is taken.

NOTES

AND is referred to as a Boolean operator, and an expression containing an AND as a Boolean expression.

For bitwise AND operations, see the item on MASK.

RULES

When a Boolean expression containing the operators NOT, AND and OR is evaluated, the NOTs are evaluated first, followed by ANDs, and then by ORs. This order of priorities can be altered by using parentheses (round brackets). For instance:

```
IF (P = 1 OR Q = 1) AND T = 1 THEN MOVE 0,28
```

The OR is forced to be evaluated first. If the parentheses were omitted:

```
IF P = 1 OR Q = 1 AND T = 1 THEN MOVE 0,28
```

The AND would be evaluated first, i.e. the command would be equivalent to:

```
IF P = 1 OR (Q = 1 AND T = 1) THEN MOVE 0,28
```

since the AND operator takes priority over the OR operator.

FUNCTION

To set the background colour used by subsequent WINDOW, Text and graphic commands in the program.

FORMAT

BACKGROUND colour

where:
BACKGROUND is the keyword.
'colour' is the name of a previously defined colour.

RULES

The name of the colour must be either one of the default colours, or of some other colour defined earlier in the format or function definition with the COLOUR command. If it is not, then, when the format or function is compiled, an error message saying "UNDEFINED COLOUR USED" is displayed at the top left-hand corner of the monitor screen in flashing characters. Also, the area of screen that was meant to be in the undefined colour will, instead, display in GREENFLASH.

When keying in, the keyword BACKGROUND may be shortened to a minimum of BAC, but it will be expanded to the full keyword on any program listings or displays.

NOTES

The background colour is the colour used for the pixels that form the background to text, spaces between the dashes of patterned lines produced by DRAW and RDRAW, and the spaces between pattern lines when a patterned fill is used with the FILL, RECTANGLE and TRIANGLE commands.

The program can contain several BACKGROUND commands. For instance, if the background colour is changed with a BACKGROUND command immediately before displaying text (see item on 'Text'), any line of text will display on a strip of the new background colour.

EXAMPLES

Consider the following simple format definition:-

```
FOREGROUND WHITE
BACKGROUND BLACK
WINDOW 280 224 559 448
BACKGROUND RED
"Message"
```

When the WINDOW command is encountered, the upper right-hand quarter of the display will be set to BLACK from the first BACKGROUND command. Then, when the text 'Message' is encountered, this text will be displayed as WHITE characters on a RED background strip from the second BACKGROUND command. The rest of the WINDOW area will remain BLACK.

FUNCTION

To call up a function from within a format definition, or from within another function definition, where the function called up affects the display only and does not calculate a value.

FORMAT

CALL name (P1,P2,P3 ....)

where:
CALL is the keyword.
'name' is the name of the function being called up.
P1,P2,P3 etc. are values (variables or expressions) that the function needs for its internal calculations or logic.

RULES

Name' must be the name of a function that is already defined and stored in an external memory. If no function of the given 'name' exists, an error message is displayed on the video monitor screen when the format is invoked by one of the controller's L-tables.

The quantity of parameters in the list P1,P2,P3 etc. must match the quantity of values required by the function. If too many parameters are provided, the excess will be ignored. If too few parameters are provided, those not given will default to a value of 0 (zero). The list of parameters must be included in parentheses (round brackets) as shown. If there are no parameters, parentheses should not be used.

NOTES

CALL is only needed where a function produces an effect on the display only. If using a function to calculate a value from some mathematical or logical formula, the function can be treated as though it were just another variable (see item on 'Variables'). For more information on parameter passing, see the item on 'Functions'.

EXAMPLES

If a previously defined a function called VALVE produces a symbol for a valve, probably consisting of two triangles, the following instruction could be written:-

```
MOVE 100,200
CALL VALVE
```

This would draw the valve symbol starting at co-ordinates 100,200 within the current WINDOW.

FUNCTION

To display a specified quantity of characters of text, taken from data held in one of the controller's data tables, starting from a specified table location.

FORMAT

CHAR f,t

where:
'f' is the field width of the text, i.e. the quantity of characters.
't' is the first table location from which text is to be taken.

RULES

'f' can be a constant, variable or expression.

'f' must evaluate to a number in the range 0 to 32 767 inclusive. If it evaluates to a negative number, 'f' will default to zero, and a fault code will be written to table location F9.

If 'f' is zero, whether by default or not, then no characters will be displayed.

't' is the table location containing, as its lower byte, the code for the first character to be displayed.

If the specified table location 't' does not exist in the controller, then 'f' question marks ('?') will be displayed.

If 't' exists but 'f' is sufficiently large for the end of the table to be reached before 'f' character codes have been found, nothing is displayed.

NOTES

Each word in the controller's data table 't' is treated as two bytes. Each byte specifies one character, taken from the character set previously set up using the Character Editor (see Section 6 of this manual). If extra characters have not been added, any byte that is greater than hexadecimal 7F causes a space to be displayed.

Of the two bytes that make up a word, the less significant byte (bits 0 to 7) is taken as the code for the first character displayed, and the more significant byte (bits 8 to 15) for the second character displayed. If the field width 'f' is made equal to 1, to display just one character, the code for this character will be taken from the lower byte.

Where it is desired to display text messages taken from several contiguous table locations, or from the messages area of P-table on GEM80 Controllers other than 200 series, the STRING command is easier to use.

EXAMPLES

To display a single character whose code is stored in the lower byte of table location J0 in the controller:

```
CHAR 1,J[0]
```

To display a variable length message, starting with the character whose code is stored in the lower byte of table location W72, where the number of characters in the message is stored in table location G15:

```
CHAR G[15],W[72]
```

FUNCTION

To define a new colour, or to re-define an existing colour, that can then be used in the FOREGROUND and BACKGROUND commands.

FORMAT

COLOUR name red,green,blue,time

or

COLOUR name red,green,blue,time;red,green,blue,time; ... etc.

where:
COLOUR is the keyword.
'name' is the name of the colour being defined.
'red', 'green' and 'blue' are numbers representing the percentage intensities of the primary colours red, green and blue (in spectrum order) between 0 and 100% of their maximum intensities.
'time' is a number representing tenths of a second for which the particular colour intensities are to be displayed before moving on to the next 'red,green,blue' settings in the sequence.

RULES

'Name' must begin with a letter, followed by one to seven further alphabetic or numeric characters. The minimum length of 'name' is therefore two characters, and the maximum length eight. In fact, more than eight characters can be used, but only the first eight are significant.

Note...Spaces are not allowed, and case is ignored, i.e. upper and lower case letters are not distinguished.

When the end of the sequence of colour settings is reached, the video re-cycles and starts at the beginning of the sequence again.

The numbers used for the colour intensities can be constants, variables or expressions. Expressions can include any of the in-built functions (COS, LOC and SIN), but must not include user-defined functions.

Where the colour being defined does not flash, the sequence consists of only one 'red,green,blue,time' setting. The 'time' determines how often the colour settings are reviewed. The 'time' value is therefore of no consequence if the 'red,green,blue' settings are constants; it can be either omitted or set to zero. However, if the settings are variable values derived from the GEM80 Controller's data tables, the 'time' value is important. If 'time' is omitted, the colour settings will be set only once, and will not be reviewed while the format or function definition containing the particular COLOUR command is operating.

The separators between the colour and time values obey the standard rules (see item on Separators).

When keyed, the keyword COLOUR may be shortened to a minimum of COL, but it will be expanded to the full keyword on any program listings or displays.

DEFAULT COLOURS

The following table gives a list of the names of the default set of colours, and their red, green and blue compositions.

Name	Equivalent COLOUR definition
BLACK	COLOUR 0,0,0
RED	COLOUR 100,0,0
GREEN	COLOUR 0,100,0
BLUE	COLOUR 20,20,100
REDFLASH	COLOUR 100,0,0,4;50,0,0,2
GREENFLASH	COLOUR 0,100,0,4;0,50,0,2
BLUEFLASH	COLOUR 20,20,100,4;10,10,50,2
CYAN	COLOUR 0,100,100
MAGENTA	COLOUR 100,0,100
YELLOW	COLOUR 100,100,0
CYANFLASH	COLOUR 0,100,100,4;0,50,50,2
MAGENTAFLASH	COLOUR 100,0,100,4;50,0,50,2
YELLOWFLASH	COLOUR 100,100,0,4;50,50,0,2
WHITE	COLOUR 100,100,100
WHITEFLASH	COLOUR 100,100,100,4;50,50,50,2
GREY	COLOUR 75,75,75

NOTES

The COLOUR command can be used to:

- define new colours having new names.
- re-define the composition of any previously defined colour, whether this is one of the default set given in the table above, or one with a new name defined by the user.

As many colours as desired can be defined, but not more than 16 different ones can be used at any one time. As a screen display can consist of up to 4 formats, the maximum figure of 16 applies across all 4 formats. In general, the same named colours will be used in different formats.

It is possible to ***make use*** of 16 colours in the format definitions without displaying them all at any one time. For instance, if a value from within a GEM80 Controller table exceeded a given limit value, it may be desirable to change a colour from plain to flashing. In this case, the format definition would have to include both the plain and flashing colours in the definition, even though both might not be displayed at the same time. The maximum figure of 16 is for colours ***in use***, not for colours being displayed.

If the number of colours in use is greater than 16, the message "TOO MANY COLOURS" is displayed on the video monitor when the seventeenth colour is encountered during execution.

Although the number of colour names in the format and function definitions in use at any one time is limited to 16, these colours can have variable composition. For instance, the red, green and blue values can come from the controller's data tables. See examples below:-

EXAMPLES

```
COLOUR REDFLASH 100,0,0,7;50,0,0,3
```

This redefines REDFLASH as a flashing red that repeatedly cycles through bright red for 7/10ths of a second followed by a dimmer red for 3/10ths of a second.

```
COLOUR PINK 80,10,20
```

This defines a new colour made up of a fixed set of colour intensities, which is set up once in the format or function and then left.

```
COLOUR TINT G[0],G[1],G[2],5
```

This defines a colour, the composition of which is variable and depends on the values held in the controller data table locations G0, G1 and G2. Because the time value is given as 5, the composition of the colour is reviewed every 5.10ths of a second, and updated if there have been any changes to the contents of these table locations.

```
COLOUR TINT G[0],G[1],G[2]
```

This defines a colour, the composition of which is variable and depends on the values held in the controller data table locations G0, G1 and G2 as in the previous example, but because no time value is given, the composition of the colour is set up when the command is encountered and not reviewed. That is, any subsequent changes to the contents of G0, G1 and G2 will be ignored.

DUPLICATE COLOUR DEFINITIONS

The program could be written in such a way that there is more than one COLOUR definition for a given named colour within the formats that make up a display. This is most likely to occur where the displays are assembled from different combinations and permutations of a set of formats.

In this case, the resulting colour composition for the colour of a given name will be determined by:

- any COLOUR definition for it that is actually executed in the ***earliest*** format making up the display. Any COLOUR definitions in later formats are ignored.

- if there is more than one COLOUR definition for the given colour name within this format, the *last* COLOUR definition executed in that format.

If, for instance, a format definition is written such as:

```
COLOUR TEST1 100,0,0
(Instructions block 1)
COLOUR TEST1 0,100,0
(Instructions block 2)
```

the first definition of the colour TEST1 will be ***ignored;*** the second COLOUR statement will be used to define TEST1 for ***both*** blocks of program instructions.

EXAMPLE OF DUPLICATED COLOUR STATEMENTS

```
Format 1:   IF A = 1 THEN (COLOUR TEST1 100,0,0)   ' Red ';
            IF B = 1 THEN (COLOUR TEST1 0,100,0)   ' Green ';

Format 2:   COLOUR TEST1 0,0,100                   ' Blue '.
```

Suppose that these formats are being used on Video 0, in the order listed. This means that the L-Table values will have been set to:

```
L0 = 1
L1 = 2
```

If A and B are both zero, no COLOUR statement is executed in Format 1; the colour is set to BLUE by the COLOUR statement in Format 2.

If, on the other hand, A is set to 1, the first COLOUR statement in Format 1 is executed but not the second. The colour TEST1 is therefore set to RED. Since Format 1 has set TEST1 already, the COLOUR statement in Format 2 is then ignored.

If A and B are both set to 1, both COLOUR statements in Format 1 are executed. The colour TEST1 depends on the last COLOUR statement executed in this format and it is therefore set to GREEN. Since Format 1 has set TEST1 already, the COLOUR statement in Format 2 is ignored.

FIXED, VARIABLE AND FLASHING COLOURS

For a fixed (non-variable, non-flashing) colour, any COLOUR definition need only be executed once. The definition remains in force until such time as the composition of the colour is re-defined.

For a variable or a flashing colour, it is recommended that the COLOUR definition for it should be executed at every execution of the program.

EXAMPLE OF UNDEFINED FLASHING COLOUR

Format 1:
```
IF A = 1 THEN (COLOUR TEST1 100,0,0)    " Red ".
IF B = 1 THEN (COLOUR TEST1 0,100,0)    " Green ".
```

Format 2:
```
COLOUR TEST1 0,0,100,10,0,0,50,10       " Blueflash "
```

Suppose that these formats are being used on Video 0 in the order listed as in the previous example. If A = 0 and B = 0, the the colour TEST1 will be set by Format 2 to flashing blue. If Format 2 is now deleted from the display, colour TEST1 becomes undefined. In the previous example, doing this would not have mattered as the colour was fixed. However, in this case the colour is flashing, so that the COLOUR statement needs to be executed regularly. As a result, the colour TEST1 defaults to GREENFLASH from the default colour set.

Care should be taken when writing programs where a colour of a given name changes using an IF..THEN statement dependent on some condition. Where this is done it is recommended that the ELSE clause is always used so that a colour is defined as one of the two possible compositions. If this is not done an undefined colour could well be obtained. If the colour is fixed and has been previously defined, this doesn't matter. If, however, the colour is variable or flashing, it does matter, and the undefined colour defaults to GREENFLASH.

For instance, use:

```
IF G[100]>45 THEN (
    COLOUR ALARM 100,0,0,5,50,0,0,5
)
ELSE (
    COLOUR ALARM 0,100,0
)
```

rather than simply:

```
IF G[100]>45 THEN (
    COLOUR ALARM 100,0,0,5,50,0,0,5
)
```

However, it is much better to write:

```
COLOUR ALARM1 100,0,0,5,50,0,0,5
COLOUR ALARM2 0,100,0
IF G[100]>45 THEN (
    BACKGROUND ALARM1
)
ELSE (
    BACKGROUND ALARM2
)
```

This way, two separate colours are defined instead of one, and the COLOUR definitions are executed on every pass through the program. The only occasion when this cannot be done, and when the COLOUR statements themselves might have to be included in the IF..THEN clause is where the number of colours in use would otherwise exceed 16.

PLACEMENT OF COLOUR STATEMENT WITHIN FORMAT AND FUNCTION DEFINITIONS

As explained above, COLOUR statements can be placed anywhere in the program. The rules have been given on how these statements are obeyed, especially where there are duplicated definitions for the same colour. However, it is best if any special colours are defined at the beginning of the format or function definition. Using COLOUR commands half-way through a program does not give good program readability, and so the start of a program is the best place for all COLOUR commands.

COLOUR COMPOSITIONS

The Video system caters for 16 different intensity levels for each of the red, green and blue primary colours. This allows any colour to be defined as one out of 4096 possibilities. For convenience, the intensity values given with a COLOUR command are expressed as percentages of maximum intensity, but internally the video system converts these percentages into the nearest of the 16 intensity levels.

When defining colours, remember that video monitors form colours by ***adding together*** light of the primary colours red, green and blue. Video colours do not mix like paints or pigments. Pigments give a particular colour by ***absorbing*** some of the sprectral colours out of incident white light (e.g. daylight) and only reflecting what is left. As you mix more pigments of different colours together, more colours of light are absorbed, less reflected, so the colour tends toward black. The opposite happens with video monitors. When the three primary colours red, green and blue are added together in the right proportions, white light results.

To make a colour lighter, add equal amounts of red, green and blue. For instance, in the default colour set, BLUE has been set to consist of 20% intensity red, 20% intensity green and 100% intensity blue. Consider this as:

(20% red + 20% green + 20% blue) + 80% blue
= 20% white + 80% blue

BLUE has been set this way because, on most video monitors, pure blue is too dark, so it has been lightened. However, BLUE can always be re-defined with a COLOUR command if a different composition better suits one particular video monitor.

With a very few exceptions, comments can be included anywhere in a format or function definition. It is recommended that this facility should be used so that, if a format or function definition ever has to be modified, say 6 months after it was originally written, there are sufficient 'memory joggers' on how it was written, and why it was written in any particular way.

To include a comment, write the commenting text between ' and ' symbol pairs, e.g.

```
' Alarms message format '
```

Comments are ignored for display purposes, but are printed out on listings for any documentation. See Section 9 for documentation facilities.

RULES

Comments are not nestable, i.e. one comment cannot be included within another.

Comments must not split up numbers or names.

Comments must not be included in text strings.

RECOMMENDATIONS

Since formats, unlike functions, are referenced by format numbers, it is a good idea to put a comment at the very beginning of any format definition to provide it with a title.

Use comments as headings to divide long format or function definitions into sections, in the same way that paragraph headings are put in a written report.

Use comments to explain what is going on in complicated sections of code. If a section of code needs a lot of explanation, look for a simpler way of writing that section. However, do not over-comment by explaining what is quite explicit from the programming commands.

Also, use comments as reminders of what is contained in any of the controller's table locations used in the program. This may save cross referring to another print-out, e.g.:

```
' G[100] = Feedwater temperature '
```

DECIMAL CONSTANTS

The commonest way of defining constants is simply as a string of digits, e.g.

1234

Decimal constants must always be integer, without decimal points. Apart from a leading minus sign if a constant is negative, the only characters allowed in a decimal constant are digits in the range '0' to '9' inclusive. Leading zeros are ignored, and are removed on any program listing or display.

The range of constants permissible is -2147483648 to 2147483647. However, most commands expect smaller numbers than these and limit or truncate such large numbers. For details, see the item for any particular command in use.

HEXADECIMAL CONSTANTS

A hexadecimal constant is distinguished from a decimal constant by a leading 'commercial at' symbol ('@'), e.g.

@41AF

Minus signs are not allowed, but the letters 'A' to 'F' can be used in addition to digits. These letters can be keyed in either upper or lower case, but they are always shown on subsequent listings and displays in upper case. Leading zeros are ignored, and are removed on any program listing or display.

The range of hexadecimal constants permitted is from @0 to @FFFFFFFF.

CHARACTER CONSTANTS

A character constant takes the form of a single printing character or space enclosed within either single or double quotes, e.g.

'P'

The value is taken as the ASCII (ISO-7) code for the character, which therefore lies in the range 20 to FE hexadecimal (32 to 126 decimal).

Although character constants can be keyed in within either single or double quotes, they are subsequently displayed or listed within single quotes.

To enter the character constant for a single quote, double quote or commercial at symbol, enter the character twice, e.g.

produces the code value for a double quote.

VARIABLES AND EXPRESSIONS

In all cases where IMAGEM Display Language requires a value, it can be provided either as a constant, a Variable or an Expression. See the items in this Section for Variables and for Expressions.

FUNCTION

To provide the cosine of an angle.

FORMAT

COS (e1)

where:
COS is the keyword.
(e1) is a value or expression.

RULES

(e1) must be in degrees.

The result given by the COS function equals the trigonometrical cosine multiplied by 1000 and rounded to the nearest integer.

EXAMPLES

COS(45)

returns a value of 707, i.e. 0.7071 * 1000 rounded to the nearest integer.

P = COS(G[100])

assigns to the variable P, a value equal to 1000 times the cosine of the value held in the controller's table location G100, rounded to the nearest integer.

FUNCTION

To alter the width and height of characters in a display.

FORMAT

CSIZE n1, n2

where:
CSIZE is the keyword.
'n1' is the character width in pixels.
'n2' is the character height in pixels.

RULES

'n1' and 'n2' may be constants, variables or expressions.

Both 'n1' and 'n2' must be specified.

The character width 'n1' may take a value of either 6 or 7. If it is specified as, or evaluates to, any other number, 'n1' takes the value 7 by default.

The character height 'n2' must be in the range -32768 to 32767. However, the expected range is between 1 and 14. Values that are negative or zero are treated as 1 and values greater than 14 are treated as 14.

NOTES

Character heights can also be changed with the SIZE command, but not character widths. The rules governing character height for the CSIZE command are exactly the same as those for the SIZE command. Refer to the item for this command.

If, in a format or function definition, there are SIZE and CSIZE commands calling for conflicting character heights, the latest height value encountered is used.

With reduced width (6-pixel wide) characters, each character is formed from its 7 x 14 definition (see Section 7, Option 3) with the right-hand column of pixels removed.

With reduced height characters, each character is formed by removing the bottom rows of its pixel definition.

FUNCTION

To display numeric data in signed decimal form.

FORMAT

DECIMAL f,n

where:
DECIMAL is the keyword.
'f' specifies the required field width and type.
'n' specifies the data for display.

FIELD DESCRIPTOR

The field descriptor 'f' represents, in pictorial form, the way in which the numeric data is to be displayed. The symbols used are identical to those used for message definitions in GEM80 Controllers other than 200 series, and are as given below:-

Symbol	Name	Meaning
^	Caret	Leading space
0	Zero	Leading zero
#	Hash	Significant digit
.	Full stop or period	Decimal point

EXAMPLES

DECIMAL ^^^^#.# G[25]

displays the contents of the controller's table location G25 in signed form, with one decimal place, and with up to five leading digits, or up to four leading digits with a minus sign if the contents of G25 are negative.

DECIMAL 0000#.# G[25]

is essentially the same, except that leading zeros will be displayed.

The following table shows how these examples would display for different contents of table location G25.

G25 contents	Display	
	First example	Second example
1	0.1	00000.1
12	1.2	00001.2
123	12.3	00012.3
1234	123.4	00123.4
12345	1234.5	01234.5
-1	-0.1	-0000.1
-12	-1.2	-0001.2
-123	-12.3	-0012.3
-1234	-123.4	-0123.4
-12345	-1234.5	-1234.5

To display a long number, it could be written as, for instance:

```
DECIMAL ^^^^^^^^^#.### G[25] - 10000*G[26]
```

which would take the contents of table location G26, multiply it by 10000, add this to the contents of table location G25, and then display the result.

RULES

The data for display 'n' may be a constant, variable or expression. If the number contains more digits than will fit into the space allocated in the field descriptor, the display will take the form of question marks presented in the specified layout.

The maximum quantity of digits that can be specified with a field descriptor is 15.

The size of characters displayed, as for Text, is determined by the last SIZE or CSIZE command encountered. See items on SIZE and CSIZE.

When keyed in, the keyword DECIMAL can be shortened to a minimum of DEC, but it will be expanded to the full keyword on any program listings or displays.

FUNCTION

Bitwise exclusive-OR operator.

EXAMPLES

The following command:-

```
IF G[50] DIFFER @104 <>0 THEN FOREGROUND REDFLASH
```

sets the FOREGROUND to REDFLASH if some of the bits in the controller's table location G50 do not correspond with those in the hexadecimal constant 104 (= 0000 0001 0000 0100).

