XVM/RSX PART X
GRAPHICS

CHAPTER 1

INTRODUCTION

XVM/RSX GRAPHICS is comprised of a set of FORTRAN callable routines, a multiscope VT-15 handler (2 VT-15 processors, each with 2 scopes), and a handler for as many as four VW01 writing tablets. The FORTRAN routines provide both compatibility with the XVM/DOS graphics package and significant additional capability. Therefore, a DOS FORTRAN graphics program, with only a few minor modifications can be compiled, task-built, and run under XVM/RSX.

It is assumed that the reader is thoroughly familiar with the XVM/RSX system. While a familiarity with XVM/DOS graphics would be helpful because the systems are similar, it is not essential. And, although the RSX graphics package provides access to the graphics hardware, it is not necessary to have a thorough knowledge of the hardware. For instance, if the user is concerned about picture wraparound, he should consult the hardware manual.

1.1 GENERAL DESCRIPTION

The user prepares a FORTRAN source program. This program includes subroutine calls to the graphics system, as described in later chapters of this manual. The user FORTRAN program is then task-built (if this term is unfamiliar, see the On Line Task Development, Section) with the binary of the FORTRAN-callable routines and subsequently installed.

A MACRO user need merely simulate the FORTRAN subroutine calling sequences to access the graphics routines. Chapter 7 describes MACRO usage of the graphics system. If the MACRO user wishes to generate his own graphics code, and access the handler, that too is possible.

There is a considerable difference in the emphasis in the design of RSX graphics, as compared to DOS graphics. Basically, it is the intent of RSX graphics to remove the distinction between main and subpicture files, as much as possible, without destroying compatibility with DOS. Thus, the calls described in Chapter 3 do not have to be used at all in constructing new RSX graphics programs.

Although considerable effort has been made so that the conversion from DOS to RSX shall be of minimum difficulty, significant differences between the systems and the graphics packages remain. A knowledge of these differences should help the user to stay out of trouble.

It is expected that the user will run under the protection/relocation hardware in user mode rather than exec mode. This means that the user program runs in a maximum relative address space of 128K, with the relative zero address at any 400-word (octal) boundary. The VT15 processor is not affected by the protection/relocation hardware. It continues to run in an absolute address space.

There are two practical results of running under protection/relocation. First, a user-mode program cannot destroy other programs in the system; the protection hardware prevents this. At the same time, the VT15 processor can destroy other programs in the system, because it is not affected by the protection hardware. This means that the programmer should be careful when debugging graphics programs that are background to important tasks. It also means that it is dangerous for the user to modify arrays containing display code.

Second, a display file may not span an absolute 8K boundary. This creates a potential problem in laying out system partitions and FORTRAN arrays. The user must be careful and plan ahead when doing core layouts. The simplest method is to have the partition that contains the program start on an 8K boundary. The graphics system detects an error of this type before it occurs and stops the program before the VT15 processor is lost. When this type of error occurs, the user must rearrange his core layout, recompile and re-task build.

The RSX graphics system provides limited error detection for the user (see Appendix A).

1.2 REFERENCE MANUALS

The following manuals contain information useful in understanding the contents of this manual:

VT15 XVM Graphics Software Manual
GRAPHICS-15 Reference Manual
MACRO XVM Assembly Language Manual
FORTRAN IV XVM Language Manual
VW01 Writing Table Maintenance Manual

1.3 MINIMUM HARDWARE

Besides the minimum RSX hardware, RSX GRAPHICS requires a VT15 processor with arbitrary vector, and a VT04 scope.

In the maximum configuration, the system can handle two VT15 processors, four VT04 scopes and four VW01 writing tablets. In this maximum configuration, each of the four scopes can show completely different pictures. Alternately, the pair of scopes on each VT15 processor can display pictures in which part or all of the image is generated from one display file. LK35 keyboards are treated as input-only terminals by the RSX Monitor.

CHAPTER 2

SUBPICTURE ROUTINES

The FORTRAN user defines integer arrays into which are built his display files (subpictures). The display files are merely subroutines to be executed by the VT-15 display processor. A call to the subpicture routines places graphics code at the end of the existing graphics code in the specified display file. A brief description of each of the subpicture routines follows:

LINE - generates the display code to draw a line (intensified) or to reposition the beam (unintensified). The arbitrary vector hardware option is required for RSX GRAPHICS.