DIFFER can also be used in expressions to produce a value which consists of 1's where two variables contain unequal bits, e.g.

```
C = A DIFFER B
```

would perform a calculation such as the following:-

```
A    0011 1100 1010 0101
B    1111 0000 1111 0000
     ---------------------
C    1100 1100 0101 0101  = A DIFFER B
```

NOTES

For brevity, the examples shown above have 16-bit numbers. In practice, simple variables are 32-bit, and values taken from the controller's table locations are 16-bit padded to out to 32-bit, with leading zeros if positive, or suitably adjusted if negative.

PRIORITY

When an expression is evaluated:

- HIGH, LOW, Shift, MASK and UNION are evaluated before DIFFER.
- Multiplication, division, addition and subtraction are evaluated after DIFFER.

Details of this keyword are given in the item for 'FOR..TO'.

FUNCTION

To draw a straight line, circle, or circular arc, starting from the current writing position.

FORMAT

DRAW X2,Y2,n

RDRAW X2,Y2,n

DRAW X2,Y2 VIA X3,Y3,n

RDRAW X2,Y2 VIA X3,Y3,n

where:
DRAW or RDRAW is the keyword.
X2,Y2 are the co-ordinates of the end of the line to be drawn.
VIA is an optional additional keyword used only if drawing a circle or circular arc.
X3,Y3 are the co-ordinates of a point that the circular arc must pass through.
'n' is an optional line pattern number.

LINE PATTERN NUMBERS

The table below gives the patterns available and their pattern numbers:-

Number	Description
0	Solid line (default)
1	Dashed line Type 1
2	Dashed line Type 2
3	Dashed line Type 3
4	Dashed line Type 4
5	Short dashed line Type 1
6	Short dashed line Type 2
7	Short dashed line Type 3
8	Chain dotted line Type 1
9	Chain dotted line Type 2
10	Long dashed line Type 1
11	Long dashed line Type 2
12	Long dashed line Type 3

The detail of these line patterns is shown in Figure 1.

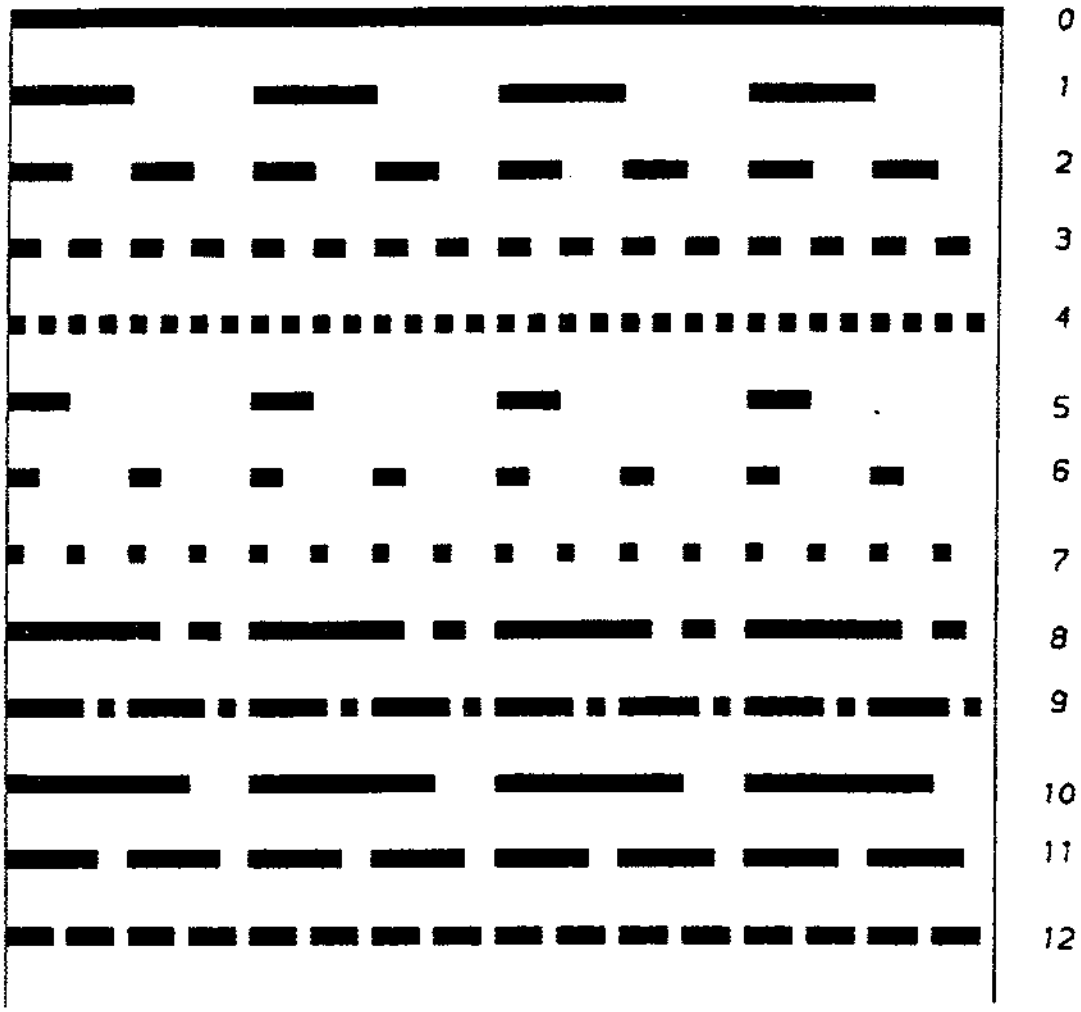


Figure 1

If the SIZE command is used to increase the thickness of lines from 1 pixel to a number of pixels, the lengths of the dashes in the patterned lines are increased pro rata, maintaining the same geometrical proportions.

RULES

If VIA X3,Y3 is omitted, the line drawn will be straight.

If 'VIA X3,Y3' is included, the line drawn will be an arc of a circle terminating at X2,Y2, but going via X3,Y3. If the co-ordinates specified by X2,Y2 are identical to the current writing position, so that the line end position is the same as its starting position, a complete circle will be drawn. The diameter of this circle will be the distance between X2,Y2 and X3,Y3.

If the DRAW command is used, the co-ordinates specified for all of X2,Y2,X3 and Y3 are taken as relative to the origin of the current WINDOW.

If the RDRAW command is used instead, all co-ordinates are taken as relative to the current writing position.

The pattern number 'n' must always come last. If drawing a straight line and the 'VIA X3,Y3' is omitted, the 'n' must follow 'Y2'.

NOTES

When using a patterned line, the spaces between the dashes from which the line is made display in the current BACKGROUND colour. If a BACKGROUND command is used to re-define this colour shortly before a DRAW or RDRAW, a dashed line may be obtained with alternating dashes of the current FOREGROUND and BACKGROUND colours, on top of the previous background.

After a DRAW or RDRAW command, the current writing position will be at the end of the line drawn by the command. If drawing a figure such as a box or unfilled triangle, it is only necessary to use a MOVE to set the starting co-ordinates of the first line. See the last of the Examples below:-

EXAMPLES

To draw a straight line at 45° to the axes from 100,100 to 300,300:

```
MOVE 100,100
DRAW 300,300
```

The same line could have been obtained with RDRAW as follows:-

```
MOVE 100,100
RDRAW 200,200
```

To draw a semicircle with the same endpoints as the above straight line:

```
MOVE 100,100
DRAW 300,300 VIA 100,300
```

or, using RDRAW:

```
MOVE 100,100
RDRAW 200,200 VIA 0,200
```

To draw a complete circle, instead of the semicircle of the previous example:

```
MOVE 100,100
DRAW 100,100 VIA 300,300
```

or:

```
MOVE 100,100
RDRAW 0,0 VIA 200,200
```

All the above examples draw solid lines one pixel thick. To draw the straight line of the first example with a chain dotted line Type 2 three pixels thick, write:

```
MOVE 100,100
SIZE 3
DRAW 300,300,9
```

or, for a similarly patterned circular line:

```
MOVE 100,100
SIZE 3
DRAW 100,100 VIA 300,300,9
```

To draw a square box of side 200 pixels long whose bottom left-hand corner is at co-ordinates 100,100:

```
MOVE 100,100
DRAW 300,100
DRAW 300,300
DRAW 100,300
DRAW 100,100
```

or:

```
MOVE 100,100
RDRAW 200,0
RDRAW 0,200
RDRAW -200,0
RDRAW 0,-200
```

FUNCTION

To allow some other action to be defined when using an IF..THEN clause, converting it into an IF..THEN..ELSE clause.

DETAILS

Refer to the item on IF..THEN..ELSE in this Section.

Within a format or function definition, mathematical expressions can be used in any place where a number is required.

An expression consists of variables and or constants with mathematical and logical operations. Details of the variables available are given in a separate item (see Variables). A mathematical operator consists of one of the following four:-

- Multiplication (*)
- Division (/)
- Addition (+)
- Subtraction (-)

Logical operators work on the individual bits in a word. They are represented by keywords, for which more details are given in items of these names in this Section:

- Bitwise AND (MASK)
- Bitwise OR (UNION)
- Bitwise EXCLUSIVE OR (DIFFER)

In addition to the above, there are also byte extraction and shift operators. Details of these are given in items in this Section under their keywords and 'Shift':

- Extract byte, bits 0-7 (LOW)
- Extract byte, bits 8-15 (HIGH)
- Shift right (>>)
- Shift left (<<)

In an expression, the order of calculation is:

1. HIGH and LOW
2. Shift left and right
3. MASK
4. UNION
5. DIFFER
6. Multiplication and division
7. Addition and subtraction

The normal sequence of evaluation can be altered by using parentheses (round brackets).

Where two operations have equal priority, they are evaluated left-to-right. For example:

D = A.B/C

is evaluated as:

D = (A.B)/C

and ***not*** as:

D = A/(B.C)

All calculations are handled in 32-bit integer arithmetic. Since input data to the video system from the controller's data tables is 16-bit, a calculation like A*B/C can be performed without losing accuracy from truncation errors.

Note... This accuracy is only obtainable if calculations are performed in a sensible order. Thus, if the following calculation is attempted:-

```
E=(W[100]/W[101])*W[102]
```

the first operation is a division of two 16-bit numbers which will result in a truncation error. However, if the expression is written as:

```
E=(W[100]*W[102])/W[101]
```

the first operation is a multiplication of two 16-bit numbers, giving a 32-bit number. This can then be divided by a 16-bit number without a significant truncation error.

EXAMPLES

```
HEX ^^^^^^^# (G[25] MASK @FFFF)+G[26]<<16
```

This displays a number in hexadecimal notation, where the less significant four characters represent the contents of table location G25, and the upper four characters represent the contents of G26. As each table location contains 16 bits, G26 has to be shifted 16 bits to the left. As shift is a higher priority operation than plus, parentheses are not necessary.

```
D=W[100]*COS([W[101]/360)+(W[102] MASK @FFF0 * HIGH[W103])/100
```

This assigns a complicated expression to the simple variable D. It is simply an example, of no practical use, included to show the sort of expression that can be included in a format or function definition when necessary.

FUNCTION

To call for an outline produced with the DRAW or RDRAW commands to be blocked in with colour, and to define the pattern if a patterned fill is required rather than solid colour.

FORMAT

FILL n

where:
FILL is the keyword.
'n' is the pattern number.

FILL PATTERN NUMBERS

The table below gives the patterns available and their pattern numbers:-

Number	Description
0	Solid fill (default)
1	Fine dots
2	Coarse dots
3	Chequerboard
4	Horizontal stripes
5	Fine vertical stripes
6	Fine vertical stripes, close spaced
7	Vertical stripes
8	Diagonal stripes - right sloping - type 1
9	Diagonal stripes - right sloping - type 2
10	Diagonal stripes - left sloping - type 1
11	Diagonal stripes - left sloping - type 2
12	Cross hatch
13	Fine chequerboard

The same pattern numbers are used when defining the fill patterns of TRIANGLEs and RECTANGLEs.

The detail of these patterns is shown in Figure 2 below.

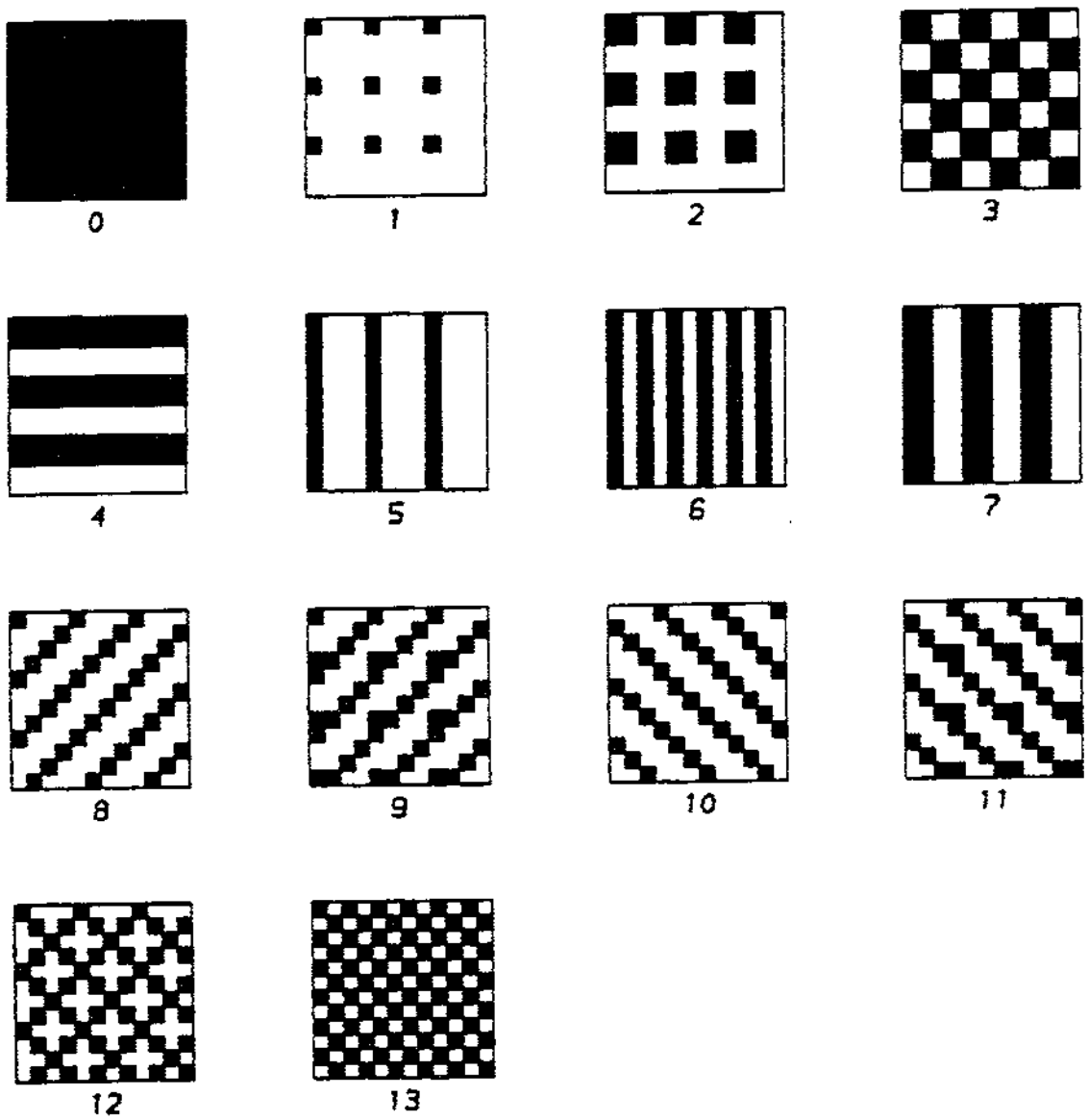


Figure 2

RULES

If n is omitted, the fill is solid.

The colour of a fill is the current FOREGROUND colour. With a patterned fill, the colour of the spaces between the stripes, cross hatching, etc., is the current BACKGROUND colour.

Before using a FILL command, the current writing position must already be within the required outline, and also within the current WINDOW area. If it is not, a MOVE or RMOVE command must be used to move the current writing position into the area to be filled.

If using a solid fill, the pixel at which the current writing position is set must not already be set to the FOREGROUND colour.

If using a patterned fill, the pixel at which the current writing position is set must neither be in the FOREGROUND nor the BACKGROUND colour.

The size of patterns is not affected by the SIZE or SCALE commands. The patterns are always as shown in Figure 2 and their position on the screen is fixed. If there are a number of shapes filled with the same pattern that abut or overlap, there will not be any dislocation of the pattern where the shapes meet.

After a FILL is executed, the current writing position finishes up at the same place as it started, i.e. at the pixel at which it was located prior to the FILL.

The outline to be filled must be a closed outline. If there is any gap, even of only a single pixel, the fill may 'leak out' and flood the whole screen.

When keyed in, the keyword FILL may be shortened to FIL, but it will be expanded to the full keyword on any program listings or displays.

NOTES

The FILL command executes much slower than the figure drawing commands, RECTANGLE and TRIANGLE. These commands should be used in preference to FILL for simple shapes.

The method of operation of FILL is as follows. First, the system reads back the colour of the pixel at the current writing position. It then re-sets the colour of this pixel and all adjacent pixels of the same colour to the current FOREGROUND colour. Consequently, it does not alter the colour of any pixel in the lines from which the figure boundary is drawn, neither does it alter the colour of any figures included within the outline. The FILL will truncate if it reaches the edge of the current WINDOW.

When using a patterned fill, the pixel within the outline to which the current writing position is first moved must be neither the current FOREGROUND colour nor the current BACKGROUND colour. It may be that the BACKGROUND colour of the pattern (i.e. the colour of the spaces between the lines, cross hatching or chequerboard) is to remain unchanged. In this case, two FILLs must be used - first a solid fill, painting the figure some colour that is neither the FOREGROUND colour nor the required BACKGROUND colour, and then the patterned fill of the required colours. See the following Examples:-

EXAMPLES

To produce a RED half-moon shape on a BLACK background:

```
BACKGROUND BLACK
FOREGROUND RED
WINDOW 0 0 559 447
MOVE 100 100
DRAW 300 300
DRAW 100 100 VIA 200 200
MOVE 150 200 ./* Set writing position inside shape */
FILL
```

Suppose instead that the half-moon is to be filled, not with solid RED, but with RED diagonal stripes, pattern type 8, on a BLACK background. If the last command was simply altered to:

```
FILL 8
```

the FILL command would fail to work, since the current writing position is already set to the BACKGROUND colour BLACK. Instead, a solid fill of some different colour must be performed first, e.g. e.g.:

```
BACKGROUND BLACK
FOREGROUND RED
WINDOW 0 0 559 447
MOVE 100 100
DRAW 300 300
DRAW 100 100 VIA 200 200
FOREGROUND GREEN
MOVE 150 200  ./* Set writing position inside shape */
FILL          ./* Solid green fill */
FOREGROUND RED
FILL 8
```

Note...After the first FILL by the arbitrary colour GREEN, the current writing position remains at its starting position, and is therefore in the correct place for the second FILL; another MOVE is not necessary.

However, filling the area twice is time consuming. If there is a spare colour (i.e. an unused colour from the maximum of 16 different colours: refer to the item on COLOUR), a quicker way of processing the format or function definition is to define a colour of identical composition to the existing BACKGROUND, but with a different name. The equivalent of the previous example could then be performed, but at nearly twice the speed, by writing:

```
COLOUR BLACK2 0,0,0 ./* Identical composition to BLACK */
BACKGROUND BLACK
FOREGROUND RED
WINDOW 0 0 559 447
MOVE 100 100
DRAW 300 300
DRAW 100 100 VIA 200 200
BACKGROUND BLACK2
MOVE 150 200 ./* Set writing position inside shape */
FILL 8
```

EXTRA COLOURS

Pattern 13 is so fine that it does not show up as a pattern from normal viewing distances. Hence it can be used for producing extra colours, by suitable choice of FOREGROUND and BACKGROUND colours, without introducing extra COLOUR names.

FUNCTION

To repeat a command or set of commands several times.

FORMAT

FOR v = e1 TO e2 s1

FOR v = e1 DOWNTO e2 s1

where:
FOR, TO and DOWNTO are keywords.
'v' is a simple variable.
e1 and e2 are constants, variables or expressions representing the start and end conditions.
s1 is the statement (a command or series of commands) that will be obeyed while 'v' is within the range 'e1' to 'e2'.

RULES

For the definition of a simple variable, see the item on Variables.

If either or both 'e1' and 'e2' are variables or expressions, they are evaluated only once, when the particular FOR statement is first encountered in the format or function definition. Any subsequent changes to 'e1' and 'e2' are ignored.

Initially, 'v' is set equal to 'e1'. After each execution of the statement 's1', the value of 'v' is incremented with FOR..TO, or decremented with FOR..DOWNTO. That is, its value is increased by 1 or decreased by 1 respectively.

The FOR loop terminates (i.e. 's1' is ignored) when 'v' exceeds 'e2' with FOR..TO, or becomes less than 'e2' with FOR..DOWNTO.

If initially, when 'v' is set equal to 'e1', it already exceeds 'e2' with FOR..TO, or is less than 'e2' with FOR..DOWNTO, the FOR loop never operates, i.e. 's1' is never obeyed.

After termination of a FOR loop, execution of the format or function definition continues at the first command after 's1'.

NOTES

FOR loops can be nested. That is, a statement 's1' can be written which itself contains a FOR loop. This FOR loop can contain a further FOR loop, and so on.

To guard against a FOR loop taking an excessive time to execute if the number of repeats is large, especially if these FOR loops contain further FOR loops, a software watchdog is started whenever a format starts execution. See Section 11, Test Facilities.

The maximum depth of nesting allowable in a format or function definition is 16. The maximum depth of nesting allowed at display time is 24; this figure is higher because functions can be called. An error message is displayed on the monitor or screen.

EXAMPLES

```
FOR P = 1 TO W[250]
    (MOVE P*10,30
     RDRAW 0,W[250 + P],10
    )
```

is a FOR..TO loop drawing a bar chart, where the quantity of bars is taken from the controller's table location W250, and the heights of the bars are taken from table locations W251 onwards. The statement 's1' is a compound statement consisting of two commands (a MOVE and a RDRAW), which are therefore enclosed in parentheses. If the parentheses had been omitted, only the MOVE command would have been repeated, until the quantity of repeats specified by W250 had been exhausted, after which the RDRAW command would have been executed once only.

```
FOR P = 20 DOWNTO 1
S = (S + 10000/S)/2
```

is an example of a calculation repeated 20 times, using a FOR..DOWNTO loop (it could equally as well have been a FOR..TO loop in this instance). Since the statement 's1' consists of a single command (an assignment), there is no need to enclose it in parentheses.

FUNCTION

To set the foreground colour used by subsequent text and graphic commands in the program.

FORMAT

FOREGROUND colour

where:
FOREGROUND is the keyword.
colour' is the name of a previously defined colour.

RULES

The name of the colour must be either one of the default colours, or some other colour which was previously defined in the format or function definition with the COLOUR command. If it is not, then, when the format or function is compiled, an error message saying "UNDEFINED COLOUR USED" is displayed in flashing characters at the top left-hand corner of the monitor screen. Also, the area of screen that was meant to be in the undefined colour will display in GREENFLASH.

When keying in, the keyword FOREGROUND may be shortened to a minimum of FORE, but it will be expanded to the full keyword on any program listings or displays.

Note...Unlike most other keywords, the minimum form consists of four characters, not three, so as to distinguish it from the keyword FOR).

NOTES

The foreground colour is the colour used for the pixels that form the characters in text, lines produced by the DRAW and RDRAW commands, and the fill colours used for the FILL, RECTANGLE and TRIANGLE commands.

INTRODUCTION

Many format definitions require sequences of commands that occur more than once, although there may be slight variations such as different values. Further, certain such sections of program may repeat in several different format definitions.

To save having to key in the same sequences of commands several times, the IMAGEM Display Language allows sections of program to be set up as functions. A function is equivalent to what is called a procedure or a subroutine in some high level programming languages.

Normally, a function may be defined for:

- A graphic construction for a particular shape or symbol required several times in one or more formats.
- A mathematical formula or calculation that is to be used several times.

In some cases, it may be required to define a function that does both.

NAMES

Format definitions have numbers, to allow particular formats to be called up from the controller's L-tables whereas functions have names. Apart from this one difference, a function can be set up and edited in exactly the same way as a format. Refer to Section 6 for how to use the Format Editor and what constitutes a valid name for a function.

CALLING UP A FUNCTION

If a defined function called VALVE displays a valve symbol, then, to display the symbol, put the command:

```
CALL VALVE
```

at the appropriate place in the format definition.

If a defined function called HIGHER, calculates which of two numbers is the larger, it could be put in an expression such as:

```
P = 100*HIGHER(G[100],10, 25)
```

In this second example, the function HIGHER is said to 'return' a value. This value is used directly in the expression, without the keyword CALL being needed.

Even if, as in the second example, the CALL command is not used, reference is still made to the format definition 'calling' the function, and the function 'being called' by the format.

Besides a format being called by a format definition, a function can also be called from within another function. A function can even be made to call itself. Care should be exercised in doing this since it is possible to run out of space in Internal Store because of all the variables required, and get a display-time error.

VARIABLES

When a function is called from either a format or another function, the function is allowed to have its own set of simple variables, A to Z inclusive. These are independent of the variables of the same letter designations in the calling format or function. However, the simple variables V, X and Y, have their own pre-defined usage as follows:-

- V This is set to the L-table number that called up the particular format. V is set to this number when the format definition is executed. Subsequently, the value could be overwritten by assigning some other value to V.

- X,Y These two variables hold the X and Y co-ordinates of the current writing position. Consequently their values change as the format definition is being processed.

These particular three simple variables are said to be 'global' and are accessible from anywhere within a format definition, functions it calls, and further functions called by functions.

PASSING VALUES TO AND FROM A FUNCTION

Suppose a function is being written called HYPOTENUSE that works out the length of the hypotenuse of a right-angled triangle, i.e. it works out the square root of (a^2+b^2). To be able to use such a function, it must be given the two values representing the lengths of the other two sides, e.g.

```
C = HYPOTENUSE (3,4)
```

or, if the values were variable, it might be:

```
C = HYPOTENUSE (W[100],W[101])
```

What any function does internally is to:

- assign the first value to its own local variable A.
- assign the second value to local variable B.
- assign the next value to local variable C.

and so on. The function could, perhaps, be written as:

```
C = SQRT(A*A + B*B)
RETURN C
```

where SQRT is another function, written for evaluating a square root. This is an example of one function calling another function. The function would not work if it had been written as:

```
C = SQRT(P*P + Q*Q)
RETURN C
```

because, since the function HYPOTENUSE has two values given with it, the first (3 or W[100]) is assigned to A, and the second (4 or W[101]) is assigned to B, and nothing is assigned to P or Q.

To pass the answer calculated by the function back to the calling format or function, the RETURN command must be used. An expression is allowed with RETURN, so the function could have been written as the single line:

```
RETURN SQRT(A*A + B*B)
```

GLOBAL VARIABLES

Another way of passing values to a function is to assign them to simple variables within the calling format or function, e.g.

```
P = W[100]
Q = W[101]
R = HYPOTENUSE
```

and then, in the function HYPOTENUSE, writing:

```
GLOBAL P Q
R = SQRT(P*P + Q*Q)
RETURN R
```

In this case, the GLOBAL command tells the function to use the variables P and Q from the calling format. The function cannot, after the GLOBAL statement is encountered, have any simple variables P and Q of its own. See the item on GLOBAL.

FUNCTION

To extract data from the off-line storage section of memory such as that available on GEM80/310 or 700 series controllers.

FORMAT

GET (n1, n2, n3)

where:
GET is the keyword.
'n1', 'n2' and 'n3' are constants, variables or expressions. They define, in order:

- the required block number
- the file number within that block
- the word offset of the data within that file

RULES

Execution of the command at display time is allowed only if the IMAGEM video is fitted in a controller such as a GEM80/310 or 700 series that has off-line data storage facilities included.

The command accesses a single 16-bit word, but sign extends it to 32 bits, in exactly the same way as for array variables taken from the controller's tables. Refer to the item on Variables.

If any of the 'n1', 'n2' or n3 parameters are specified as, or evaluate to, numbers exceeding the block.file.offset limits, a question mark ('?') will be displayed on the monitor screen at the current writing position. 0 (zero) is valid for any of the parameters.

Checksum calculations are not performed on the data read from off-line storage.

FUNCTION

To extract data from the off-line storage section of memory such as that available on GEM80/300 or /700 series controllers, and to display this as text. Effectively, GETCHAR combines the functions of GET and CHAR.

FORMAT

GETCHAR f, n1, n2, n3

where:
GETCHAR is the keyword.
'f' is the field width of the text, i.e. the quantity of characters.
'n1', 'n2' and 'n3' are constants, variables or expressions. They define, in order:

- the required block number
- the file number within that block
- the word offset of the data within that file

RULES

Execution of the command at display time is allowed only if the IMAGEM video is fitted in a controller such as GEM80/300 or /700 series that has off-line data storage facilities included.

The command accesses 'f' characters and 'f' must evaluate to a number in the range 0 to 32000. If 'f' evaluates to zero, no characters will be displayed. If 'f' evaluates to a negative number, again no characters will be displayed and a fault code is also be written to table location F9.

Since each table location holds a 16-bit word, and since each character is represented by an 8-bit byte, each word is displayed as two characters. The low byte is displayed first and the high byte second. If 'f' is, or evaluates to, an odd number, the last byte displayed will be a low byte.

A character can only be displayed where a byte represents a character that is included in the character set, otherwise it is displayed as a space. If the character set has not been extended beyond the default set, this would mean than any code beyond @7F would display as a space.

If the 'n1', 'n2' or 'n3' parameters are specified as, or evaluate to, block/file/offset numbers for a table location that does not exist in the controller, 'f' question marks are displayed. If any of the 'n1', 'n2' or 'n3' parameters are specified as, or evaluate to, numbers exceeding the block/file/offset limits, up to 'f' question marks ('?') are displayed on the monitor screen in place of the remaining text that could not be found. 0 (zero) is valid for any of the parameters.

Checksum calculations are not performed on the data.

When keying in, the keyword GETCHAR may be shortened to a minimum of GETC, but it will be expanded to the full keyword on any program listings or displays. Note that, unlike most other keywords, the minimum consists of four characters, not three, so as to distinguish it from the keyword GET.

EXAMPLES

GETCHAR 1,0,1,10

displays one character taken from off-line storage, block 0, file 1, offset 10.

GETCHAR G[100],0,W[100],W[200]

displays the quantity of characters defined by the contents of G100, taken their codes from block 0, the file whose number is specified by the contents of W100, and with offset specified by the contents of W200.

FUNCTION

To declare which simple variables used in a format definition will be accessed by a function. The function thereafter uses these variables instead of its own local variables.

FORMAT

GLOBAL v1,v2,v3 ...

where:
GLOBAL is the keyword
'v1', 'v2' etc. are the simple variables in the format definition that the function requires.

RULES

The simple variables V, X and Y are always global, and do not need any GLOBAL command to enable access to them by the current function.

Where a simple variable is declared global, the function in which the GLOBAL command subsequently occurs always uses the variables from the format definition, and not from any intermediate function definition. That is, if a format calls a first function, and the first function then calls a second function, a GLOBAL command in the second function causes that function to access the variables from the format, and not from the intermediate function.

Once a variable has been declared global within a function, it cannot be used for passing parameters to it. This means that declaring variables designated by early letters of the alphabet as global, should be avoided. Refer to the item on Functions.

The separators between the variables in the list following GLOBAL obey the standard rules (see the item on Separators). When keying in, the keyword GLOBAL may be shortened to a minimum of GLO, but it will be expanded to the full keyword on any program listings or displays.

FUNCTION

To display numeric data in hexadecimal form.

FORMAT

HEX f,n

where:
HEX is the keyword.
'f' specifies the required field width and type.
'n' specifies the data for display.

FIELD DESCRIPTOR

The field descriptor 'f' represents, in pictorial form, the way in which the numeric data is to be displayed. The symbols used are identical to those used for message definitions in GEM80 Controllers other than 200 series, and are as given below:-

Symbol	Name	Meaning
^	Caret	Leading space
0	Zero	Leading zero
#	Hash	Significant digit

EXAMPLES

HEX ^^^^# G[25]

will display the contents of the controller's table location G25 in hexadecimal form, and with up to five digits.

HEX 0000# G[25]

is essentially the same, except that leading zeros will be displayed.

The following table shows how these examples would display for different contents of table location G25.

G25 contents (hexadecimal)	Display First example	Second example
9	9	00009
9A	9A	0009A
9AB	9AB	009AB
9ABC	9ABC	09ABC

A long number display, could be written as, for instance:

HEX ^^^^^^^^^# (G[25] MASK @FFFF) + G[26]<<16

which would take the contents of table location G26, shift it left by 16 bits, add this to the contents of table location G25, and then display the result.

Note... The MASK @FFFF is needed in case the content of G25 was a negative number, because it would then be extended to a 32-bit negative number in which the leading bits would not be zeros.

RULES

The data for display 'n may be a constant, variable or expression.

If the number contains more digits than will fit into the space allocated in the field descriptor, the display will take the form of question marks presented in the specified layout.

The maximum quantity of digits that can be specified with a field descriptor is 15.

The size of characters displayed, as for Text, is determined by the last SIZE or CSIZE command encountered. Refer to the items on SIZE and CSIZE.

FUNCTION

To extract bits 8 to 15 of a variable. For a GEM80 table location, these 8 bits correspond to the more significant byte of the 16-bit word held in the table location.

FORMAT

HIGH v

where:
HIGH is the keyword.
'v' is the variable from which the next-to-bottom byte is to be extracted.

RULES

'v' can be a constant, variable or expression.

The value resulting from a HIGH operation consists of a 32-bit value, the top 24 bits of which are all zero, and the bottom 8 bits are taken from bits 8 to 15 of 'v'.

FUNCTION

An IF..THEN construction allows a set of commands to be performed only if a certain condition is met. Otherwise, the set of commands is skipped.

An IF..THEN..ELSE construction is similar, but allows switching between two alternative sets of commands depending on whether a condition is met or not.

FORMAT

IF b THEN s1

IF b THEN s1 ELSE s2

where:
'b' is a Boolean expression or a bit-selection expression.
's1' and 's2' are statements.

RULES

If 's1' and/or 's2' consist of more than one command, the set of commands must be enclosed in parentheses (round brackets).

Although a string of commands can be keyed in on one line, they will be listed on displays or print-outs with one command per line, and the closing parenthesis will appear on a line of its own. See examples.

The condition 'b' can take one of three forms:

1. **Boolean comparison**

   Format: e1 comparator e2 Example: C> = 5
   where 'e1' and 'e2' are constants, variables or expressions, and the comparator is one of:

=	Equals
<>	Not equals
> =	Greater than or equals
< =	Less than or equals
>	Greater than
<	Less than

   This form of the condition 'b' is the most commonly used form, e.g.

   IF C > = 5 THEN FOREGROUND REDFLASH.

2. **Boolean expression**

   Format: e1 Example: C-4
   where 'e1' is a variable or expression.

   The condition is met, and the statement 's1' obeyed, if 'e1' is non-zero. If 'e1' is zero, 's2' will be obeyed if an 'ELSE s2' is included.

   For example:

   IF C-4 THEN FOREGROUND GREEN

   would set the foreground colour to GREEN in all cases except when C was equal to 4. However, the program is probably easier to read if the condition is written as a Boolean comparison. The following construction would be equivalent:

```
IF C <> 4 THEN FOREGROUND GREEN
```

3. **Bit-selection expression**

Format: e1.b1 Example: G[100].12
where 'e1' is a variable or expression, and 'b1' is a constant, variable or expression evaluating to a number in the range 0 to 31.

The condition is met, and the statement 's1' obeyed, if the particular bit specified by 'b1' is ON. If it is OFF, 's2' is obeyed provided that an 'ELSE s2' is included.

For example:

```
IF A[11].7 THEN FOREGROUND BLUE ELSE FOREGROUND RED
```

EXAMPLES

```
IF A[14].13 THEN (
    P = X
    Q = Y
    MOVE 400,433
    SIZE 1
    BACKGROUND REDFLASH
    "  "
    FOREGROUND YELLOW
    BACKGROUND BLUE
    " OVERTEMPERATURE"
    MOVE P,Q
)
```

causes a message to be displayed, if bit A14.13 is ON, towards the top right-hand corner of the screen, and then restores the current writing position to where it was previously. It also puts a flashing red square before the message.

Note... The set of commands to be obeyed is enclosed in parentheses. If bit A14.13 is OFF, the execution jumps to the first command in the format or function definition appearing after the closing parenthesis.

```
IF W[100] > 10000 AND W[101] > 10000 OR W[102] = 0 THEN (
    FOREGROUND RED
    BACKGROUND WHITE
)
ELSE (
    FOREGROUND GREEN
    BACKGROUND YELLOW
)
```

shows an example where the condition to be met is a Boolean expression containing a number of separate conditions, combined with the Boolean operators AND and OR. Refer to the items for AND, NOT and OR.

FUNCTION

To assign a value to a variable.

FORMAT

LET v = e

where:
LET is the keyword.
'v' is either a simple variable, or an L-table array variable.
'e' is a constant, variable or expression.

RULES

If 'v' is a simple variable, the keyword LET is optional, and is usually omitted. However, if 'v' is an L-table variable, the keyword LET is obligatory.

LET cannot be used with any array variable other than L-table variables. Any attempt to do so results in a compiler error message.

EXAMPLES

LET L[32 + A] = 27

assigns the value 27 to L-table location 32 + A.

NOTES

Values can be assigned to any L-table location available in the controller. However, remember that the early L-table locations from L0 to L31 are used to control which formats are displayed on each video, to monitor the activity of the videos, and to hold overflow flags. Unexpected effects may occur if data is assigned to these early table locations without considering the implications. For normal use, it is recommended that values are assigned only to table locations L32 upwards.

For more information on assignment, refer to the item in this Section on Variables.

FUNCTION

To return the absolute address of a specified controller data table location.

FORMAT

LOC(a[e])

where:
LOC is the keyword.
'a' is the required table letter.
'e' is a constant, variable or expression evaluating to the required table location number.

RULES

A location of the specified table letter and number must exist in the controller's data tables, otherwise a display time fault will occur, and zero is returned.

EXAMPLES

```
LOC(G[10])
```

returns the absolute address of table location G10.

NOTES

The chief use of LOC is prior to calling functions. For example, it may be required to use a function to produce bar charts for various sets of data taken from different tables in the controller. It is easy enough to write a function which uses a particular table, and the format definition could then be written in, e.g.

```
CALL BARCHART (120,20)
```

where the values given might mean the table offset and the number of bars, e.g. the function in this instance is to draw 20 bars starting with a value taken from W[120]. However, if the function is to be able to draw bars with values taken from any of the controller's data tables, it is necessary to pass a number to the function, defining where the absolute address of the table location can be found. This function is obviously different from the previous one. At one point in the format definition, the following could be written:-

```
CALL BARCHT2(LOC(C[32]),20)
```

which might cause a bar chart to be drawn for 20 values starting from C32, while at another point in the format definition there may be:

```
CALL BARCHT2(LOC(G[100]),10)
```

to draw 10 bars starting from G100. Using LOC to calculate the address of table locations, a suitably written function using these addresses can then be produced that will cover several applications, utilizing data from any of the controller's data tables.

FUNCTION

To extract the least significant byte (bits 0 to 7) of a variable.

FORMAT

LOW v

where:
LOW is the keyword.
'v' is the variable from which the bottom byte is to be extracted.

RULES

'v' can be a constant, variable or expression.

The value resulting from a LOW operation consists of a 32-bit value, the top 24 bits of which are all zero, and the bottom 8 bits are taken from the bottom 8 bits of 'v'.

FUNCTION

Bitwise And operator.

EXAMPLES

The following command:-

```
IF G[50] MASK @104 <>0 THEN FOREGROUND REDFLASH
```

sets the FOREGROUND to REDFLASH if either bit 2 or bit 8 in the controller's table location G50 are set to 1. This is because the hexadecimal constant 104, written out in binary, is equal to

0000 0001 0000 0100.

MASK can also be used in expressions to produce a value which consists of 1's where two variables both contain 1 for the same bit numbers, e.g.

```
C = A MASK B
```

would perform a calculation such as the following:-

```
A    0011 1100 1010 0101
B    1111 0000 1111 0000
     ---------------------
C    0011 0000 1010 0000 = A MASK B
```

NOTES

For brevity, the examples above are shown with 16-bit numbers. In practice, simple variables are 32-bit, and values taken from the controller's table locations are 16-bit padded to out to 32-bit, with leading zeros if positive, or suitably adjusted if negative.

PRIORITY

When an expression is evaluated, HIGH, LOW and Shift are evaluated before MASK, UNION, DIFFER, multiplication, division, addition and subtraction are evaluated after MASK.

On the video monitor screen, text can be displayed that is either defined in the format and function definitions or taken from the controller's data tables.

- For details of the method of outputting text defined in the format and function definitions, refer to the item on Text.
- To output text taken taken from the controller's data tables, use the STRING command. To output odd characters, the CHAR command may be more convenient. When used by the STRING command, the textual data must be in the correct format in the controller's data tables, as given below:-

FORMAT OF MESSAGES

Messages held in the controller's data tables can be stored in either of two forms:

- a string of bytes in any retained workspace table.
- on GEM80 Controllers other than 200 series, as messages in the 'messages area of the P-table.

MESSAGES IN RETAINED WORKSPACE TABLES

Each table location used for a message is treated as being made up of two bytes. The lower byte (bits 0 to 7) is used first, and the higher byte (bits 8 to 15) is used second.

At the start of the message, the first table location must contain as the first (lower) byte a number representing the quantity of characters in the message (not including the length byte itself). This number should always be an odd number.

The remaining bytes in the message consist of the characters that will be displayed. Any CR or LF characters (@0A and @0D) are treated, when used for display by IMAGEM, as Newline characters (see the item on NEWLINE). If an odd byte is left over at the end of the message, make this byte a NUL (@00).

As an example, to set up the message 'Help' in table locations W100 onward, set up W100 to W102 to contain:

Table Location	Contents in Hex.	Explanation
W100:	@4805	ASCII 'H' + 05 (for 5 bytes in message)
W101:	@6C65	ASCII 'l' + ASCII 'e'
W102:	@0070	ASCII NUL + ASCII 'p'

On programmers, the codes for printing characters can be entered directly, without having to convert them into ASCII codes, but it may be necessary to enter special characters by their code numbers. Refer to the item on the OFFSET command.

Several contiguous messages can be set up in a workspace table , that is, the next message begins in the next table location, e.g. W103 in the example above. In this case, in a STRING command, it can be defined that the nth message is to be displayed. Thus, in the example above, if there was a second message starting in W103, it could be stated that message 2 display was to start from W100. Further details are given in the item on the STRING command.

MESSAGES IN P-TABLE MESSAGES AREA

On a GEM80 Controller other than a 200 series, messages can be entered directly into the 'messages area' of P-table using the controller's message editor. When such messages are entered, it is not necessary to bother about which table locations they are stored in, as the message editor sorts this out. In fact, they are stored at the upper end of the P-table, chained together in a block.

When it is required to display one of these messages using a Graphic video system, it is necessary to establish the table location of the first message. To avoid having to perform this calculation in the format or function definition, the video system includes the pseudo table location X[0] which can be accessed directly.

Note...GEM80 Controllers do not contain any X-table, so the designation X[0] does not cause any conflict.

In addition to X[0], it is only necessary to specify the message number in the STRING command. Refer to the item on the STRING commands for more details.

FUNCTION

To move the writing position from its current position to any pixel defined.

FORMAT

MOVE X1,Y1

RMOVE X1,Y1

where:
MOVE or RMOVE is the keyword.
X1 is the X co-ordinate of the pixel to be moved to.
Y1 is the Y co-ordinate of the pixel to be moved to.

RULES

With MOVE, X1 and Y1 must be given relative to the origin of the current WINDOW. Which pixel this is can be modified with the ORIGIN command.

With RMOVE, X1 and Y1 must be given relative to the current writing position.

X1 and Y1 are independent of any changes made by the SIZE command to character sizes and line thicknesses.

The separators between co-ordinates obey the standard rules. Refer to the item on Separators.

NOTES

MOVE is normally used to set the current writing position prior to displaying text or drawing lines, so as to set the position for the start of the text or the line. Refer to the items in this Section for Text and DRAW.

The first character of a string of text being displayed appears with its top left-hand pixel where the current writing position is set. On completion of displaying a text string, the current writing position moves forward to the first pixel after the last character in the string, adjacent to the top right-hand pixel of this character, i.e. ready for another character, if there is one.

FUNCTION

To provide a carriage-return/line-feed sequence.

FORMAT

NEWLINE

where:
NEWLINE is the keyword.

RULES

NEWLINE requires no values following it.

NOTES

NEWLINE is provided as a separate keyword because, when specifying a text string, it can become difficult to read the program if embedded carriage-returns or line-feeds have to be used, e.g.

```
"PIECE OF TEXT
"
```

is not as clear to read as:

```
"PIECE OF TEXT"
NEWLINE
```

Without using NEWLINE, readability gets worse if text is conditionally displayed, e.g.

```
IF Z=4 THEN "
" ELSE 'PIECE OF TEXT
"
```

is not as clear to read as:

```
IF Z=4 THEN
    NEWLINE
ELSE
    ("PIECE OF TEXT'
    NEWLINE
    )
```

There is little to be gained with long passages of text, so that embedded carriage will not make any significant difference. However, where there is a display including other commands such as DECIMAL, the use of NEWLINE can improve readability.

RULES

When a NEWLINE (or a carriage-return or a line-feed) is encountered in the program, the current writing position moves down by the current character size height, as set by the last SIZE or CSIZE command.

If a SIZE has been omitted, then by default the text will be SIZE 1, and display 14 pixels high; in this case, a NEWLINE moves the current writing position down 14 pixels.