TEXT - generates code to display strings of 5/7 ASCII previously defined by the user in dimensioned arrays.

COPY - generates code to call another display file as a subroutine. May optionally include the use of the hardware SAVE-RESTORE feature.

PRAMTR - the PRAMTR (parameter) routine generates the display code to control such hardware options as scale, intensity, blink, light pen sensitivity, etc., for this display file, or some portion thereof.

GRAPH - generates the code to display data arrays in graphical form.

BLANK - generates the code to 'blank out' all appearances of a given display file.

UNBLNK - reverses the action of the BLANK subroutine.

POINT - generates code to absolutely position the beam. The point corresponding to the final position may optionally be intensified.

ANY - places a user-provided array of display code into the display file.

Example programs (Figures 2-1 through 2-8) using some of these commands are found in paragraph 2.14 at the end of this chapter.

2.1 GENERAL RESTRICTIONS

The following general restrictions apply to the use of the GRAPHICS system.

1. All arguments in the graphics system calls must be in integer format unless specifically noted otherwise. This requirement must be adhered to even if the argument name shown in examples and prototype calls is not in the normal FORTRAN integer variable character convention (I through N).

2. All display file storage must be declared by the FORTRAN user in the form of dimensioned integer arrays (may optionally be in COMMON).

3. The first location of each display file array is used by the graphics system as an end-of-file pointer. This location must be zeroed by the user prior to the first reference to that display file.

4. All references to display files in argument lists to the graphics routines must be of the form IFILE(1) rather than IFILE, unless otherwise specified.

5. Each provided array must be long enough to contain the display code to be built into it. No error messages are returned to the user upon array overflow. See paragraph 2.2 for further discussion.

6. No display file can span an absolute 8K core boundary. The graphics system will return an error code (see Appendix A) if this happens. The user can then either reposition his storage, or re-task-build.

7. Note, as of RSX PLUS III, and continuing through XVM/RSX, the FORTRAN subroutine calling conventions have been changed. A reference to IFILE is now entirely equivalent to a reference to IFILE(1). This change was implemented with FORTRAN version 044, and graphics primitives version VPR.33. Supercedes restriction 4.

   A FORTRAN program being brought through this version change must be recompiled with the new compiler, and re-task-built with the new graphics primitives. A MACRO program calling the primitives package with calls of the form IFILE must be modified. One level of indirection must be removed from these calls. Finally, reassemble and re-task-build.

8. Note, as of XVM/RSX, the format of the CNAME editing pointer has been changed. Since 17 bit addressing is supported, there is no longer room in CNAME for a 3-bit count field. The external symptoms of this change are fairly minor. Display files will occasionally become a few locations longer. Finally, REPLOTs which place PRAMTR instructions over other instructions may require an extra location. See Chapters 2 and 3 for further discussion.

2.2 GENERAL FORMAT OF DISPLAY FILES

2.2.1 Storage Overhead

A display file containing four vector commands, for example, has the following format:

```
LOCATION        CONTENTS

PNAME           6
+1              return address from DJMS*
+2              vector command
+3              vector command
+4              vector command
+5              vector command
+6              DJMP* PNAME+1
```

Three words are used for storage overhead in each display file. The first location (PNAME) contains the current length of the display file. This value must be initially set to zero by the user. After each reference to the display file by a subpicture routine, this value is automatically updated to reflect the current length of the file. The second location stores the return address and the last location contains the exit instruction.

The corresponding sequence of FORTRAN calls to generate this display file might typically be:

```
          DIMENSION IFILE(10)
          IFILE(1)=0
C         SET INITIAL VALUE OF FILE LENGTH TO ZERO

               .

          CALL LINE (100,0,1,IFILE(1))
          CALL LINE (0,100,1)
          CALL LINE (-100,0)
          CALL LINE (0,-100,1)
```

This display file would simply draw a square. Note that it is possible to provide variable numbers of arguments to the same graphics call. For example, with the LINE call, if only two arguments are provided, an intensified line is generated in the same display file that was last used. Subsequent paragraphs will provide a detailed description of the argument lists of the graphics calls.