Also, the current writing position returns to X co-ordinate 0 (zero). In most cases, this will be at the left-hand edge of the current WINDOW, as the origin of a window (the pixel whose co-ordinates are 0,0) is normally the bottom left-hand pixel. However, if this is altered this with the ORIGIN command, the current writing position will return to be vertically in line with the origin pixel. If the origin is set to be outside the WINDOW, e.g. if the X co-ordinate of the origin is negative, the text string may be truncated, or possibly not be displayed at all.

FUNCTION

To negate a Boolean or bit-selection expression, i.e. to make it true if false and vice versa.

EXAMPLES

```
IF NOT A[12].5 THEN FOREGROUND BLUE
```

sets the FOREGROUND colour to BLUE if bit A12.5 in the controller's A-table is OFF.

NOTES

Although NOT is mainly of use in bit-selection expressions as in the example above, it can also be used in Boolean comparisons, e.g.:

```
IF NOT C = 5 THEN FOREGROUND REDFLASH
```

but the equivalent is frequently preferred:

```
IF C<>5 THE FOREGROUND REDFLASH
```

Similarly, NOT can be used in Boolean expressions such as:

```
IF NOT C-D THEN P = W[100]
```

which sets P equal to the value held in the Controller's data table location W100 whenever C and D are equal. Again, the following equivalent is more often preferred:-

```
IF C = D THEN P = W[100]
```

However, with bit-selection expressions, where a command is required, obeying when a particular bit is OFF, using NOT is the easiest way of doing it.

RULES

When a Boolean expression containing the operators NOT, AND and OR is evaluated, NOTs are evaluated first, followed by ANDs, and then by ORs. This order of priorities can be altered by using parentheses (round brackets). For example:

```
IF NOT A[12].4 AND A[12].5 THEN MOVE 0,75
```

the MOVE is obeyed if bit A12.4 is OFF and bit A12.5 is ON. However, if you write:

```
IF NOT (A[12].4 AND A[12].5) THEN MOVE 0,75
```

the MOVE is obeyed if either of the bits A12.4 and A12.5 are OFF.

FUNCTION

To bodily shift the codes of the character set.

FORMAT

OFFSET n

where:
OFFSET is the keyword.
'n' is the numeric offset.

RULES

The value 'n' given with OFFSET can be a constant, variable or expression.

Only the least significant 8 bits of 'n' are used in evaluating the offset. If a number outside the range 0 to 224 is used, this is evaluated to 0 having no shift offset.

If, after adding the offset, a resultant character code is greater than 255 decimal (= @FF hexadecimal), the value wraps round to the start of the displayable character set, from @20 onwards.

OFFSET remains in force until re-set by another OFFSET command or the end of the format definition is reached.

OFFSET affects all characters displayed, i.e. not only in Text specified in quotes, by the STRING and CHAR commands, but also figures displayed by the DECIMAL, HEX and UNSIGNED commands.

When keying in, the keyword OFFSET may be shortened to a minimum of OFF, but it will be expanded to the full keyword on any program listings or displays.

EXAMPLES

The command:

```
OFFSET @3F
```

would cause:

```
"A" to display as user-defined character @80
"B" to display as user-defined character @81
etc.
```

This command may be found useful, if for instance, an italic character set was defined in addition to the default Roman character set. It would then be possible to enter text strings in normal characters but display them in italics. Thus, if the italic alphabet was defined as characters with codes @80 to @99 (= 128 to 151 decimal), the following could be writen:-

```
OFFSET 63
"TEXT DISPLAYING IN ITALICS"
```

Note... If only an alphabet has been defined as above, other punctuation marks and spaces may not come out as intended. It is better to define a complete second printing character set. In the example above, the spaces would display as underline characters.

Note... The offset value can be entered in either decimal or hexadecimal, as for all other values in the Display Language. However, an OFFSET also affects figures displayed by DECIMAL, HEX and UNSIGNED commands. Hence, it may be necessary to use:

```
OFFSET 0
```

to re-set the codes of the character set before displaying any numbers.

FUNCTION

To combine two conditions when using an IF..THEN statement.

NOTES

OR is referred to as a Boolean operator, and an expression containing an OR as a Boolean expression. For bitwise Or operations, refer to the item on UNION.

EXAMPLES

The following command:-

```
IF P>4 OR Q>45 THEN FOREGROUND RED
```

sets the FOREGROUND to RED if either or both of the specified conditions (P greater than 4, Q greater than 45) are met, otherwise no action is taken.

RULES

When a Boolean expression containing the operators NOT, AND and OR is evaluated, NOTs are evaluated first, followed by ANDs, and then by ORs. This order of priorities can be altered by using parentheses (round brackets). For example:

```
IF (P=1 OR Q=1) AND T=1 THEN MOVE 0,28
```

forces the OR to be evaluated first. If the parentheses were omitted:

```
IF P=1 OR Q=1 AND T=1 THEN MOVE 0,28
```

the AND would be evaluated first, i.e. the command would be equivalent to:

```
IF P=1 OR (Q=1 AND T=1) THEN MOVE 0,28
```

since the AND operator takes priority over the OR operator.

FUNCTION

To set which pixel is to be treated as the origin of a WINDOW, i.e. which pixel is treated as having X,Y co-ordinates of 0,0.

FORMAT

ORIGIN X1,Y1

where:
ORIGIN is the keyword.
X1,Y1 are the co-ordinates of the pixel to be used as the origin, measured relative to the bottom left-hand corner of the window.

RULES

X1 and Y1 may be constants, variables or expressions.

X1 and Y1 must be in the range -32768 to 32767. Numbers outside this range will be limited to these values.

An ORIGIN command must come before the WINDOW command to which it applies.

The separators between the co-ordinates obey the standard rules. Refer to the item on Separators.

When keying in, the keyword ORIGIN may be shortened to a minimum of ORI, but it will be expanded to the full keyword on any program listings or displays.

NOTES

What is defined in a format definition can include text and graphic figures with co-ordinates outside the area of the screen. The WINDOW command defines how much of this format definition is visible on the video monitor screen. Using the ORIGIN command, the format definition can effectively be shifted to vary what comes within the window.

EXAMPLES

Suppose it is required to set up a format definition with the default settings of:

```
ORIGIN 0,0
WINDOW 0,0,559,447
```

so that the definition produces a display of:

```
400 401 402 403 404 405
300 301 302 303 304 305
200 201 202 203 204 205
100 101 102 103 104 105
000 001 002 003 004 005
```

If the WINDOW is changed without changing the ORIGIN, only that portion that is displayed is altered, still working from the bottom left-hand corner, e.g. changing the window as follows:-

```
ORIGIN 0,0
WINDOW 280,171,559,447
```

will produce a display of:

```
. . . . . 200 201 202
. . . . . 100 101 102
. . . . . 000 001 002
. . . . . . . . . . . .
. . . . . . . . . . . .
```

If both the WINDOW and the ORIGIN are changed, setting the origin back to the corner of the display as follows:-

```
ORIGIN -280,-171
WINDOW 280,171,559,447
```

the display would be:

```
. . . . . 403 404 405
. . . . . 303 304 305
. . . . . 203 204 205
. . . . . . . . . . . .
. . . . . . . . . . . .
```

Note...The co-ordinates given for the corner of the screen in an ORIGIN command must be relative to the corner of the window. The co-ordinates given in a WINDOW command, however, must always be relative to the screen itself, i.e. WINDOW co-ordinates must always be absolute.

If the format definition extends the array of figures off the screen to the right-hand side, then by moving the origin, what is displayed could be shifted, e.g.

```
ORIGIN -280,0
WINDOW 0,0,559,447
```

might give a display of:

```
403 404 405 406 407 408
303 304 305 306 307 308
203 204 205 206 207 208
103 104 105 106 107 108
003 004 005 006 007 008
```

EXERCISE

The following exercise may familiarise the operator with the way that WINDOW, ORIGIN and SCALE interact. Set up two formats as follows:-

FIRST FORMAT

```
* Format to set background to black */
BACKGROUND BLACK
WINDOW 0,0,559,447
```

SECOND FORMAT

```
' Format setting up number array '
SIZE 3
SCALE G[0]
ORIGIN G[1],G[2]
FOREGROUND BLACK
BACKGROUND CYAN
WINDOW G[3],G[4],G[5],G[6]
FOR A = 0 TO 9
        (MOVE 20,28 + A*84
        FOR B = 0 TO 19
                (DECIMAL 00# A*100 + B
                ' "
                )
        )
```

Call up the first format by putting its number in the controller's table location L0. Call up the second format by putting its number in L1. By writing values to the controller's table locations G1 and G2 for the ORIGIN, and G3, G4, G5 and G6 for the WINDOW, the examples given above can be demonstrated. The effect of changing the SCALE with G0 can also be demonstrated. Refer to the item on SCALE for further details of this command.

Details of this command are given in the item for 'DRAW and RDRAW'.

FUNCTION

To produce a filled-in rectangle whose edges are parallel to the X and Y axes.

FORMAT

RECTANGLE X1,Y1,X2,Y2,n

where:
RECTANGLE is the keyword.
X1,Y1 are the co-ordinates of the first corner.
X2,Y2 are the co-ordinates of the diagonally opposite corner.
'n' is the fill pattern number.

RULES

The fill pattern number is optional. If it is omitted, the fill will be solid, and in the current FOREGROUND colour. If a fill pattern number is included, the stripes, cross hatching, etc., will be in the current FOREGROUND colour with the spaces between in the current BACKGROUND colour. For fill pattern numbers, refer to the FILL command.

As long as the co-ordinates given are for two diagonally opposite corners of the rectangle, it does not matter which corners are choosen or the order in which they are given.

If any corner of the rectangle has co-ordinates which fall outside the current WINDOW, the rectangle and its fill will be truncated at the edge of the window.

The separators between the co-ordinates, and between the last co-ordinate and the fill pattern number, obey the standard rules. Refer to the item on Separators.

When keying in, the keyword RECTANGLE may be shortened to a minimum of REC, but it will be expanded to the full keyword on any program listings or displays.

EXAMPLES

The commands:

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0 0 559 447
RECTANGLE 60 140 200 400 12
```

produce a black cross-hatched rectangle on a green background. When keying this section of program in, the keywords could be shortened as follows:-

```
FORE BLACK
BACK GREEN
WIND 0 0 559 447
REC 60 140 200 400 12
```

but on re-displaying the format or function definition, it would be listed with the keywords in full as given previously.

The following shows the use of commas and/or semicolons as separators and how the co-ordinates are paired:-

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0,0;559,447
RECTANGLE 60,140;200,400;12
```

For a solid fill, omit the final parameter:

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0 0 559 447
RECTANGLE 60 140 200 400
```

NOTES

For an open rectangle with no fill, use the DRAW or RDRAW commands.

For a rectangle whose sides are not parallel to the X and Y axes, either build it from two TRIANGLEs with a common side, or use the DRAW or RDRAW commands to produce the outline, followed by a FILL.

Where a shape is built up from a number of RECTANGLEs, TRIANGLEs and or DRAWn shapes and afterwards FILLed, any FILL pattern will continue across the adjacent or overlapping edges without dislocation.

Where a pattern is used, a RECTANGLE does not give any outline. If therefore a pattern is used that has little foreground content (e.g. FILL pattern number 1) with the same background as was shown previously, the edges of the rectangle may not show clearly. Either use a different pattern, or draw the outline using the DRAW or RDRAW commands.

FUNCTION

To return a value from a function.

FORMAT

RETURN v

where:
RETURN is the keyword.
'v' is the value to be returned.

RULES

A RETURN can be used in a format or function definition. However, it is intended for use in function definitions only. When a RETURN is encountered in a format definition, causes the format execution to terminate.

'v' may be a variable or an expression. There would be little point in making it an array variable.

If 'v' is omitted, a value of 0 (zero) is returned.

When a RETURN is encountered in a function, the function terminates, and execution returns to the calling format or function.

Where there are different paths through a function, i.e. where an IF..THEN..ELSE construction is used, more than one RETURN can be used in the function.

If a RETURN instruction is omitted from the end of a function, RETURN 0 is assumed by default.

When keying in, the keyword RETURN may be shortened to a minimum of RET, but it will be expanded to the full keyword on any program listings or displays.

EXAMPLES

```
RETURN P
```

causes a function to terminate and to return the value of its local variable P to the calling function or format.

```
IF P>1000 THEN
      RETURN 1000
ELSE
      RETURN P
```

is an example of a possible ending to a function where the value returned is limited to 1000. It shows the use of two separate RETURNs, only one of which is ever encountered on any particular execution of the function.

Details of this command are given in the item for 'MOVE and RMOVE'.

FUNCTION

To scale down the co-ordinates, character sizes and line thicknesses to a percentage of those specified in the format or function definition.

FORMAT

SCALE n

where:
SCALE is the keyword.
'n' is the scale factor, as a percentage of full size.

RULES

'n' may be a constant, variable or expression.

'n' should evaluate to a number in the range 10 to 100 inclusive. If it evaluates to a number greater than 100, it will be limited to 100. If 'n' is less than 10, or is negative, it is evaluated to 100.

NOTES

SCALE is normally used with 'n' as a variable or expression; there is little point in using the command with a constant. By varying 'n' from a lower number to a higher number, a 'zoom' effect is created which enlarges a format and allows detail that was previously hidden to be displayed.

SCALE has no effect on WINDOW commands. The co-ordinates for a WINDOW are always absolute, relative to the screen area. Refer to the item on WINDOW for more details.

SCALE has no effect on FILL patterns. These are always of the sizes shown in the item on FILL, in this Section. It does, however, affect the size of line patterns; see the item on DRAW, in this Section.

After a SCALE command is encountered in a format or function definition, all co-ordinates and SIZEs are multiplied by n/100, rounded to the nearest integer. Since the sizes of characters can only change to the nearest integer, this means that they will be 14, 28, 42 etc. pixels high for full height characters, and intermediate character heights are not obtainable. If a truncated character height has been set with the SIZE or CSIZE command, characters can only be a multiple of this truncated height.

If, as a result of the multiplication by n/100, character sizes or line thicknesses become less than 50%, they will be rounded to zero, and therefore will not appear on the display.

Where a command requires multiple parameters, e.g. one or more co-ordinate pairs, these values can generally be separated with:

- Spaces
- Commas
- Semicolons
- Any mixture of the three.

Thus, the command:

```
RECTANGLE 100 100 300 300
```

could equally well be entered as:

```
RECTANGLE 100,100,300,300
```

or as:

```
RECTANGLE 100;100;300;300
```

or as:

```
RECTANGLE 100,100;300,300
```

and so on.

The only exception is where one of the values is negative. A negative value must not be separated from a previous one with a space, but with a comma or semicolon. Thus, if the following was written:-

```
ORIGIN 100 -50
```

the video compiler would perform the subtraction, making the entry equivalent to:

```
ORIGIN 50
```

However, if the following was written:-

```
ORIGIN 100,-50
```

the compiler would interpret the two numbers as separate values.

If the command does not use constants but variables or expressions, it does not matter if these variables or expressions evaluate to negative values. Thus, if the following was written:-

```
ORIGIN P Q
```

where P and Q are variables, with a space as the separator between them, it does not matter if, on evaluation, P equals 100 and Q equals -50. Similarly, writing:

```
ORIGIN G[100] G[101]
```

if table locations G100 and G101 contain 100 and -50 respectively, this again will be interpreted correctly with a space as the separator as shown.

All numbers in the video system are treated as 32-bit words. Where a value is taken from one of the controller's data tables, it is expanded from 16-bit to 32-bit. For a positive value, leading zeros are added. For a negative value, the number is appropriately adjusted, bit 31 becoming the sign bit instead of bit 15. Besides the normal arithmetic operators (+, -, * and /), IMAGEM Display Language includes the shift operators:

>> Shift right
<< Shift left

A shift operator must be followed by a constant, variable or expression evaluating to a number between 0 and 32. Numbers outside this range will be limited to 32 giving an output of zero.

When a number is shifted to the right or to the left, any bits shifted outside the 32-bit word are lost, and vacancies are filled with zeros. If the equivalent of a rotate operation is required, an expression can be written such as:

```
Q = P>>3 UNION P<<29
```

which is equivalent to rotating right by 3 bits. The first term (P>>3) shifts P right by 3 bits, the least significant 3 bits are lost, and the most significant 3 bits are filled with zeros. The second term (P<<29) shifts P left by 29 bits, so that all but the least significant 3 bits are lost, and the least significant 29 bits are filled with zeros. Taking the UNION of the two terms combines the two to give the required answer.

Note... The keyword HIGH gives the upper byte of a table location directly, without having to use shift operators. For instance:

```
P = HIGH G[100]
```

is equivalent to:

```
P = G[100] >> 8 MASK @FF
```

FUNCTION

To provide the sine of an angle.

FORMAT

SIN (e1)

where:
SIN is the keyword.
(e1) is a value or expression.

RULES

(e1) must be in degrees.

The result given by the SIN function equals the trigonometrical sine multiplied by 1000 and rounded to the nearest integer.

EXAMPLES

SIN(45)

returns a value of 707, i.e. 0.7071 * 1000 rounded to the nearest integer.

P = SIN(G[100])

assigns to the variable P a value equal to 1000 times the sine of the value held in the controller s table location G100, rounded to the nearest integer.

FUNCTION

To define the size of characters and the thickness of lines in a display.

FORMAT

SIZE n1,n2,n3

where:
SIZE is the keyword.
'n1' is the size multiplier in the X direction for characters, and also the thickness multiplier for line thicknesses.
'n2' is the size multiplier in the Y direction for characters.
'n3' is the character height.

RULES

'n1', 'n2' and 'n3' may be constants, variables or expressions.

If 'n2' is omitted, the size of characters is multiplied by the same factor in both the X and Y directions. 'n2' has no effect on line thicknesses.

Both 'n1' and 'n2' must be in the range 0 to 32 767 inclusive.

If 'n3' is omitted, the characters are displayed in full and not truncated in height.

If characters are required truncated in height, values for 'n2' and 'n3' must be specified.

'n3' must be in the range -32 768 to 32 767. However, the expected range is between 1 and 14. Values that are negative or zero are treated as 1. Values greater than 14 are treated as 14.

If a request is made for text to be displayed or lines to be drawn before the first SIZE command is encountered in the format or function definition, the system defaults to SIZE 1,1,14.

NOTES - TEXT

Characters are stored as an array of 'off.on' elements in a 7 wide by 14 high array. Details are given in Section 6, using the Character Editor, in the item on Option 3.

When a character is displayed with SIZE 1 (or SIZE 1,1), each 'on' element in the character definition is shown by a single pixel in the current FOREGROUND colour.

When a character is displayed with enlarged SIZE n1,n2, each element in the character definition is displayed by a rectanguiar block of pixels 'n1' wide by 'n2' high. When 'n2' is omitted, each rectangular block becomes an 'n1' by 'n1' square of pixels.

When 'n3' is included, only the top 'n3' rows of the character array are displayed. This gives the facility for displaying lines of text closer together under certain conditions. For instance, upper case characters in the default set do not use the last three rows of the array, although these rows are used for lower case characters with descenders (g, j, p, q and y). Therefore, if the text uses upper case characters and digits only, try truncating the height of the characters to the first 11 rows, using, for instance, the command:

SIZE 1,1,11

Note...'n3' is the number of rows of the character definition used, not the number of pixels, and is therefore independent of 'n1' and 'n2'. Thus:

SIZE 2,3,11

gives the same effect as for the previous command but with characters twice the width and three times as high. When a NEWLINE (or an embedded line feed or carriage return) is encountered, the current writing position moves down by the current character height.

If SIZE is increased dynamically, e.g. where values are taken from the controller's data tables, the size of characters is expanded downward and to the right, and vice versa if the SIZE is reduced. In other words, the top left-hand corner of the text string remains fixed.

If a SIZE is specified that makes characters too large to display within the current WINDOW, these characters will not be displayed. In preference to being truncated at the edges of the window, they will not appear on the display at all.

NOTES - LINES

Lines drawn by the DRAW and RDRAW commands are 'n1' pixels thick. 'n2' and 'n3' do not have any effect on line thicknesses.

Compared with the single-pixel thick lines produced with SIZE 1, thicker lines are produced by extra pixels on both sides of the line. If 'n1' is an odd number, an equal number of pixels are added on each side, e.g. a SIZE 5 line would add 2 pixels on each side of a SIZE 1 line. Where 'n1' is an even number, the extra pixel will be added in the positive co-ordinate direction. Thus, a SIZE 6 horizontal line will have 3 pixels above and 2 pixels below compared with a SIZE 1 line; a SIZE 6 vertical line will have 3 pixels to the right and 2 pixels to the left compared with a SIZE 1 line.

NOTES - INTERACTION WTIH SCALE COMMAND

The SCALE command affects the SIZEs of lines and characters. Refer to the item on the SCALE command. When a SCALE command is encountered, SIZEs in the format or function definition are thereafter (until another SCALE command is encountered) multiplied by the percentage SCALE factor, and rounded to the nearest integer. The smallest percentage factor available with SCALE is 10%. If the result of multiplying the SIZE factor by the SCALE factor is less than 50%, characters and lines will not be displayed. See Examples below.

The display of text can be suppressed by setting 'n1' to zero. If a variable or expression is used for 'n1', text can be switched on or off by making 'n1' non-zero or zero. However, there are other ways of achieving the same result (such as using IF..THEN constructions) that are easier to understand when reading back the format or function definition as a display or print-out.

EXAMPLES

To display double size characters starting from pixel 100,100:

```
MOVE 100,100
SIZE 2
"DOUBLE SIZE CHARACTERS"
```

To display characters that are double width but of normal height:

```
MOVE 100,100
SIZE 2,1
"DOUBLE WIDTH CHARACTERS"
```

To draw a line 3 pixels thick:

```
MOVE 100,100
SIZE 3
DRAW 200,300
```

The characters "TEXT" produced by the following sequence of commands:-

```
SCALE 30
MOVE 200,100
SIZE 4
"TEXT"
```

will be of SIZE 1, since SIZE 4 multiplied by a SCALE factor of 30% gives a size of 1.2 that, when rounded to the nearest integer, is a size of 1. However, with the following commands:-

```
SCALE 15
MOVE 120,300
SIZE 3
 TEXT"
```

the character string "TEXT" will not be displayed, since SIZE 3 multiplied by a SCALE factor of 15% gives a size of 0.45. This is less than 50%, and when rounded to the nearest integer gives a size of 0 (zero), so nothing is displayed.

The following commands cause characters of variable size to be displayed according to the value held in the controller's table location G50:

```
MOVE 0,28
SIZE G[50]
"TEXT"
```

and text will not be displayed if G50 contains zero.

FUNCTION

To display a text message taken from the controller s data tables.

FORMAT

STRING t1,n,t2...

where:
STRING is the keyword.
t1' is the controller's table location at which the message or a chain of messages start.
n' is the message number.
t2 is the start of the controller's table locations whose values are to be embedded in the message from a GEM80 Controller other than a 200 series.

RULES

't1' can have one of the following formats:-

1. a[e] **Example: G[100]**

where 'a' is the table letter, and 'e' is a constant, variable or expression evaluating to the table location number.

2. [e] **Example: [35232]**

where 'e' is the absolute byte address in the memory of the table location. This could be found in the controller's program using the LOCATE Special Function S20. It could also be found within the format or function definition in the video system using the function LOC. Refer to the item on LOC.

3. [a[e]] **Example: [G[230]]**

where 'a[e]' is a table location using the same notation as for 1. above. However, instead of this table location being the one where the message starts, it is a table location whose contents are the absolute byte address of where the message starts, as in 2. above. In other words, a[e] is a location whose contents point to where the message starts.

'n' is a constant, variable or expression defining the message number.

't2' can have any of the same formats as for t1'. 't2' is only required for GEM80 Controller messages on controllers other than 200 series that have embedded values and otherwise are ignored. When 't2' is required, 'n' must always be included in the command.

NOTES

For the format of data in the controller's data tables for messages, refer to the item on Messages.

Note...For 't1', 't2', it is possible to further extend the use of square brackets, to use table locations whose contents point to table locations that point to other table locations, etc. This is described in the item on Variables.

EXAMPLES

STRING W[100]

displays the message beginning at table location W100.

```
STRING W[100],3
```

displays the third message in a chain of messages, where the first message begins at table location W100.

```
STRING [X[0]],1
```

displays the first message in a GEM80 Controller other than a 200 series taken from the messages area in P-table. This example uses the special pseudo table location X[0] that points to where the message area starts (refer to the items on Messages and Variables). A message number must always be given when using a STRING with X[0].

```
STRING [X[0]],5
```

displays the fifth message in a GEM80 Controller other than a 200 series.

```
STRING [X[0]],4,A[13]
```

displays the fourth message in a GEM80 Controller other than a 200 series, incorporating into the display a value taken from table location A13. The format of the display depends on the field descriptor used in the message in the controller. These field descriptors, compatible with those used by the video system for the DECIMAL command, are described in the Controller's Technical Manual.

```
STRING [X[0]],G[35],A[G[35]+5]
```

displays a message in a GEM80 Controller other than a 200 series, where the message number is variable and taken from table location G35. Incorporated in the message are two values, the first taken from A-table and the second from the next consecutive A-table. The table location numbers are evaluated by expressions dependent, in this example, on the message number taken from G35.

It is not possible to enter more than one variable parameter after the message number for output of numeric data within the message i.e.

```
STRING [X[0]],1000,E[2],E[1]
```

will result in a compiler error, producing the error message "Compiler Error F9 = 137 Illegal Character Found"

FIXED TEXT

To display fixed text in the format or function definition, simply specify the text, enclosed in either single or double quotes, without any keyword, e.g.

```
"PIECE OF TEXT"
```

The size of the characters can be enlarged by using the SIZE or CSIZE command preceding the text. See the items on SIZE and CSIZE.

The text will appear, starting at the top left-hand corner of the current WINDOW, unless the starting position has already been moved using MOVE or RMOVE. Further text will carry on where the last text string finished, e.g. the commands:

```
"PIECE OF TEXT"
"SECOND PIECE OF TEXT"
```

will produce a display of:

```
PIECE OF TEXTSECOND PIECE OF TEXT
```

To move the writing position to the next line:

- Type a carriage-return or line-feed inside the quotes. The system will provide the other, converting what is typed into a CR:LF sequence.

- Preferably, use a NEWLINE command, as below:-

```
"PIECE OF TEXT"
NEWLINE
"SECOND PIECE OF TEXT"
```

which will produce a display of:

```
PIECE OF TEXT
SECOND PIECE OF TEXT
```

Note...Although text can be entered between single or double quotes, it is always re-displayed enclosed in double quotes.

There are facilities for producing a set of up to 224 different character types, see Section 6, 'Using the Character Editor'. To include any special characters which do not have corresponding keys on the programmer, their hexadecimal codes must be given, preceded by the 'commercial at' symbol '@', for instance:

```
"Text containing user defined characters @F3@91"
```

Control characters can also be included, i.e. those with hexadecimal codes 00 to 1F which do not display symbols but could affect the display. There is no particular reason why this should be done since all control characters are ignored except for newline code, 0D hexadecimal.

If any of the following is to be included within the text:-

- Double quotes (")
- Single quotes (')
- Commercial at (@)

these symbols must be repeated. For instance:

```
" ' 'PIECE@@OF@@TEXT@@IN@@SINGLE@@QUOTES' ' "
```

will produce a display of:

PIECE@OF@TEXT@IN@SINGLE@QUOTES'

VARIABLE TEXT

There are two commands in the IMAGEM Display Language for handling text taken from the controller's tables:

- STRING. Intended for variable text messages.
- CHAR. Intended for individual characters. Can be used for longer strings, but STRING is generally more convenient for this purpose.

Refer to the items relating to these commands for details.

There is also a command for handling text taken from the off-line storage section of memory on controllers that have this facility:

- GETCHAR. Intended, like CHAR, for individual characters, but can be used for longer strings.

Refer to the item relating to this command for details.

NUMBERS

There are three commands for displaying numbers:

- DECIMAL. Intended for signed decimal numbers.
- UNSIGNED. As for DECIMAL but treating all numbers as positive.
- HEX. Displays numbers in hexadecimal form, with each four bits of the number being treated as one hexadecimal digit (0-9 and A-F).

Refer to the items relating to these commands for details.

COMBINED NUMBERS AND TEXT

To display text and numbers, write the commands in order, e.g.

```
"FEEDWATER TEMPERATURE = '
DECIMAL ^^#.## G[340]
' Degrees C'
```

will give a display, if G340 contains 7435, of:

FEEDWATER TEMPERATURE = 74.35 Degrees C

Details of this keyword are given in the item for 'IF..THEN..ELSE'.

Details of this keyword are given in the item on 'FOR..TO'.

FUNCTION

To display text on top of previously defined text, i.e. the equivalent of a printer 'overstrike' facility. Also allows patterned lines to be drawn with the gaps between the dashes not overwriting anything which was previously displayed.

FORMAT

TRANSPARENT

where:
TRANSPARENT is the keyword

RULES

Although TRANSPARENT may seem at first sight to be a colour, it is not, and must not be preceded by the keyword BACKGROUND. It affects characters and drawn lines only, and has no effect on patterned FILLs.

NOTES

When text is displayed, each character is shown on the display in a 7 by 14 array of elements. The elements that are 'on' in the character definition are shown in the current FOREGROUND colour, and the elements that are 'off' are shown in the current BACKGROUND colour. However, if TRANSPARENT is specified, new text is shown over the top of that previously displayed. That is, only those pixels defined by the 'on' elements in the character definition array are altered. Pixels defined by 'off' elements in the character definition array are left unchanged in whatever BACKGROUND or FOREGROUND colour they were previously shown.

EXAMPLES

Without using TRANSPARENT:

```
FOREGROUND BLACK
BACKGROUND WHITE
MOVE 100,100
"TEXT"
MOVE 100,100
"    "
 ----
```

produces a display of:

```
    -----
```

because TEXT is overwritten by both the background and foreground colours of . ... However:

```
FOREGROUND BLACK
BACKGROUND WHITE
MOVE 100,100
"TEXT"
TRANSPARENT
MOVE 100,100
"    "
 ----
```

produces a display of:

TEXT

FUNCTION

To produce a filled-in triangle.

FORMAT

TRIANGLE X1,Y1,X2,Y2,X3,Y3,n

where:
TRIANGLE is the keyword.
X1,Y1 are the co-ordinates of the first corner.
X2,Y2 are the co-ordinates of the second corner.
X3,Y3 are the co-ordinates of the third corner.
'n' is the fill pattern number.

RULES

The fill pattern number is optional. If it is omitted, the fill will be solid, and in the current FOREGROUND colour. If a fill pattern number is included, the stripes, cross hatching, etc., will be in the current FOREGROUND colour with the in-between spaces in the current BACKGROUND colour. For fill pattern numbers, refer to the FILL command.

If any corner of the triangle has co-ordinates which fall outside the current WINDOW, the triangle and its fill will be truncated at the edge of the window.

The separators between the co-ordinates, and between the last co-ordinate and the fill pattern number, obey the standard rules. Refer to the item on Separators.

When keying in, the keyword TRIANGLE may be shortened to a minimum of TRI, but it will be expanded to the full keyword on any program listings or displays.

EXAMPLES

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0 0 559 447
TRIANGLE 60 140 60 60 99 90 12
```

produces a black cross-hatched triangle on a green background. When keying this section of program in, the keywords could be shortened as follows:-

```
FORE BLACK
BACK GREEN
WIND 0 0 559 447
TRI 60 140 60 60 99 90 12
```

on re-displaying the format or function definition, it would be listed with the keywords in full as given previously.

Commas and/or semicolons can be used as separators to show how the co-ordinates are paired, as follows:-

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0,0;559,447
TRIANGLE 60,140;60,60;99,90;12
```

For a solid fill, omit the final parameter:

```
FOREGROUND BLACK
BACKGROUND GREEN
WINDOW 0 0 559 447
TRIANGLE 60 140 60 60 99 90
```

NOTES

For an open triangle without a fill, use the DRAW or RDRAW commands.

Where a shape is built up from a number of RECTANGLEs, TRIANGLEs and.or DRAWn shapes afterwards FILLed, any FILL pattern will be continued across the adjacent or overlapping edges without dislocation.

Where a pattern is used, a TRIANGLE does not give any outline. If a pattern that has little foreground content is used, e.g. FILL pattern number 1, with the same background as shown previously, the edges of the triangle may not show clearly. Either use a different pattern, or draw the outline using the DRAW or RDRAW commands.

FUNCTION

Bitwise Or operator.

EXAMPLES

The following command:-

```
IF G[50] UNION G[51] <>0 THEN FOREGROUND REDFLASH
```

sets the FOREGROUND to REDFLASH if any of the bits in the controller's table locations G50 or G51 are set to 1.

UNION can also be used in expressions to produce a value that consists of 1's, where two variables both contain 1 for the same bit numbers, e.g.

```
C = A UNION B
```

would perform a calculation such as the following:-

```
A    0011 1100 1010 0101
B    1111 0000 1111 0000
     ---------------------
C    1111 1100 1111 0101  = A UNION B
```

NOTES

For brevity, the examples above have been shown with 16-bit numbers. In practice, simple variables are 32-bit, and values taken from the controller's table locations are 16-bit padded to out to 32-bit (with leading zeros if positive, or suitably adjusted if negative).

PRIORITY

When an expression is evaluated, HIGH, LOW, Shift and MASK are evaluated before UNION. DIFFER, multiplication, division, addition and subtraction are evaluated after UNION.

FUNCTION

To display numeric data in unsigned decimal form.

FORMAT

UNSIGNED f,n

where:
UNSIGNED is the keyword.
'f' specifies the required field width and type.
'n' specifies the data for display.

FIELD DESCRIPTOR

The field descriptor 'f' represents, in pictorial form, the way in which the numeric data is to be displayed. The symbols used are identical to those used for message definitions in GEM80 Controllers other than 200 series, and are as given below:-

Symbol	Name	Meaning
^	Caret	Leading space
0	Zero	Leading zero
#	Hash	Significant digit
.	Full stop or period	Decimal point

NOTES

If the contents of one of the controller's table locations is displayed, UNSIGNED will treat the contents as being in the range 0 to 4294967295 inclusive, rather than the range -32768 to 32767. Numbers which are entered or monitored with a programmer plugged into the controller's processor module(s) as negative numbers will generally be meaningless when displayed by UNSIGNED. If numbers are to be displayed with signs, use the DECIMAL command. The primary use of UNSIGNED is for long numbers, longer than those held in the controller's table locations, that would generally be held in the video system as simple variables.

EXAMPLES

UNSIGNED ^^^^#.# G[25]

will display the contents of the controller's table location G25 in unsigned form, with one decimal place, and with up to five leading digits.

UNSIGNED 0000#.# G[25]

is essentially the same, except that leading zeros are displayed.

The following table shows how these examples would display for different contents of table location G25:-

G25 contents	Display	
	First example	Second example
1	0.1	00000.1
12	1.2	00001.2
123	12.3	00012.3
1234	123.4	00123.4
12345	1234.5	01234.5

```
UNSIGNED ^^^^#.## P
```

displays the value of the simple variable P in unsigned form, catering for up to 7 digits, with 2 digits after the decimal point.

For example, a long number could be displayed by writing the following:-

```
UNSIGNED ^^^^^^^^^#.### G[25]+10000*G[26]
```

which would take the contents of table location G26, multiply it by 10000, add this to the contents of table location G25, and then display the result.

RULES

The data for display 'n' may be a constant, variable or expression.

If the number contains more digits than will fit into the space allocated in the field descriptor, the display takes the form of question marks presented in the specified layout.

The maximum quantity of digits that can be specified with a field descriptor is 15.

The size of characters displayed, as for Text, is determined by the last SIZE or CSIZE command encountered. Refer to the items on SIZE and CSIZE.

SIMPLE VARIABLES

Within any format or function definition, the single letters A to Z inclusive can be used to represent variables. Each variable is treated as a 32-bit integer number. The rules which apply to these 26 possible variables are:

- All variables are set to zero when a format or function is first called up.
- Once a format is being used in a display, the variables are not re-set to zero. Each time a format definition is processed, the variables retain their values when the processing is completed, and so can carry count values over from one execution to the next.
- Each format definition has its own set of 26 variables. Where a display consists of more than one format, if the variable C was used in the first format, its value would be entirely independent of the variable C used in the second format.

Three simple variables have pre-defined usage as follows:-

- V This is set to the L-table number that called up the particular format. V is set to this number when the format definition is executed. This value could subsequently be overwritten by assigning some other value to V.
- X,Y These two variables hold the X and Y co-ordinates of the current writing position. Their values therefore change as the format definition is being processed.

ARRAY VARIABLES

The contents of the controller's tables are represented by the same table letters and numbers as used in the controller, except with square brackets enclosing the number. For example, if the value contained in the controller's table location G100 is to be used, the value is accessed by the video system as the variable G[100].

In common with simple variables, array variables are 32-bit integers. However, since the values in the controller's tables are only 16-bit words, the equivalent variables in the video system are padded out to 32 bits. For positive values, this is done by adding leading zeros. For negative values, the numbers are padded out with leading 1's because of the different position of the sign bit.

Thus:

0101 0101 0101 0101 becomes 0000 0000 0000 0000 0101 0101 0101 0101

and:

1010 1010 1010 1010 becomes 1111 1111 1111 1111 1010 1010 1010 1010

A table location's contents can be accessed by using an expression, rather than just a constant, within the square brackets, e.g. the following could be used:-

G[D + 4]

In this instance, if the variable D was made equal to 12, the variable G[D + 4] would contain the value held in the controller s table location G16.

RESTRICTIONS ON TABLE ACCESS

For details regarding which of the controller's tables are accessible by a video system, refer to the Product Data Sheet, User Information leaflet or Technical Manual for the particular model of controller. For instance, the F-table and R-table in a GEM80/200 Series controller are not accessible by a video.

In addition, with a 200 series Controller, the P-table is only accessible up to the last P-table location actually declared in the controller's program. In certain cases, it may be necessary to insert a dummy rung or instruction which includes a reference to the highest P-table location needed by the IMAGEM video system.

ASSIGNMENT

For assigning a value to a simple variable, the equals sign is used, e.g.

```
C = 5
```

sets the simple variable C equal to 5. Alternatively the keyword LET can be used, e.g.:

```
LET C = 5
```

Values can only be assigned to simple variables, and to the L-table array variable. If an attempt is made to assign a value to any other array variable, a compiler error message is displayed. This message will not always be the same, since it depends on the context in which the compiler found the array variable.

Within a function or format definition, the values of array variables are read from the controller's data tables as and when they occur in the program. If the same array variable is used more than once, its value could be different on different occurrences, as the controller might have updated the contents of its data tables between successive read operations. For consistent results, it is recommended that, in such cases, the value is assigned to a simple variable, e.g.

```
D = W[234]
```

sets the simple variable D equal to the contents of the controller's table location W234. The variable D can then be used several times in the format or function definition. It will always have the same value throughout the definition on any particular execution, whereas if W[234] is used several times, its value might change. A second advantage of this technique is that W234 has to be read from the controller's data tables only once, rather than several times, so that processing the format or function definition will be faster.

L ARRAY VARIABLE ASSIGNMENT

When a value is assigned to an L array variable, it is in effect, writing data back to the controller's L-table. This is the only table in the controller's memory that the IMAGEM video system can write to. When assigning a value to an L array variable, the keyword LET must always be used.

Since the L-table in the controller stores data in 16-bit words, so does the L array variable. The range of values that can be used is therefore -32768 to +32767. If an attempt is made to assign a value outside this range to an L array variable, the least significant 16 bits of the value are used, and an overflow flag is set.

The overflow flag consists of a bit in the L-table location after the video frames counter, i.e. in:

- L17 for Video 0
- L21 for Video 1
- L25 for Video 2
- L29 for Video 3

The bit set is the first, second, third or fourth bit in the L-table word, depending on which format caused the overflow error. If an overflow error occurs in more than one format, more than one of these bits will be set at the same time. For instance, if Format 23 contains the statement:

```
LET L[30] = 100000
```

(where 100000 is too large to be represented by a 16-bit number and will cause overflow), and if Format 23 is specified by the controller's table location L9 (to make it the second format on the second video), bit 1 in L21 will be set to 1. In the absence of any other errors, this assigns a value of 2 to the contents of L21. If there was also an overflow error in the format called up by the controller's table location L8, bit 0 in L21 would also be set to 1, making the contents of L21 equal to 3.

Note...These bits in the controller's L17, L21, L25 and L29 tables are not set back to 0 (zero) by IMAGEM when the overflow condition disappears. Instead, they must be re-set by means of the ladder diagram program in the controller, or by assigning zero to them using a LET instruction from the IMAGEM. For instance:

```
LET L[21] = 0
```

would re-set all overflow flags for Video 1.

If an overflow occurs while editing an IMAGEM format or function definition, this will set bit 3 in the flag word for that video, making the value of its contents equal to 8. The effect is the same as if an overflow occurred in the fourth format during running on that video.

Details of this keyword are given in the item for 'DRAW and RDRAW'

FUNCTION

To tell a video format definition the number of the IMAGEM processor that is executing it.

FORMAT

VID

RULES

VID may be treated in expressions as a simple variable. Its value will be 0, 1, 2 or 3 depending on which IMAGEM processor is executing the format definition.

Do not try to assign any value to VID; if attempted, a function out of context error message will be output from the compiler.

EXAMPLES

```
DECIMAL ^# VID
```

will display the current video number.