2.2.2 Space Allocation

In the FORTRAN example above the dimension statement allocated ten locations for the display file. In this particular case only seven of the ten locations are used. Each of the four line commands requires a single location, and there are three locations used for the display file overhead. For a larger display file, the user does not wish to carry out a detailed count. What is recommended is to make a debugging version of the program with a much-too-large display file. When the program is done, the first location of the display file contains one less than the number of locations actually used. It is

then possible to move to a final version of the program with precise knowledge of your core requirements.

It has been stated that the subpicture routines add display commands to the end of display files. This can be done while the particular subpicture is displayed on the screen. There are two other methods of removing or modifying previously created graphics code. Whole display files can be reused by calling BLANK and then zeroing the first location. The graphics system then starts refilling the display file from the top. This can be done with the display processor running through the display file.

An additional method of modifying graphics code is thoroughly discribed in Chapter 4. This method writes a single group of commands (corresponding to a single call to the graphics system) over existing graphics code. This over-write requires an editing pointer to the beginning of the old group of commands. Under RSX GRAPHICS (not under DOS), the pointer can be obtained from the subpicture routines. Throughout this manual, the pointer is referred to as CNAME.

A description of each of the subpicture routine calls follows. The code-generating capability of a number of these calls can be accessed by the routines PLOT and REPLOT. The LINE routine, for example, is accessed by a call to PLOT or REPLOT with a selection argument of 1. The description in this chapter is directed toward the subpicture call, but applies equally to the corresponding PLOT and REPLOT calls.

2.3 LINE SUBROUTINE

The LINE subroutine adds to the specified display file the display commands necessary to draw a line (beam intensified) or move the beam (not intensified) through a specified displacement from the current beam position. RSX GRAPHICS requires that the random-vector hardware option be present.

The calling sequence is:

```
CALL LINE (IDX,IDY[,INT[,PNAME(1)[,CNAME]]])
```

where: The enclosing brackets, "[" and "]", indicate an optional argument.

IDX is the integer X-axis displacement in raster units.

IDY is the integer Y-axis displacement is raster units.

INT indicates whether the line is to be intensified. (For nonzero INT, the line is visible. For INT=0, the line is not visible. If the INT argument is omitted, a nonzero INT is assumed by default.)

PNAME(1) is the address of the display file. When the PNAME(1) argument is not provided, the generated display code is added to the display file to which code was last added.

CNAME, when present, is the output argument used for the return of an edit address.

Note that foresight is required to edit over a group of commands. CNAME is included in the argument list. The graphics system places the address of the command group in CNAME. This CNAME is then used as an argument in the call to REPLOT to write in the new command group. See Chapter 4 for more detail.

The square brackets are used to indicate which arguments can be omitted. (The square brackets themselves do not appear in any of the final calling sequences). Those arguments, (and commas) appearing between any matched [and] can be omitted. Thus, for this call, the user may provide from 2 to 5 arguments. The two arguments would be IDX and IDY. Three arguments would be IDX, IDY, and INT. Four arguments would be IDX, IDY, INT, and PNAME(1). Five arguments would be IDX, IDY, INT, PNAME(1), and CNAME.

NOTE

The discussion of the use of square brackets is applicable to the description of all argument lists contained in this manual.

The LINE call to the graphics system adds one or two locations to the display file. If either IDX or IDY is zero, or their absolute magnitudes are equal, the basic vector hardware command (requiring one location) is used. Note that IDX and IDY are truncated to ten bits (with retention of sign) without any warning to the user. If both IDX and IDY are zero, no code is generated. If IDX and IDY define any other line, the random vector hardware command (requiring two locations) will be used.

2.4 TEXT SUBROUTINE

The TEXT subroutine adds to the specified subpicture the VT-15 hardware commands necessary to display a string of 5/7 ASCII. Such a string will usually be generated by Hollerith data statements. The string is displayed starting at the current beam position. The standard text font is displayed on a 10 by 14 raster unit matrix (assuming hardware scale to be 0). With the exception of certain control characters, each character causes the beam to move 14 raster units to the right (positive X-axis). The calling sequence is:

```
CALL TEXT (STR(1),N[,PNAME(1)[,CNAME]])
```

where: STR(1) is the address of the text string and is normally a real array.