```
LET L[VID*20+32+A] = 27
```

could be used, provided that A was in the range 1 to 20 inclusive, to ensure that data from a particular IMAGEM always arrived in specific L-table locations.

FUNCTION

To define the area of the screen into which a format is to fit.

FORMAT

WINDOW X1,Y1,X2,Y2

where:
WINDOW is the keyword.
X1,Y1 are the co-ordinates of the first corner of the window.
X2,Y2 are the co-ordinates of a diagonally opposite corner.

RULES

As long as the co-ordinates given are for two diagonally opposite corners of the window, it does not matter which corners are chosen or the order in which they are given.

The co-ordinates may be constants, variables or expressions.

The co-ordinates given must be relative to the screen area itself. The bottom left-hand pixel in the screen area has co-ordinates 0,0 and the top right-hand pixel has co-ordinates 559,447.

If either X co-ordinate is given as, or evaluates to, a number larger than 559, it will be limited to 559. Similarly, if it is given as, or evaluates to, a negative number, it will be limited to 0.

Similarly, Y co-ordinates will be limited to the range 0 to 447. However, X and Y co-ordinates could be specified in the range -32768 to 32767.

The separators between the co-ordinates obey the standard rules. Refer to the item on Separators.

When keying in, the keyword WINDOW may be shortened to a minimum of WIN, but it will be expanded to the full keyword on any program listings or displays.

NOTES

Each time a new format is called up for display, the window is initially set to the whole screen area, and only changes when a WINDOW command is encountered in the format definition. If a WINDOW command is not included, the window remains set to the whole screen area.

If the format is not going to cover the whole screen area, a BACKGROUND colour command must be included before using the WINDOW command in the format definition if the background colour for the window is to be different from its previous colour.

After a WINDOW command is encountered, all further commands using co-ordinates (like DRAW and MOVE) will take their co-ordinates as relative to the origin of the window, rather than to the origin of the screen. The bottom left-hand pixel of the window is taken as the origin with 0,0 co-ordinates, unless the origin is re-defined. To do this, use the ORIGIN command.

It may be found useful to specify a format definition with X and Y co-ordinates outside the WINDOW area. By using a SCALE command, the co-ordinates of the format definition can be reduced to a minimum of 10%, and thus bring parts of the format or function that were previously outside the WINDOW into the WINDOW area. Refer to the item on the SCALE command.

USING THE FORMAT EDITIOR

This page left intentionally blank

INTRODUCTION

To enter the editor, select option '2' on the video main menu. A prompt is given to enter the format number or function name.

- A valid format number is any number in the range 1 to 32767.

- A valid function name is one that starts with an alphabetic character, followed by one to seven further alphabetic or numeric characters. The minimum length of function name is two characters. The maximum length allowed is eight characters.

  Note... Spaces are not allowed, and case is ignored, i.e. upper case and lower case letters are not distinguished.

If the name entered is valid but no previous entry for this name has been made, the message 'New file' is output, and an empty file is set up in IMAGEM, ready to accept the format or function definition. If the name is that of an existing format or function, that format or function is de-compiled, and the resultant text set up as a file in IMAGEM, ready for editing.

When the editing session is finished, there is the option of directing the output to a format or function different from that used for input.

EDITOR COMMANDS

The editor commands are all single character commands which can be entered in either upper or lower case. Whenever the editor is ready to accept a command or series of commands, a prompt character is output. This takes the form of '>'.

Multiple commands can be entered on the same line. In this case, the commands are executed one at a time from left to right, in exactly the same way as they would be if each had been entered on a separate line.

A single 'Escape' (ESC) character (displayed as $) can be used at any point between commands as a separator character. When data is included with a command, the data must be separated from the next command with an ESC character. If this is not done, the editor has no way of knowing when the data stops and the next command begins. Without the separator character, everything is considered data. The ESC character is also used as a separator between multiple data fields in a command.

Two ESC characters (displayed as $$) are used to terminate any command or series of commands.

NOTATION

In the following pages in this Section which detail the editor commands, square brackets are used to enclose those parts of a command which are optional. The square brackets themselves must *not* be keyed in. This will become clear from the examples.

Where asked to type, for instance, Control-B, this means that the Control key should be held down while, at the same time, pressing B.

COMMANDS AVAILABLE

Text commands

The editor contains commands to enter text into a file, change existing text, delete an entire line, and delete a single character, as follows:-

I Insert text.

S Substitute text.

D Delete characters.

K Kill lines of text.

W Window editor. Usable with System Programmers only. Allows direct insertion and deletion of text without using the I and D commands. Displays the pointer as a visible cursor, which can be moved using the cursor control (arrow) keys instead of using the B, Z, L and C commands.

Pointer movement commands

The editor keeps a pointer in the data in a file at all times. When many of the editor commands are issued, the action takes place from where the pointer is located. For example, the Type command (T) types the data from the pointer location to the end of a line. The Insert command (I) inserts data into a file between the pointer location and the previous character. All the text commands listed above affect the pointer and the effect described in each command description.

To update a file, it must be possible to move the pointer in the file. The editor has five pointer manipulation commands:

B Beginning of Text. Moves the pointer to the first character in the file.

Z End of Text. Moves the pointer just beyond the last character in the file.

L Line. Moves the pointer to the first character in a line.

C Character. Moves the pointer one character on in the file.

F Find. Moves the pointer to the character next after the character of a data string supplied in the command.

Typing a file

T Types the file, i.e. displays it on the programmer screen.

Control commands

The editor contains commands that have no effect on the contents of the format or function file, but provide useful information.

V View. Inhibits/enables the results of the format or function definition to be displayed on the video monitor screen.

? Help. Provides a brief explanation of the use of keywords.

M Memory. Displays the number of bytes of memory still available.

Terminating a session and saving your file

Two commands are provided to terminate an editing session and return to the user interface level. These are:

E Exit. Saves the format or function definition file by compiling it and copying it into the external memory.

Q Quit. Returns to the user interface level without making any changes to what is held in the external memory.

COMMAND STRING ITERATIONS

Command strings can be entered where commands are asked to be repeated a specified number of times. The last page of this Section gives details of how to achieve this.

MULTIPLE VIDEOS

If a GEM80 Controller contains more than one IMAGEM Graphic Video system, and there are programmers connected to more than one video at the same time, a format or function cannot be edited if it is being loaded or edited simultaneously by another video.

FUNCTION

To enable a short message to be displayed, giving the parameters reqired, options, etc., associated with a command keyword.

FORMAT

The format of the ? command is:

?$$

where:
? is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key has been pressed twice to terminate the command.

NOTES

In use, the editor inspects the character under the text pointer. If this is not an alphabetic character, the editor back-tracks until one is found. It then back-tracks to the beginning of the word containing that character. This word is then matched against the IMAGEM Display Language keywords. If a match is found, the associated help message is output. If no match is found, or if no word to match is found, the message:

'Cannot find keyword'

is output, and the keywords and in-built functions are listed.

POINTER POSITION

The ? command does not move the pointer.

FUNCTION

To move the pointer to the first character in the file.

FORMAT

The format of the B command is:

B$$

where:
B is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key,is pressed twice to terminate the command.

EXAMPLES

To move the pointer to the beginning of a file:

B$$

To type the first line, i.e. display it on the programmer, add the Type command (T) (described below) to the B command:

BT$$

The pointer moves to the beginning of the file, then the first line is typed on the programmer.

NOTES

The B command is useful to:

- move the pointer to the beginning of a file when the whole file is to be typed (see T command).
- establish a starting point when searching a whole file to find a certain character string (see F and S commands).
- insert text in front of the existing text (see I command).

When entering the editor, the pointer is already set to the beginning of the file so that the B command is not necessary.

FUNCTION

To move the pointer a specified number of characters forward or backward in the file.

FORMAT

The format of the C command is:

[n]C$$

where:
'n' specifies the number of characters the pointer is to be moved.
C is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

A positive 'n' specifies a forward movement.

A negative 'n' specifies a backward movement.

If 'n' is omitted, 1 is assumed.

If a minus sign only is used, -1 is assumed.

'n' must lie within the range -32768 to 32767 inclusive.

An unsigned number is assumed to be positive, i.e. it is not necessary to use a plus sign.

EXAMPLES

To move the pointer to the next character in the file:

C$$

To move the pointer forward 10 characters:

10C$$

To move the pointer backward 100 characters:

-100C$$

To move the pointer backward one character:

-1C$$ or -C$$

NOTES

If the beginning or end of the file is encountered before the specified number is reached, the pointer stops at the beginning or end of the file respectively.

FUNCTION

To delete a specified number of characters, starting at the pointer.

FORMAT

The format of the D command is:

['n]D$$

where:
'n specifies the number of characters to be deleted.
D is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

If 'n' is positive, the deletion is performed in a forward direction from the pointer, including the character at the pointer.

If 'n' is negative, the deletion is performed in a backward direction from the pointer, not including the character at the pointer.

If 'n' is omitted, 1 is assumed.

If a minus sign only is used, -1 is assumed.

'n' must lie within the range -32768 to 32767 inclusive.

An unsigned number is assumed to be positive, i.e. a plus sign is not necessary.

EXAMPLES

To delete the next 10 characters following the current pointer location, including the character at the pointer:

10D$$

To delete the 10 characters preceding the current pointer location:

-10D$$

To delete the character preceding the current pointer location:

-D$$ or -1D$$

POINTER POSITION

If the deletion is performed in a forward direction, the pointer finishes up immediately preceding the first character not deleted.

If the deletion is performed in a backward direction, the pointer finishes up immediately following the first character not deleted.

FUNCTION

To exit from the editor, saving all changes.

FORMAT

The format of the E command is:

E[destination]$$

where:
E is the command code.
'destination' is the number or name the edited format or function will have when stored.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

If 'destination' is a valid format number or function name, the format or function will be saved under this name. If a format or function of this name already exists, it will be over-written.

If 'destination' is an invalid format number or function name, a prompt is given for a valid one.

If 'destination' is omitted, the edited format or function over-writes the original.

EXAMPLES

To store the edited format or function in the file where it came from:

E$$

To store the edited format or function called 'Copper mill', the name has to be truncated to no more than 8 characters in length.

ECoppermi$$

If more than 8 characters are entered, the name must be retyped and re-entered.

NOTES

The E command causes the contents of the edit file to be compiled to the internal form, and then transfers the resulting code to the external store. If a format or function of the same name already exists in external store, it will be over-written.

If an error is found during compilation, an appropriate error message is output and the command rejected. In this case, no change is made to external store and control remains with the editor. The text pointer is left pointing at the first character on the line in which the fault was found and this line is typed on the programmer. Whether or not the view facility was enabled (see V command), the format will be displayed up to, but excluding, the error, and the cross-wire cursor will be at this position.

If the format or function compiles correctly but there is insufficient room in the external store to store it, an appropriate error message will be output. No change is made to the contents of the external store, and control returns to the editor. In this case either use the editor to shorten the format or function, or quit (see Q command).

FUNCTION

To find a specified character string by searching through the file.

FORMAT

The format of the F command is:

[n]Ftext$$

where:
'n' specifies the number of times the search is to be done.
F is the command code.
'text' is the string of characters to be found
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

A positive 'n' specifies a forward search, from the current pointer location toward the end of the file.

A negative 'n' is not allowed, and will result in an error message.

If 'n' is omitted, 1 is assumed.

'n' must lie within the range 1 to 32767 inclusive.

EXAMPLES

To find the character string 'LOOP1' starting from the current location of the pointer:

FLOOP1$$

To find the same character string, but starting the search from the beginning of the file:

BFLOOP1$$

To find the same character string starting from the beginning of the file, and then to move the pointer to the beginning of the line containing the string:

BFLOOP1$0L$$

In this example, if the ESC character after the string had not been entered, so that the command series was:

BLOOP10L$$

the editor would have searched for the string 'LOOP10L' and presumably would not have found it.

NOTES

The search begins at the current location of the pointer and continues until the string is found or the end of the file is encountered.

The specified string will only be found if there is an exact match. For an exact match, all characters (both printing and non-printing, such as newline) must agree, but cases are ignored.

If the F command is included in a command string, an ESC character must be used to terminate the text string, as shown in the last example above. Otherwise, the next command will be used as part of the text string.

POINTER POSITION

If the character string is found, the pointer is moved to the next character after the last character of the string.

If the character string is not found, the pointer is not moved from its current location and the following message is displayed:-

Cannot find "text"

where "text" is the character string specified in the command.

FUNCTION

To insert text into the file between the character being pointed at and the preceding character.

FORMAT

The format of the I command is:

[n]Itext$$

where:
'n' specifies the number of times the data is to be inserted.
I is the command code.
'text' is the data to be inserted.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

'n' must lie within the range 1 to 32767 inclusive.

A negative 'n' is not allowed, and results in an error message.

If 'n' is omitted, it is assumed to be 1.

EXAMPLES

To enter the single character 'K' at the pointer:

IK$$

To enter a line at the beginning of the file:

BIApril is the cruelest month, breeding <CR>$$

To enter more than one line at the end of an existing file:

ZILilacs out of the dead land, mixing <CR>
Memory and desire, stirring <CR>
Dull roots with spring rain. <CR>
Winter kept us warm, covering <CR>
Earth in forgetful snow, feeding <CR>
A little life with dried tubers. <CR>$$

To enter six identical lines after the second occurrence of 'hello':

B2Fhello$6Ithe line to be entered <CR>$$

NOTES

If the text pointer is pointing to the first character in the file, the text is entered at the start of the file.

If the text pointer is at the end of the file, following a Z command, the text is entered at the end of the file.

During initial entry of data into a file, the I command normally goes on for many lines.

All data following the I command is inserted into the file until an ESC is pressed. This includes newline characters (marked 'CR' on the keyboard). If no more room is available, the following message is displayed:-

Insert has failed due to insufficient space

POINTER POSITION

When the command is completed, the pointer is at the next character after the inserted data.

FUNCTION

To delete all the characters in a line beginning at the current location of the pointer.

FORMAT

The format of the K command is:

[n]K$$

where:
'n' specifies the number of lines to be deleted.
K is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

If 'n' is positive, the deletion is performed in a forward direction, starting with the character at the pointer.

If 'n' is negative, the deletion is performed in a backward direction, starting with the character immediately before the pointer.

If 'n' is omitted, 1 is assumed.

If a minus sign only is used, -1 is assumed

'n' must lie within the range -32768 to 32767 inclusive.

An unsigned number is assumed to be positive, i.e. a plus sign is not necessary.

EXAMPLES

To delete an entire line, ensure that the pointer is at the beginning of the line:

0LK$$

To delete all the characters in a line, starting with the one at the pointer and including the newline character at the end:

K$$

To delete the entire line in which the pointer is located and the three lines that follow it:

0L4K$$

NOTES

When deleting forwards, the characters in the line that precede the pointer are not deleted.

The K command also deletes the newline character at the end of the line.

POINTER POSITION

The pointer is left at the first character after the place where text has been removed.

FUNCTION

To move the pointer to the first character of the next line in the file.

FORMAT

The format of the L command is:

[n]L$$

where:
'n' specifies the number of lines the pointer is to be moved.
L is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

A positive 'n' specifies a forward movement.

A negative 'n' specifies a backward movement.

0 (zero) makes the pointer move to the beginning of the current line.

If 'n' is omitted, 1 is assumed.

If a minus sign only is used, -1 is assumed.

'n' must lie within the range -32768 to 32767 inclusive.

An unsigned number is assumed to be positive, i.e. a plus sign is not necessary.

EXAMPLES

To move the pointer to the beginning of the next line:

L$$

To move the pointer to the beginning of the line five lines back:

-5L$$

To move the pointer to the beginning of the current line:

0L$$

To move the pointer ahead 12 lines:

12L$$

FUNCTION

To compute and display on the programmer, the number of bytes of memory still available in IMAGEM's internal store.

FORMAT

The format of the M command is:

M$$

where:
M is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

NOTES

The number of bytes remaining gives only a rough estimate of how much room is left for extending the format or function definition. This is because what is held in memory as commands that have been entered requires further memory space when it is compiled. Therefore, although there is room to insert more commands, the format or function definition will fail to compile because of insufficient space.

EXAMPLE

To check on the amount of storage available:

M$$

The information is supplied in the following message:-

'nnnn bytes remain'

where 'nnnn' is a decimal integer.

POINTER POSITION

The M command does not move the pointer.

FUNCTION

To exit from the editor, without saving any changes made during the editing session.

FORMAT

The format of the Q command is:

Q$$

where:
Q is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

NOTES

To exit from the editor and save all changes, use the E command.

By using the Q command, it is possible to abandon the editor and return to the user interface level. In order to avoid accidentally discarding data, a prompt asks for confirmation of the command before execution takes place, unless the file has not been altered by the editor (i.e. none of the D, I, K, S, or W commands have been used).

If this is confirmed, any data held in the edit file will be discarded. If the command is not confirmed, control remains with the editor. In either case, external store remains as it was before the editor was invoked.

FUNCTION

To find a character string and substitute another character string in its place.

The command can also be used to delete a specified character string.

FORMAT

The format of the S command is:

[n]Sold-text[$new-text]$$

where:
'n' specifies how many occurrences of the string 'old-text' are to be replaced by the string 'new-text'.
S is the command code.
'old-text' is the string to be searched for and replaced if found.
'new-text' is the character string to replace 'old-text'.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

A positive 'n' specifies a forward search, from the current pointer location toward the end of the file.

A negative 'n' is not allowed, and will result in an error message.

If 'n' is omitted, 1 is assumed.

'n' must lie within the range 1 to 32767 inclusive.

If 'new-text' is omitted, the character string specified by 'old-text' is deleted.

EXAMPLES

To replace the string 'RED' with the string 'BLUE', starting the search from the pointer position:

SRED$BLUE$$

To make the same substitution but starting the search from the beginning of the file:

BSRED$BLUE$$

To make the same substitution, searching from the beginning and then typing the entire line in which the change was made:

BSRED$BLUE$0LT$$

To delete the first occurrence of the string 'GREEN' from a file:

BSGREEN$$

In the above example, replacement data is not supplied with the command. Thus, the command will find the string and delete it.

To change a file containing only 'This, that and the other' so that it contains 'Dis, dat and de udder':

BSThD2Sth$d$Soth$udd$$

NOTES

The search begins at the current location of the pointer and continues until the 'old-text' string is found or the end of the file is encountered. A substitution is made only if the search finds an exact match. For an exact match, all characters (both printing and non-printing such as newline) must agree, but cases are ignored.

When using the S command, remember that it searches to the end of the file. If a typing error is made in the 'old-text' string and if it so happens that exactly the same string exists somewhere in the file, it will be changed by the S command. It is a good idea to type the line just changed (as in the third example above, by appending CLT) until experience with the command is gained.

POINTER POSITION

After a successful match, the pointer is moved to the next character after the last string inserted.

If a match is not found in the search, the pointer is not moved from its current location, and the following message is displayed:-

'Cannot find "text"'

where "text" is the string specifed as 'old-text' in the command.

FUNCTION

To display a line or lines of the format or function definition on the programmer.

FORMAT

The format of the T command is:

[n]T$$

where:
'n' specifies the number of lines to be displayed.
T is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

RULES

If 'n' is positive, the display starts at the pointer and proceeds forward in the file.

If 'n' is negative, the display starts 'n lines before the pointer and displays up to the pointer.

If 'n' is zero, the display starts from the beginning of the current line and proceeds to the pointer.

If 'n' is omitted, 1 is assumed, i.e. the display starts from the pointer up to and including the newline character.

If a minus sign only is used, -1 is assumed.

'n' must lie in the range -32768 to 32767 inclusive.

An unsigned number is assumed to be positive, i.e. a plus sign is not necessary.

EXAMPLES

To display a line from the pointer to the end of the line:

T$$

To display from the beginning of the line to the location of the pointer:

0T$$

To display the entire line no matter where the pointer is located within it:

0TT$$

To move the pointer to the beginning of the line and then display the line:

0LT$$

To display the previous line and move the pointer to the beginning of that line:

-LT$$

To display the entire file if the number of lines in it is not known but a maximum of 500 is certain:

B500T$$

When the end of the file is reached the command stops.

POINTER POSITION

The T command does not move the pointer.

FUNCTION

To inhibit/enable the display of the results of the format or function definition while it is still being edited.

FORMAT

The format of the V command is:

V$$

where:
V is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

NOTES

When the editor is entered, the view facility is enabled. When editing an existing file, the format or function that it defines is displayed on the video monitor.

Whenever ESC ESC is used to terminate any command or series of commands, then, after the command has been processed, the format or function will be displayed on the video monitor. A cross-wire cursor will show the final position on the video monitor.

Where a number of small changes are being made to a large file, the delay in compiling the format and then displaying it may become obtrusive. The view facility may be inhibited by use of the V command. A further V command will re-enable the view facility.

If using a system programmer and the W command is entered, the format will be viewed so long as the facility has not been turned off. When in window editing mode (see W command), pressing ESC causes the format to be viewed.

POINTER POSITION

The V command does not move the pointer.

FUNCTION

To enter an alternative easier-to-use edit mode when using a system programmer only.

FORMAT

The format of the W command is:

W$$

where:
W is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

NOTES

When a W command is issued, *any* programmer screen will display the file being edited from two lines before, and up to 21 lines after the line referenced by the text pointer, assuming that there is sufficient text in the file. However, the pointer can only be moved in window mode using a system programmer, and not with a portable programmer. The following notes therefore apply to system programmers only.

POINTER MOVEMENT

The text pointer is displayed as a cursor which can be moved by means of the cursor keys on the keyboard (↑ ← → and ↓). If an attempt is made to move the cursor off the top or bottom of the screen, the display will scroll up or down one line. If an attempt is made to move the cursor off either side of the screen, it will stop and is prevented from moving any further.

The home key (diagonal arrow) can be used to move through the file in overlapping blocks of 24 lines, where 2 lines of the previous display are carried over and 22 lines are fresh. The direction is taken from the last used up or down cursor movement (↑ or ↓). If the home key is pressed before pressing either of the up or down cursor keys, the direction is assumed to be down.

Pressing Control-B moves the cursor to the start of the file. Pressing Control-Z moves the cursor to the end of the file. In either case, the file will be re-displayed if appropriate.

INSERTIONS

When in window mode, characters typed on the keyboard will be inserted at the current cursor position provided this is:

within a line of text

at the immediate end of a line of text

at the first position of a blank line.

If the cursor is beyond the right-hand end of a line when a character is typed, the cursor will move to the end of that line, and the character will be inserted there, i.e. in the position preceding the newline character that terminates the line. The cursor and any text beyond it will be moved along one position.

The system programmer screen can display lines up to 80 characters long although lines longer than this can be created using the editor. The eightieth character, if it is not newline, will be displayed as '!' at the right-hand margin. Characters beyond the eightieth cannot be read by the user or altered by the editor in Window Mode.

DELETIONS

Pressing the RUB OUT key deletes the character ***immediately before*** the cursor and adjusts the following text to match. If the new cursor is at the left-hand end of a line, the preceding newline character will be deleted and the display adjusted appropriately.

Pressing Control-D will delete the character *at* the cursor, and adjusts the following text to match.

In either case, if the cursor is beyond the right-hand end of a line, the cursor will move the end of that line before the appropriate character is deleted. This will be either the newline character or its predecessor.

HELP

Pressing Control-? causes the editor to inspect the character at the cursor. If this is not an alphabetic character, the editor backtracks until one is found. It then backtracks to the beginning of the word containing that character. This word is then matched against the graphic Display Language keywords. If a match is found, the associated help message is output. If no match is found, the message:

"Cannot find keyword"

is output, and the keywords and in-built functions are listed.

RETURNING TO THE MAIN EDITOR

Using window editing, it is not normally necessary to use the B, C, D, I, L, T and Z editor commands, since the same effects can be obtained by cursor movement or Control key commands. Eventually it will be necessary to return to the main edit level to terminate the session using the E or Q commands.

A return to the main edit level from window mode can be achieved by pressing ESC ESC, as for the completion of all other edit commands. The first press of ESC causes the format or function definition to be compiled and appear on the video monitor screen; the second press returns to the main edit level.

FUNCTION

To move the pointer to the position immediately following the last character in the file.

FORMAT

The format of the Z command is:

Z$$

where:
Z is the command code.
$$ represents what is displayed on the programmer when the ESC (Escape) key is pressed twice to terminate the command.

EXAMPLE

To move the pointer to the end of the file:

Z$$

NOTES

The Z command is useful to add text at the end of a file.

When the pointer is at the end of the file, there is no text following it. After issuing a Z command, and then a command such as Type (T), nothing is typed because there is no text to type.

A command string or a single command can be repeated by enclosing the command string in angle brackets (< >), preceded by a number that specifies the number of times the command string is to be performed. This number should be in the range 1 to 32767 inclusive.

A typical use of this facility is for changing a character string that exists in several places throughout the file. For example, suppose there is a program that uses a function named 'X1', and all occurrences are to be changed to the name of something meaningful to anyone else reading the code. To change 'X1' to 'VALVE' throughout the file, and to type each change as it is done, use the following:-

```
B100<SX1$VALVE$0LT>$$
```

Note...In this example, a B command was used first to move the pointer to the beginning of the file and that an ESC character was used to separate VALVE from the 0LT command. This prevents the editor from using the 0LT as part of the second string parameter to the S command.

Command string iterations can be nested. For example, the command:

```
100<3C2<5CD>L>$$
```

will go through a 100 line file. On each line it will advance the pointer 3 characters and then, twice, it will advance five more characters and delete a character. The inner command series advances the pointer five characters and deletes the sixth character. The outer commands advance the pointer three characters before executing the inner command string, and then advances the pointer to the next line.

Iteration command strings can be nested eight deep. If more than eight strings are nested, the following error message is issued and the innermost commands are not executed:-

'Nesting too deep'

This page left intentionally blank

USING THE CHARACTER EDITOR

This page left intentionally blank

This Section provides instructions on using the character editor. Text contained within double quotes denotes messages or prompts displayed via the programmer. Where appropriate, the required response is shown by a numeric value immediately followed by a Carriage Return. All numbers are assumed to be decimal unless appended with the character 'H' denoting hexadecimal notation (base 16).

The IMAGEM video system can deal with up to 224 characters, all of which can be modified. A further 32 characters are used internally and cannot be displayed. All characters are referenced by 2-digit hexadecimal values. The full character set is as follows:-

- Characters 00H to 1FH: 32 control characters reserved for internal use.
- Characters 20H to 7FH: The default set, consisting of the standard ASCII 96-character set. These characters are stored within the IMAGEM video system.
- Characters 80H to FFH: 128 additional characters which can be defined. Initially, they are set to display as an ASCII space character.

All characters are based on a 7*14 matrix. The system contains a dedicated character editor that allows the in-built default character set to be extended and/or modified.

All character editor commands can be performed with just a programmer. However, additional information is presented on the video monitor if there is one connected. It will display the complete character set with any selected character(s) highlighted. During character editing, the selected character will also be displayed in enlarged sizes.

User-defined (or pre-defined but user altered) characters are stored within the first 16K of the external battery supported memory subject to room being available.

SELECTING THE CHARACTER EDITOR

To enter the Character Editor, select option '3' of the IMAGEM graphic video main menu.

On successfully entering the character editor, a working area is set up, consisting of the default set as modified by any user-defined characters. Changes made during the character edit are all made on this working set. Any modified characters are only stored if the character editor is exited through one of the proper routes.

USING THE CHARACTER EDITOR

The main menu of the Character Editor will now be displayed listing the following options:-

```
Character Editor

    1    Copy Character(s)
    2    Delete Character(s)
    3    Edit Character(s)
    4    Delete all User Defined Characters
    5    List externally stored User Defined Characters
    6    Abort Editing Session -all changes lost

    R To Return To Previous Menu.

Please Press Required Option [ ]
```

At the same time, the video monitor displays the character set in the form of a 16*14 grid.

Whenever returning from any one of the options numbered 1-5 inclusive, this menu and format will be re-displayed.

The ESC key can be used at any level to effect a premature end.

MULTIPLE VIDEOS

If there is a GEM80 Controller containing more than one IMAGEM video system, and programmers are connected to more than one video at the same time, the character set cannot be edited if it is being edited simultaneously by another video. If tried, the following error message is displayed:-

> 'The character set is already being edited.
> Press space bar to continue".

Press the space bar to return to the main menu of the IMAGEM graphic video system.

FUNCTION

To copy one or more characters from their present character value/position to a new character value/position.

EXAMPLE

If it is required to copy 16 characters, starting at character value 20H to character value 80H, the following entries should be made to the prompts:-

'Please enter the first character number to copy from in hexadecimal >" 20<CR>

'and the first character number to copy to in hexadecimal >" 80<CR>

'and the number of characters in decimal >" 16<CR>

The selected characters blocks of characters are highlighted in green and cyan respectively. A prompt is now displayed with the further question:

"Do you wish to perform the copy ? (Y or N) [ ]"

If 'y' or 'Y' is pressed, the copy is performed and the display updated to show the new character set, as well as re-displaying the Character Editor main menu. Pressing 'n' or 'N' will re-display the initial character set and again return to the Character Editor main menu. Pressing any other key will be ignored.

RULES

Copying overlapping blocks is allowed.

It is illegal to enter the same character value to the prompts 'copy from' and 'copy to'. If such an entry is made, the message:

"Source and destination addresses should not be the same.
Press space bar to continue."

will be displayed. Pressing the space bar returns to the Character Editor main menu.

If a request is made for a number of characters to be copied from a source or to a destination so that some of the characters would not fit (i.e. their values would be greater than FFH), only those characters that will fit are copied. There is no wrap round to the start of the character set.

FUNCTION

To delete one or more characters.

EXAMPLE

If it is required to delete 20 characters, starting at address 80H, the following entries should be made to the prompts:-

"Please enter the first character number to delete from in hexadecimal >" 80<CR>

"and the number of characters in decimal >" 20<CR>

The selected characters will be highlighted in green and prompt is displayed with the further question:

"Do you wish to perform the delete ? (Y or N) [ ]"

If 'y' or 'Y' is pressed, the deletion is performed, the monitor display is updated to show the revised character set, and the programmer re-displays the Character Editor main menu. If 'n' or 'N' is pressed, the monitor will re-display the initial character set, and again the programmer returns to the Character Editor main menu. Pressing any other key will be ignored.

RULES

If a requst is made for a number of characters to be deleted, starting from a destination so that some of the characters would be above the top of the range (i.e. some of the characters would have addresses greater than FFH), the editor deletes only those within the range. There is no wrap round to the start of the character set.

Deleting characters between 20H and 7FH inclusive causes one of the following alternatives:-

- If the character has previously been re-defined, the default character will be returned
- If the character is already the default one, deletion will not occur.

FUNCTION

To alter particular pixels within a selected character.

EXAMPLE

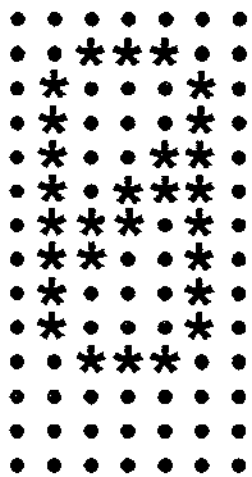
To edit character value 30H ('0') to remove the slash from the middle, respond to the prompt as follows:-

"Please enter the first character number to edit in hexadecimal >" 30<CR>

The video monitor will display the current character set with the selected character highlighted in green, together with a series of examples of the character at sizes 1, 2, 3, 4 and 5.

The programmer will display a 7*14 grid which represents the selected character. Text describing the use of the various keys will be displayed along with the grid. An example is shown below:-

Character value is 30 Hexadecimal



Use cursor keys or 8, 2, 4 or 6 to move Up, Down, Left or Right.
Use space bar to change state of pixel.
Use ESC to terminate.

Where a pixel within the character is OFF, a full stop (period) is displayed, while an ON pixel is represented by an asterisk. The cursor is initially positioned at the top left-hand corner of the grid but it can be moved within the grid using the cursor keys (up, left, right, down) on a system programmer. A portable programmer does not have any cursor keys, but the number cluster (8, 4, 6, 2) can be used instead.

The cursor movement is restricted to that part of the screen displaying the grid. If an attempt is made to move the cursor outside this grid, it will be displayed on the opposite side of the grid, i.e. the cursor will wrap around.

The pixel at the current cursor position can be turned from ON to OFF or vice versa by pressing the space bar. Any changes made will also be echoed on the video monitor.

To terminate the editing session, press the ESC key. A menu will be displayed, listing the following options:-

```
Character Editor

    1     Store Character and Terminate
    2     Discard Character and Terminate
    3     Store Character and Continue to next Character
    4     Discard Character and Continue to next Character

Please press required option [ ]
```

Options 1 and 2 will return to the Character Editor main menu, while options 3 and 4 will re-initiate an editing session for the subsequent character. In the case of character FFH, the next character is defined as 20H.

NOTE

Characters in the default set (values 20H to 7FH) correspond to the keys on the programmer. It is recommended not to alter any of the characters in this set to entirely different ones. For instance, it would be confusing if, when entering a text string, pressing the key marked 'B' produced the character 'D', say, in the video format. Therefore it is recommended that only the ***style*** of characters in the default set is altered, as in the example above, and not their functions.

FUNCTION

To restore the default character set.

EXAMPLE

On selection, the following prompt question is displayed:-

> "This command will delete all of your defined characters and restore the default set. Do you wish to do this? (Y or N) [ ]"

If 'y' or 'Y' is pressed, the default set will be restored. The Character Editor main menu will be re-displayed on the programmer screen, and the default character set will be re-displayed on the video monitor.

If 'n' or 'N' is pressed, the Character Editor main menu will be re-displayed without making any changes to the character set.

Pressing any other key except ESC will be ignored.

FUNCTION

To display the values of all those characters that have been re-defined. The delete option can be used to delete characters whose values are known. This option can be used to find those values.

NOTE

Any changes that may have occurred during this entry into the character editor will not be reflected until exiting from the character editor through one of the proper routes.

If there are no user-defined characters currently stored, the following message will be displayed:-

"There are no user-defined characters currently stored.
Press space bar to continue."

otherwise the following message is output:-

"The following characters are currently stored:"

followed by a list of all user-defined characters currently stored. On listing all the characters, the programmer will display:

"Press space bar to continue."

pressing space bar returns to the Character Editor main menu.

FUNCTION

To exit the editing session without storing any changes that may have been made.

RULES

If any changes have been made, to guard against inadvertent loss of text, the following prompt is displayed:

"Do you really wish to lose all your changes? (Y or N) [ ]"

If 'y' or 'Y' is pressed, the previous version of the character set will be restored, although not re-displayed. The character editor is then exited, and the graphic video main menu will be displayed on the programmer screen.

Pressing 'n' or 'N' also returns to the graphic video main menu but stores all changes to the character set. Pressing any other character will be ignored.

This page left intentionally blank

INTERNAL ADMINISTRATION

This page left intentionally blank

This section describes a number of facilities available to assist in administering the system. These are available by selecting options 1, 4 or 5 as appropriate, from the IMAGEM graphic video main menu shown below:-

```
IMAGEM - GEM80 Graphic Video - Main menu

    1    Self Test and Set Up
    2    Edit Format or Function
    3    Edit Character Set
    4    Copy Format or Function
    5    Delete Format or Function
    6    Print Character Set, Format or Function
    7    Load from Tape or Disk
    8    Dump to Tape or Disk
    9    Verify Tape or Disk

    R To Return to Previous Menu

Please Press Required Option [ ]
```

To clear the external store, change the field rate, or obtain a display giving a summary of what is stored in the system, select option 1 (Self-Test and Set-Up). A further menu is displayed as below:-

```
Set for 60 Fields per Second

    1    External Store Test
    2    Clear External Store
    3    Change Field Rate
    4    System Summary

    R To Return to Previous Menu

Please Press Required Option [ ]
```

Note...

1. The above menu shows the field rate that the Video system is currently set for (60 fields per second in the example above). If the field rate is changed (see below), the menu display will change correspondingly. In the latest version of IMAGEM, the field rate is not user selectable and is set at 75 fields per second.

2. Option 1 (External Store Test) is not dealt with here but in Section 11 on 'Test Facilities'.

CLEAR EXTERNAL STORE

To use this facility, select option 2 of the above menu. The facility allows all formats, functions and user defined characters to be cleared out, or to set up the external store when first used. To protect against accidental errors, confirmation will be requested that all the stored data is to be cleared out. If confirmation is given, the store will be cleared.

It can be seen that, before selecting option 3, the menu displays the current field rate. After changing the field rate, the menu will be re-displayed with the new field rate.

SYSTEM SUMMARY

To use this facility, select option 4 of the previous menu. The facility displays a summary consisting of:

- Functions: Their names and sizes.
- Formats: Their numbers and sizes.
- Characters: The quantity of user-defined characters.

If further details of individual items are required, these can be found by using the appropriate editor.

For convenience, the System Summary facility is also accessible from the Documentation and Archive areas.

COPY/DELETE FORMAT OR FUNCTION

For either of these two facilities, it is necessary to return to the IMAGEM video main menu shown at the top of the previous page.

Copy Format or Function

To use this facility, select option 4 from the video main menu. The facility allows a format or function to be copied. It can be used to:

- create a new format or function
- overwrite another format or function.

A format number or function name from which to copy will be requested. Type in the answer in the usual way, terminated with either a carriage return (CR) or line feed (LF)). If the system cannot find a format of the number, or a function of the name given, an error message will be produced.

Assuming that the system can find the format or function entered, a request is made to enter the format number or function name that it is to be copied to. The system then performs the copying operation.

If, however, the format number or function name given for copying to is one that already exists, confirmation will be requested that the previous version is to be overwritten. Once confirmed, the copy is performed.

Note...It is permissible to copy a function to a format, or vice-versa.

At any time prior to the copy being performed, a premature termination can be effected by pressing the escape key (ESC).

Delete Format or Function

To use this facility, select option 5 from the video main menu. The facility allows a single format or function to be deleted. If it is required to delete all formats and functions, use the 'Clear External Store' facility described earlier. A request will be displayed to enter a format number or function name. Once this is done the appropriate item will be deleted. At any time prior to the deletion being performed, a premature termination can be effected by pressing the escape key (ESC).

DOCUMENTATION FACILITIES

This page left intentionally blank

This section describes the facilities included to enable a print out to be obtained of all or part of the character set, and any format or function, for the purposes of documentation. To use these facilities, connect a printer to the programmer and suitably configure it. The facilities are accessed from the video main menu by selecting option 6 Print Character Set, Format or Function).

```
IMAGEM - GEM80 Graphic Video - Main menu

    1    Self Test and Set Up
    2    Edit Format or Function
    3    Edit Character Set
    4    Copy Format or Function
    5    Delete Format or Function
    6    Print Character Set, Format or Function
    7    Load from Tape or Disk
    8    Dump to Tape or Disk
    9    Verify Tape or Disk

    R To Return to Previous Menu

Please Press Required Option [ ]
```

When option 6 is selected, a further menu will be displayed as shown below:-

```
Print Character Set, Format or Function

    1    Print Character Set
    2    Print All Functions
    3    Print All Formats
    4    Print Selected Format or Function
    5    System Summary
    6    Print System Summary

    R To Return to Previous Menu

Please Press Required Option [ ]
```

PRINT CHARACTER SET

To use this facility, select option 1 of the above menu. This facility allows characters from the current character set to be printed, i.e. the in-built default character set as modified by any user-defined characters.

A prompt to enter the hexadecimal value of the first character to be printed will be made; this should be in the range 20 to FF inclusive. The next prompt will be for the number of characters to be printed. If a large number is entered (e.g. 999), this will be limited to print up to character value 'FF'. Thus entering '20' followed by '999' will print out the whole character set. Further questions may be generated by the programmer depending on its current set-up state.

The print-out shows the characters as an array of dots and asterisks, in the same way as the Character Editor displays them, refer to Section 7. Where several characters are asked to be printed, they are printed in rows of four characters.

PRINT ALL FUNCTIONS

To use this facility, select option 2. Provided that any user defined functions are present in the external store, their definitions will be printed out.

PRINT ALL FORMATS

To use this facility, select option 3. Provided that there are formats present in external store, their definitions will be printed out.

PRINT SELECTED FORMAT OR FUNCTION

To use this facility, select option 4. A prompt is given to enter a format number or function name. Provided that this format or function is present in external store, it will be printed out.

SYSTEM SUMMARY

To use this facility, select option 5. It displays a summary on the programmer screen consisting of:

- Functions: Their names and sizes.
- Formats: Their numbers and sizes.
- Characters: The quantity of user-defined characters.

This facility is useful as a reminder of what is on the system before requesting particular items to be printed out. If further details are required of individual items, these can be found by using the appropriate editor.

For convenience, the System Summary facility is also accessible from the Administration and Archive areas.

PRINT SYSTEM SUMMARY

To use this facility, select option 6. It prints out the same information as that displayed on the System Summary screen (option 5), except that the format is better suited to a print-out.

ARCHIVING FACILITIES

This page left intentionally blank

Formats, functions and user-defined characters are stored in battery supported memory on the GEM80 central highway. In order to protect against the consequences of hardware failure or user mal-operation (e.g. someone accidentally deleting a required format), it is recommended that the formats, functions and user-defined characters are archived onto magnetic tape or disk. This can be done using either

- an audio tape recorder connected to either a portable programmer or a system programmer, or
- the in-built digital tape or disk drive of a system programmer.

Archiving facilities are accessible from the IMAGEM graphic video main menu using options 7, 8 and 9:

```
IMAGEM - GEM80 Graphic Video - Main menu

    1    Self Test and Set Up
    2    Edit Format or Function
    3    Edit Character Set
    4    Copy Format or Function
    5    Delete Format or Function
    6    Print Character Set, Format or Function
    7    Load from Tape or Disk
    8    Dump to Tape or Disk
    9    Verify Tape or Disk

    R To Return to Previous Menu

Please Press Required Option [ ]
```

DUMP TO TAPE OR DISK

To access this facility, select option 8 on the video main menu. A further menu will be displayed giving a number of options for partial or total system dump, as shown below:-

```
Dump to Tape or Disk

    1    Dump Character Set
    2    Dump Format or Function
    3    Dump a Range of Formats or Functions
    4    Dump Whole System
    5    System Summary

    R To Return to Previous Menu

Please Press Required Option [ ]
```

DUMP CHARACTER SET

This facility dumps out all user-defined characters. If there are none, a warning is issued, and nothing is dumped.

DUMP FORMAT OR FUNCTION

This facility dumps out one or more functions or formats.

On selection, a prompt is displayed to enter a format number or function name. A prompt then asks if any further formats or functions are to be dumped. If positive confirmation is given, the next prompt asks for a further number or name to be entered.

When all the formats and functions to be dumped have been entered, if there are any user-defined characters, a prompt will ask if these are also to be dumped.

The system then proceeds to dump all the items requested.

DUMP A RANGE OF FORMATS OR FUNCTIONS

This facility dumps out a range of formats or functions. On selection, a prompt is given to enter a format number or function name for the first format or function to be dumped. The next prompt asks for the last format or function to be dumped.

Only either formats or functions can be dumped at any one time, not both. If a format is specified for the start of the range and a function for the end of the range, or vice versa, an appropriate error message is displayed, and the main menu is returned.

If the function/format given for the start of the range doesn't exist, an appropriate message is displayed, and the Dump menu is returned. The function/format given for the end of the range, however, need not exist. The range is treated in ascending order, alphabetically for functions or numerically for formats, starting from and including the format/function given for the start of the range, and including the format/function given for the end of the range if it exists.

Always ensure that the start of range is alphabetically or numerically lower than the end of range for correct operations.

DUMP WHOLE SYSTEM

This facility dumps out all the formats, functions, user-defined characters and the current field rate.

Due to the size limitations of audio tapes, it is recommended that this facility is used only with a system programmer's digital tape or disk. With audio tapes, it is recommended that a selected range of formats or functions per tape is dumped, using the "Dump a Range" facility given above.

TAPE/DISK IDENTIFICATION

Once it has been defined what is to be dumped, a prompt is displayed to enter the program name, the issue number and the time and date. These are suggested entries and are not checked. Any three suitable, single line (up to 77 characters) entries may be used as required. Once entered, a prompt is given to operate the tape or disk unit so that the dump can be performed.

SYSTEM SUMMARY

To assist in dumping partial systems, a System Summary can be obtained by selecting option 5 from the 'Dump to Tape or Disk' menu.

This provides a summary on the programmer screen consisting of:

- Functions: Their names and sizes.
- Formats: Their numbers and sizes.
- Characters: The quantity of user-defined characters.

This facility is useful as a reminder of what is on the system when particular items are requested to be dumped. If further details are required of individual items, they can be found by using the appropriate editor.

For convenience, the System Summary facility is also accessible from the Administration and Documentation areas.

VERIFY TAPE/DISK

Once dumped, it is recommended that the tape or disk is verified. Verification means that the system reads back the tape and compares it with the data stored in memory.

To verify a tape or disk, select option 9 from the IMAGEM video main menu. A prompt is displayed to operate the tape recorder so that verification can be performed. Any differences will result in messages being displayed on the programmer screen.

Note... Because the dump facility allows partial system dumps, verifying a tape or disk does not check that everything in the system is on the tape or disk. It only checks that everything on the tape or disk is in the system, and that the tape or disk is correct within itself.

LOAD TAPE/DISK

To use this facility, select option 7 from the IMAGEM video main menu. A prompt is displayed to operate the tape or disk unit so that the load can be performed.

Before the prompt is given to operate the tape or disk unit, a check is made to see if any other video in the subrack is editing a format or function. Such an event can only happen where the subrack contains more than one video, and where there are separate programmers plugged into separate video modules at the same time. If another video is editing a format or function, a message is output asking whether to abandon the load or to continue. It is at the user's discretion to continue. If it is decided to continue, a prompt is displayed to operate the tape or disk unit as normal.

This page left intentionally blank

TEST FACILITIES

This page left intentionally blank

This section describes:

- Self-test facilities
- Action taken on display time faults.

FRONT PANEL LIGHTS

On the front of IMAGEM, two green lights are provided as a fault location aid as follows:-

O.K. This light is on the left-hand side of the front panel. It will be lit provided that no faults are found within the system. Referred to as the O.K. light.

ACTV This light is on the right-hand side of the front panel. It toggles, i.e. it changes state from OFF to ON, or vice versa, on every update of the display. The rate at which it is switching on and off therefore indicates how long it is taking for the format definitions making up the display to be processed. Referred to as the Activity light.

FAULT DETECTION

The video system can detect faults from two causes:

- hardware mal-operation.
- programming errors.

To detect any hardware faults, the video system performs various self-test checks, first on start-up, and then repetitively during running.

When writing format or function definitions, the compiler will find most of the programming errors. It will not be possible to store a format or function definition in the external store until all these errors have been edited out. However, the video system can only detect certain types of programming errors when it processes the format definitions making up a display. For instance, if a field descriptor has been set up with a DECIMAL command catering for a certain number of digits, but a data value to be displayed consists of more digits, this programming error only becomes apparent at display time.

FAULT CODES

To give specific information in case of faults, error codes are written into locations F1 and F9 (of the video, not the controller) as follows:-

F1 Stores self-test fault codes. F1 = 0 indicates healthy state.

F9 Stores display time fault codes. F9 = 0 indicates the correct display of functions and formats.

A fault code is only written into either of these two locations if it is already clear. If two faults were found in succession, only the first would be registered. Once a fault is no longer evident, the relevant table is automatically cleared back to zero.

FAULT DISPLAYS

When either a self-test or a display-time fault is found, the IMAGEM video system attempts, whenever possible, to continue running.

Certain text messages, or question marks in place of data, may be displayed on the monitor screen. However, it cannot be guaranteed that these fault messages will always be visible. Depending on the particular fault, they could be overwritten by later formats or come outside the window area. In case of any unexpected effects on a display, a programmer should be connected to see whether any F1 or F9 fault codes are displayed.

In those cases where the video system cannot continue, the monitor screen will change to a single solid colour.

START-UP CYCLE

On power-up, the IMAGEM video system executes a series of tests to check for correct operation of the system. While these tests are being performed, the video monitor displays a blank screen. If any of these tests fail, the system halts immediately, and will not show any further activity.

The tests performed in the start-up cycle are:

- On-board EPROM check. A CRC (cyclic redundancy check) is carried out to verify that there has not been any change to the contents of the EPROMs on IMAGEM.
- On-board RAM check. A test pattern is written to and read back from each byte of on-board RAM on IMAGEM.
- External store addressing. The mechanism for addressing the external store is verified.
- External directory pointers are checked for consistency.
- Directory CRCs are checked.

On IMAGEM Version 4.0 onwards, various checks are made for the presence or not of RAM modules with EPROM capability. During these checks, an explanatory format is displayed. The EPROM checks are only done if the IMAGEM is running in a suitable controller.

SUCCESSFUL START UP

If the start-up cycle is completed satisfactorily, the video system starts normal operation. The IMAGEM default format (Format 0) will always be displayed for roughly 5 seconds. This gives the controller sufficient time to complete its own self-test cycle. After this delay, the appropriate L-tables are accessed. Any formats requested will then be displayed. If no formats are requested, i.e. if the L-table locations corresponding to the particular video system all contain zero, the default format continues to be displayed.

If no display appears but the Activity light is seen to be flickering, the video system is running. The lack of display will be due to an interface failure between the graphic video system and the monitor.

UNSUCCESSFUL START UP

If no display appears and both front panel lights on the video processor module are unlit, there has been a start-up failure.

Note.. If both lights go out after the system has been running, this does not indicate a start-up failure.

If the display is solid magenta and the Activity light is seen to be flickering, a fatal external store error has occurred, and the store must be cleared. In this condition, the main menu allows only Option 1 (Self-Test and Set-Up) to be selected. Then, select Option 2 (Clear External Store) on the next menu, and reply with Y (for Yes) to the clear store query message.

Any programmer connected to the IMAGEM front panel port displays:

Not connected to GEM80

or:

No response from GEM80

No fault code will be written to F1 on start-up failure.

REGULAR ON-LINE TESTS

Once the video system has started up, on-line tests are performed in a repetitive cycle during any free processor time.

The tests performed in the cycle are:

- On-board EPROM check; the same check as on start-up, but performed repetitively.
- On-board RAM check; again, the same check as on start-up but performed repetitively.
- CRC and chaining checks on every format definition held in the external store.
- CRC and chaining checks on every function definition held in the external store.

If an EPROM check fails, the video system halts, and the monitor screen shows solid RED all over. No fault code is written to F1 since the system will be shut down.

If one of the other checks fail, the video system continues running, but may produce some fault indication on the monitor screen or otherwise deviate from the expected display. A fault code will be written to F1, and the O.K. light will go out.

FORMAT AND FUNCTION ERROR BIT

If the CRC and chaining checks for any format or function definition fail, an error bit is set in the particular definition header. Any definition with the error bit set cannot subsequently be loaded from external store until it has been corrected by editing.

If the system summary is displayed on the programmer, any format or function definition that has its error bit set will show the message:

ERROR

displayed alongside it.

Once the particular definition has been edited, the error removed, and the definition copied back to external store, the error bit will be re-set. The error message no longer appears on the system summary display. The video system is then able to load the format or function again when called up for display.

SOFTWARE WATCHDOG

Because of the nature of the IMAGEM Display Language, it is not possible to program infinite loops (as there are no GOTOs). However, a format definition could be written that took a very long time to process if FOR..TO (or FOR..DOWNTO) commands were used repeating a large number of times or deeply nested.

To protect against any programming errors of this nature, a 'software watchdog' checks the execution time of FOR..TO commands, and prematurely terminates execution of a format definition if the time exceeds 35 seconds. If this occurs, the O.K. light goes out, and a fault code is written to F1.

USER INITIATED EXTERNAL STORE TEST

A test can be performed on the external store by plugging a programmer into the IMAGEM front port, and selecting option 1 (Self-Test and Set-Up) from the video main menu:

```
IMAGEM - GEM80 Graphic Video - Main menu

      1       Self Test and Set Up
      2       Edit Format or Function
      3       Edit Character Set
      4       Copy Format or Function
      5       Delete Format or Function
      6       Print Character Set, Format or Function
      7       Load from Tap or Diske
      8       Dump to Tape or Disk
      9       Verify Tape or Disk

      R To Return to Previous Menu

Please Press Required Option [ ]
```

This displays the menu shown below. From this, select option 1 (External Store Test):-

```
Set for 60 Fields per Second

      1       External Store Test
      2       Clear External Store
      3       Change Field Rate
      4       System Summary

      R To Return to Previous Menu

Please Press Required Option [ ]
```

Next, a prompt is given to enter the required page number for the test (in the range 0 to 33 inclusive).

If a page number is entered for which no memory module is fitted in the particular controller, or if there is a connection fault, the following message will be displayed:

Page Not Accessible

Assuming that the page can be accessed, if the test proved successful, the following message will be displayed:-

Test Passed

If, however, the test fails, the following message will be displayed:-

Test Failed at #####

where ##### represents the number of the memory location at which the test failed.

The test can be performed at any time, provided that no other utility is currently reading from or writing to the external store. If this happens, the test waits until it is able to gain access. During the short time taken to complete the test, the display on the video monitor freezes and is not updated. It is recommended that an external store test should be performed during system commissioning, and subsequently if a malfunction is suspected.

The following 8 pages give, in tabular form, what fault conditions may be observed on the video monitor screen, the fault codes and messages which may occur on a programmer when plugged into IMAGEM, the possible reasons for the fault condition, and the action necessary to recover from the fault.

As noted earlier, the effects of a fault visible on the video monitor screen cannot be guaranteed, since it is possible for fault messages to be overwritten by later commands or formats.

Video Monitor Display and Video Processor Front Panel Lights	Fault Code	Programmer Display
Incomplete display. O.K. light unlit.	F1 = 4	Self-Test failure - excessive format generation time.
Solid GREEN screen about 10 secs after the above fault. O.K. light unlit. Video system shuts down.		Only option 1 (Self-Test and Set-Up) accessible from video main menu.
O.K. light unlit	F1 = 10	Self-Test failure - on-board RAM failure.
O.K. light unlit	F1 = 11	External RAM module failure.
Solid MAGENTA screen O.K. light unlit	F1 = 14	External store self-test failure - new system or battery failure. Please clear store and re-load.
	F1 = 15	External store self-test failure - physical system changed. Please clear store and re-load.
	F1 = 16	External store self-test failure -directory pointers inconsistent
	F1 = 17	External store self-test failure - function chain CRC failure. Please clear store and re-load.
	F1 = 18	External store self-test failure - format chain CRC failure. Please clear store and re-load.
Solid YELLOW screen O.K. light unlit.	F1 = 40	External store access time-out
O.K. light unlit (Occurs only on versions earlier than V3.5)	F1 = 50	Self-Test re-commence failure - external store tests suspended
Solid MAGENTA screen	F1 = 60	Incomplete controller initialisation, controller uncompiled
Solid GREEN screen	F1 = 61	IMAGEM AND CONTROLLER MODULES ARE INCOMPATIBLE

Possible Reasons	Recovery
FOR loops with a large numbers of repeats, or deeply nested.	Edit format or function definition to reduce the number of repeat actions.
Failure of software watchdog to terminate format with excessive format generation time.	Power Controller OFF ON. If fault recurs, replace IMAGEM module.
Suspect IMAGEM module.	Power Controller OFF ON. If fault recurs, replace IMAGEM module.
Suspect RAM module.	Carry out external store test. If fault confirmed, power down the controller, replace relevant RAM module, and re-load formats from tape or disk.
New system or battery failure.	Clear store and re-load from tape or disk.
RAM has been added to or removed from the subrack, or the subrack is not correctly coded for it.	
Problem detected with store directory; can be caused by swapping RAM modules or a faulty RAM module.	
Faulty RAM module, or powering down before the editor has completed an exit.	
Faulty RAM module, or powering down before the editor has completed an exit.	
Video was prevented from accessing store from more than 20s. Usually caused by another video re-shuffling store during a load from tape or disk.	Recovery is automatic; screen returns to the same picture as before the yellow screen. The yellow screen simply indicates that display updating has been suspended.
Too much editing activity in progress for self-test to resume, especially if more thanresume. one video is being edited simultaneously.	Reduce editing activity.
IMAGEM had to wait too long for the controller to re-compile. This fault can occur if the controller remains uncompiled after power-up.	Re-compile the controller.
This fault occurs if a 164 System IMAGEM is used in a 300 system rack.	use matched systems.

Video Monitor Display and Processor Front Panel Lights	Fault Code	Fault Programmer Display
Normal screen Lost colour displayed in flashing green.	F1 = 65	COLOUR DEFINITION LOST. Colour statements probably 'IF'ed out.
Solid RED screen O.K. light unlit. Video system halts.		No fault code or message; system halts.
Solid BLUE screen. O.K. light unlit. Video system halts.		No fault code or message; system halts.
Solid GREEN screen O.K. light unlit. Video system halts.	F1 = 70	CHICS error number: - #@####.####
	F1 = 71	HELP! Contact CEGELEC Industrial Controls. Internal error No. #@###:####
	F1 = 72	RAM-EPROM module expected as module No. #@####:####
	F1 = 80	Bus time out
No characters displayed where expected. Format processing continues.	F9 = 100	Negative character field width not permissible
'TOO MANY COLOURS' in current foreground and background colours at screen top left-hand corner. Format processing continues, ignoring the last COLOUR command.	F9 = 101	Too many colours requested - only 16 permissible
'TOO MANY FUNCTIONS' in current foreground and background colours at screen top left-hand corner. Format processing continues, ignoring the function call, and returning a value of zero if the function was mathematical.	F9 = 102	Too many functions requested - only 32 permissible
'INVALID FORMAT' in flashing RED on BLACK in size 3 characters, at top left-hand corner of blank screen. Processing of format cannot begin, and system skips to next format.	F9 = 103	Invalid format requested

Possible Reasons	Recovery
Caused by removing the request to display the format which currently defines the colour. Most commonly caused by not executing a colour definition statement as a result of an IF condition. Occurs after changing display	Request L-tables
Suspect IMAGEM module.	Power Controller OFF/ON. If fault recurs, replace IMAGEM module.
Invalid 80186 instruction encountered.	Power Controller OFF/ON. If fault recurs, replace IMAGEM module.
An interprocessor communications error has occurred.	Power Controller OFF/ON. If fault recurs, replace video and other processor modules.
An internal error has occurred, possibly a firmware problem.	Power Controller OFF/ON. If fault recurs, contact the factory.
Different type of memory module expected.	Power controller OFF, replace memory module with one of the correct type, and power up again.
Faulty memory module, or subrack not coded for it.	Power controller OFF, replace memory module, and power up again.
A CHAR or a GETCHAR command has a negative value for its field width parameter.	Edit format or function definition to prevent the field width parameter going negative.
More than 16 colours are in use at the same time in the format and function definitions that make up the current display.	Edit the format definitions to reduce the number of colours in use. If any format definition calls up functions, check that these functions do not set further colours.
More than 32 different user-defined functions are called up, i.e. functions with more than 32 different names.	Edit your format definitions to reduce the number of different functions called up. Check also if any of these functions call up further different functions.
1. The controller has requested, via L-table, a format number for a format that does not exist.	Check that L-tables call up only valid format numbers or zero (0 will cause the default format to display).
2. The format definition held in external store contains an error preventing it from being decompiled in internal store.	Check that, on the video system summary, none of the formats being used has errors flagged up. If any format has, edit out the error.

Video Monitor Display and Processor Front Panel Lights	Fault Code	Fault Programmer Display
INVALID FUNCTION in flashing RED on BLACK anywhere on the screen. Format processing continues, ignoring the function call, and returning a value of zero if the function was mathematical.	F9 = 104	Invalid function requested
'INSUFFICIENT STORE' in flashing RED on BLACK in size 2 characters at top left-hand corner of blank screen. Display processing cannot begin for the format just loaded, and skips to next display	F9 = 105	Insufficient space at display time
Display responds as though data is zero where it was expected it to have a non-zero value. Effect on display depends on format or function definition, e.g. whether the data is itself displayed (e.g. as digits or a bar chart) or whether the data controls the presentation of the display.	F9 = 106	Invalid GEM80 Table letter.
	F9 = 107	Invalid GEM80 Table offset.
? (Single Question mark) in current foreground and background colours anywhere on the screen.	F9 = 108	String cannot be accessed
Several question marks in current foreground and background colours anywhere on the screen where digits were expected to appear. Otherwise, the display is as expected, and processing continues.	F9 = 109	Field width too small - will lose digits
	F9 = 110	Hex field width too small - will lose digits

Possible Reasons	Recovery
1. A format or function is trying to call up a function whose name does not exist.	Check the names of functions called up in the format or function definitions making up the display.
2. The function definition held in the external store contains an error preventing it from being de-compiled in the internal store.	Check that, on the video system summary, none of the functions being called up has errors flagged up. If any function has, then edit out the errors.
The format definitions called up, including any functions used within them, are too big to fit into internal store.	Reduce the size and or quantity of formats making up the display.
No table of the given letter exists in the controller's memory.	Check that all tables referenced in format and function definitions exist in the particular controller. For controllers such as a GEM80/310, this depends on what table references are used in the controller's program.
The controller s memory contains a table of the specified letter, but the number is too large, referring to a location beyond the end of the table.	Check that all table numbers referred to in format or function definitions exist in the particular controller. For controllers such as a GEM80/310, the highest number in any table may change when the controller's program is re-compiled.
In a Controller such as a GEM80/310, a message number has been referred to that exceeds that of the last message in P-table.	Edit format or function definition to prevent exceeding it calling up message numbers that of the last message,
The field descriptor portion of a DECIMAL or UNSIGNED command does not cater for the number of significant digits in the value that the program is trying to display	Edit format or function definition, either to extend the field descriptor or to limit the size of numbers being displayed.
The field descriptor portion of a HEX command does not cater for the number of significant digits in the value that the program is trying to display.	Edit format or function definition, either to extend the field descriptor or to limit the size of numbers being displayed.

Video Monitor Display and Processor Front Panel Lights	Fault Code	Fault Programmer Display
??? (three question marks) in current foreground and background colours anywhere on the screen.	F9 = 111	FOR statements have been nested too deep
	F9 = 112	Function stack overflow at display time
Solid GREEN screen. O.K. light unlit. Video system shuts down.	F9 = 113	Fatal display time error - insufficient internal store
'UNRECOGNIZED DISPLAY CODE' in flashing RED on BLACK anywhere on the screen. Format processing abandoned and system moves on to next format.	F9 = 114	Unrecognized display code encountered
'UNDEFINED COLOUR USED' in flashing RED on BLACK at top left-hand corner of the screen. Undefined colour represented in flashing GREEN. Format processing abandoned and system moves on to next format.	F9 = 115	Undefined colour encountered
	F9 = 116	Premature termination of string

Possible Reasons	Recovery
FOR loops have been nested more than 24 deep (an unlikely condition in a format or function definition).	Edit format or function to reduce the number of nested FOR loops to less than 24.
Too many function calls from within other function calls, resulting in insufficient memory space to cater for all the variables needed.	Edit format or function definitions to reduce the number of different functions currently active. A safe limit can be taken as 8.
	Power controller OFF ON.
1. Format definition in external store OK but corruption occurred when copying it into the internal store. 2. Format definition in the external store corrupted.	Load some other format, and then re-load the original format, by writing format numbers into the controller's L-table. If the fault persists, use the format editor to correct the format definition in external store.
Colour encountered in FOREGROUND or BACKGROUND command has not been defined.	Check the colour name spelling. Check colour name has been defined in a COLOUR command.
String of excessive length. May be caused by string data set up in one of the controller's data tables not being in standard message format or starting from the wrong word in a table.	Edit format or function to prevent excessive string length, or modify data in the controller's table locations.

This page left intentionally blank

PROGRAMMING HINTS

This page left intentionally blank

PLANNING

Once the user is familiar with an IMAGEM video system, format definitions for displays that are to be used by an operator will need to be produced.

It is recommended that the formats should be planned and for this purpose a format blank is included in the Appendices. In the Item later in this Section on 'Design of Formats', some hints are included on the factors to be considered when designing formats to meet the operator's requirements.

It is much quicker to work out the format definitions if this planning is done first. It may be necessary to deviate from the planned format if the visual effects obtained are not quite what was foreseen, but time spent in planning is time saved in coding.

COMMENTS

Comments should be included in the format and function definitions, refer to the Item on Comments in Section 5, as reminders of what the sections of a definition are designed to do. These will help if it is necessary to modify a display after it has been programmed or if other people need to modify it in the absence of the originator.

DOCUMENTATION AND ARCHIVING

It is good practice to run off fresh print-outs of format and function definitions whenever they are modified and also to archive them, as given in Sections 9 and 10.

FIXED AND VARIABLE PORTIONS OF A DISPLAY

Before a graphic video system can update the display, it must have processed the format definitions which make up the display. How quickly it can do this obviously depends on how long and complex these format definitions are. A guide to how quickly the display is being updated can be gained from the 'Activity' light on the front of the IMAGEM module.

In most formats, however, only certain parts consist of variable information taken from the controller's data tables. The rest of the format is generally fixed, therefore there is no need for this part of the format definition to be re-processed on each update.

Note...The Video system has two display buffers, one for the current display and one for the next display, these buffers being interchanged on a display update, refer to Section 3, System Interface. Therefore it is necessary to make sure that the fixed part of a display definition is processed twice, once for each buffer. After this, the fixed part of the format definition can be skipped and only the variable part processed. An example of how this may be done is given below:-

```
IF A<2 THEN
    (.....
    .....          .* Fixed portion of definition *.
    .....
    A = A + 1      .* Increment count *.
    )
.....
.....              .* Variable portion of definition *.
.....
```

In this example, the simple variable A is set to zero when the format definition is first called up. On the first time through the definition, the whole of it is processed, and A incremented to 1. Similarly, the second time through, the whole of the definition is processed, and A incremented to 2. On any subsequent execution, however, the fixed portion of the definition (which must be enclosed in parentheses as shown) is skipped.

When applying this technique, bear in mind that, in those cases where only part of the format is being re-displayed, it may be necessary to over-write a particular part of the display (e.g. with space characters) to clear off a message when it has been finished with.

QUANTITY OF PIXELS THAT HAVE TO BE REDRAWN

The length of time it takes to update a display depends on how many pixels have to have their colour re-calculated. It is possible to obtain improvements in speed by using the following techniques:

- Use graphic representation rather than text. Pictures can often present information that would otherwise need many words to describe, and therefore many pixels in the text characters.

- Don't display unnecessary spaces in text. For instance, in a tabular display of figures, don't display spaces between the columns of figures, but skip to the next colum.* using MOVE, in just the same way as a tabulator is used on a typewriter, or with a TAB command in a BASIC language program.

- Where it is required to display text that includes numbers taken from the controller's data tables, make maximum use of the controller's message facility (not available in /200 series). These messages are already formatted, and therefore save time on the video system compared with using, for instance, Text with a DECIMAL command.

- Use thinner lines and bars. Use narrower bars for bar charts than would be necesary with character graphics.

- With a dynamically changing shape, don't re-draw the whole of the shape every time. For instance, for a bar in a bar chart, draw only the extra portion when the bar is to be longer, and overwrite the end portion with background colour when the bar is to be shorter.

- Consider methods of presenting information that are possible with an IMAGEM video but which were not possible with earlier character video systems. For instance, instead of using a histogram, draw a true graph made up of straight line segments using the DRAW command. This may need less re-drawing of pixels than a histogram representation.

- Make minimum use of the FILL command. Use TRIANGLE and RECTANGLE commands wherever possible, and build shapes from combinations of these shapes.

DATA FROM THE CONTROLLER'S DATA TABLES

Where the display makes use of data taken from the controller's data tables, it takes only a short time for the video system to read this data from the controller's memory.

Each time a table reference (e.g. W[100]) appears in one of the format definitions, the contents of the table have to be read by the video. If the same table reference appears more than once in a format or function definition, the video system has to read the data separately on each occasion. Therefore a marginal time saving can be made if it is arranged to read the contents of the given table only once, and assign it to an internal variable, e.g.

```
D = W[100]
```

and thereafter use D, not W[100], in the format or function definition.

The time saving achieved this way is usually very small and probably not noticeable. The main advantage of this technique is not speed, but consistency. Refer to the Item on Variables in Section 5 under the heading 'Array Variables'.

INTRODUCTION

This section suggests a psychological approach rather than programming, by giving hints based on the experience of users rather than programmers.

BE KIND TO THE OPERATOR

While designing a video format, remember that the operator may have to look at it for the whole of a work shift whereas, in all probability, the designer will be looking at it for only a few minutes. What may be pleasing to look at for a few minutes might become irritating if observed over a long period.

FLASHING

Use flashing sparingly, for emphasizing abnormal conditions, but not to the extent that whenever an operator looks at the screen, something is flashing or moving in a regular manner. The operator could be 'hypnotised' rather than being able to produce the rapid response required.

Flashing alphanumeric data is more difficult to read. A 'soft' flash as used for the flashing colours in the default colour set, where the characters become dimmer but do not disappear while flashing, is better. For optimum legibility it is generally best to use a flashing square or rectangle alongside a message, rather than to flash the message text itself.

BRIGHTNESS OF SCREEN

Don't fill the screen with so much white or bright colour that it glares at the operator who will probably turn down the brightness on the video monitor, so that some of the intended colour contrasts are reduced. Instead, use a fair amount of black; as black is the unactivated screen colour, it produces no visible light. For example, use patterned fills with a black background colour. GREY should be used as a background colour in preference to WHITE.

VARIABLE INFORMATION

Try to avoid filling the screen with too much variable or changing information that the operator is expected to read or take notice of. Some experienced designers reckon that such information should not exceed 25% of the screen area. It is reasonable to exceed this figure if it reduces the number of times the operator needs to change from one screen format to another; once familiar with seemingly cluttered screen formats, operators can become adept at picking out any important changes. However, the 25% rule is a good starting point for the designer. Further, screens with a lot of variable information will be slower in updating.

What information is displayed on which format is of considerable importance. The designer should know sufficient about how the operator will control the plant so that he is provided, on a single screen display, with all the information required for a particular phase or mode of plant operation. At the same time, superfluous information not required for this phase or mode of plant operation, should be witheld.

ALPHANUMERIC INFORMATION

Where possible, avoid placing text and figures at the edges or corners of screens. This is because electron beam convergence in cathode ray tubes tends to fall off towards the edges of the screen, and also because of the curvature of the face of the screen. If the monitor is located near to strong electromagnetic fields (as in power plant), beam convergence can be further affected.

Try to place important alphanumeric information in the same area of the screen consistently, to reduce search time by the operator, as long as this does not otherwise make the presentation awkward.

COLOUR CHOICES

Different industries and different people have their own colour preferences for which colours to use for particular functions. However, there are certain colours and colour combinations that do not show up well, and should generally be avoided. Even so, these depend largely on:

- the particular colour monitor you are using.
- the ambient lighting conditions where it is placed.

so don't regard these recommendations as being correct for all cases.

Blue (especially light blue):	Avoid for small shapes, thin lines, alphanumeric and dynamically changing information.
Red foreground on black background.	Avoid; use red on white for better clarity.
Yellow foreground on white background.	Avoid; does not show clearly.
Yellow foreground on green background.	Avoid with some types of colour Monitor.

Good colour contrasts are:

- Black foreground on yellow background.
- Red foreground on white background.
- Blue foreground on white background.
- Green foreground on white background.

COLOUR BLINDNESS

A fraction of the population are colour blind. The number of people who cannot detect colour at all is very small, but a sizable number have difficulty in discriminating between certain colours. This lack of discrimination is what people normally mean by 'colour blindness'. More men (about 10%) have colour blindness than women.

For this reason, try to design screen formats so that colour enhances the display, but avoid designing them so that an operator has to know precisely which colours are which. If this is not done, it may be necessary to screen the operators for colour vision, or else subsequently re-define the colours used in the formats.

This page left intentionally blank

Main items on any topic have the page numbers underlined.

f means 'and the following page'; ff means and the following pages.

This page left intentionally blank

The information provided in this manual is accurate and as easy to understand as possible.

If, however, any shortcomings in this manual are evident, please note them on the page overleaf, and mail to:

Dick Wilson
Head of Control Systems Engineering
CEGELEC INDUSTRIAL CONTROLS Ltd
Kidsgrove
Stoke-on-Trent
ST7 1TW
England

or pass it to your local GEM80 agent.

IMAGEM GRAPHIC VIDEO PROGRAMMING MANUAL

Publication T390 Issue 5

Feedback from:

Name .....
Position .....
Company .....
Town or City .....
County or State .....
Postcode ZIP .....
Country .....

Items that ought to have been included:-

Items inadequately or confusingly explained (please quote page Nos.): -

Items you disagreed with (please quote page Nos.):-

Items you couldn't locate via the Index:-

Any other comments:-

*** Many thanks for your comments above ***

ALSTOM Power Conversion

France
9, rue Ampère
91345 Massy Cedex
Sales Tel: +33 (0) 8 25 02 11 02
Support Tel (International):
+33 (0) 3 84 55 33 33
Support Tel. (National):
08 25 02 11 02

Germany
Culemeyerstraße 1
D-12277 Berlin
Sales Tel: +49 (0) 30 74 96 27 27
Support Tel (International):
+49 (0) 69 66 99 831
Support Tel (National):
01 80 3 23 45 72

UK
Boughton Road, Rugby
Warwickshire, CV21 1BU
Sales Tel: +44 (0) 1788 563625
Support Tel: +44 (0) 1788 547490

USA
610 Epsilon Drive
Pittsburgh, PA 15238
Sales Tel: +1 412 967 0765
Support Tel: +1 800 800 5290

ALSTOM