N is the integer number of characters of the string to be displayed. When N is nonzero, the graphics system inserts an ALT MODE after the Nth character in the string as a stop-code for the VT-15 hardware character generator. When N is zero, no ALT MODE is placed. This last feature would be used, for instance, when one wished the same text string to appear in two different places on the screen, or the user had placed the ALT MODE himself in some way.

PNAME(1) is the display file address; when this argument is omitted, the code is placed in the display file to which code was last added.

CNAME is the output argument for the return of the edit address.

Three locations are added to the display file:

```
CHARS* .+2
DSKP
address of string
```

where CHARS* is an indirect reference to a text string for the hardware character generator, and DSKP is a display skip.

Three warnings are in order here. First, space must be provided at the end of the text string to be displayed to contain the ALT MODE placed by the graphics system. One technique is to place (but not count for N) some extra character (e.g., q or z) at the end of the string. Second, the placement of the ALT MODE will destroy from 1 to 5 characters following the string requested for display. Therefore, if the first part of a longer string is displayed, that longer string will no longer be intact. Third, if the ASCII data is from some input/output device, it may contain non-printing characters such as carriage return and line feed. These characters must be counted for N.

The following FORTRAN statements show the use of the TEXT routine to reference the text '153 ASSABET RD.' from display file IFILE.

```
DIMENSION ADDR(4)
DATA ADDR(1)/5H153 A/,ADDR(2)/5HSSABE/
DATA ADDR(3)/5HT RD./,ADDR(4)/1HZ/
IFILE(1)=0
CALL TEXT (ADDR(1),15,IFILE(1))
```

2.5 COPY SUBROUTINE

The COPY subroutine generates graphics code so that one display file can call another as a subroutine. This allows the construction of complex display images from simple building blocks. The display files may not be recursively linked. The calling sequence is:

```
CALL COPY (RST,PNAME1(1)[,PNAME(1)[,CNAME]])
```

where: RST, when nonzero, requests that the hardware SAVE-RESTORE instructions be used to save display parameters through the subroutine call. When RST is zero, the SAVE-RESTORE option is not used.

PNAME1(1) is the address of the display file to be called.

PNAME(1) is the address of the calling file; when PNAME(1) is not provided by the user, the last display file to which code has been added is used.

CNAME, when present, will be filled with an edit address.

When the SAVE-RESTORE option is not used, three locations are added to the display file:

```
DJMS* .+2
DSKP
address of PNAME+1
```

When the SAVE-RESTORE option is used, six locations are added:

```
SAVE .+4
DJMS* .+2
DJMP .+3
address of PNAME1+1
status word stored here
RSTR .-1
```

In the above code, DJMS* is the display subroutine jump indirect instruction, DSKP is the display skip instruction, SAVE is the display save status to memory instruction, DJMP is the display jump, and RSTR is the restore status from memory instruction.

The PRAMTR call is used to establish the settings of such hardware features as SCALE, INTENSITY, BLINK, etc. If SAVE-RESTORE is specified in a COPY call, then the settings of the hardware features are saved prior to the subroutine call and restored afterwards. This allows the settings to be unaffected by any action of the called subroutine. If SAVE-RESTORE is not specified, any settings changed by the subroutine will remain changed.

At the time of the COPY call, PNAME1(1) need not yet be a defined subpicture. At the time that the display processor executes the code, however, it must be. A typical call to the COPY routine could be:

```
CALL COPY (0,IWNDOW(1),IHOUSE(1))
```

This call does not use the hardware SAVE-RESTORE option; the subpicture IHOUSE calls the subpicture IWNDOW.

2.6 PRAMTR SUBROUTINE

The PRAMTR subroutine places code in a display file to control the following hardware options: (see GRAPHIC-15 Reference Manual for a more complete description).

SCALE (0-15) - Controls the displayed size of a picture element. For a line (characters are similar) of 10 raster units; a scale of 0 will produce a ten unit line; a scale of 1, 20; a scale of 2, 30; etc. For the GRAPH routine, a scale of 0 will produce a zero distance between data points; a scale of 1 will produce a 1 raster separation; a scale of 2 will produce a 2 raster separation; etc.

INTENSITY (0-7) - Controls the intensity of the picture at eight different levels. The user may wish to accentuate certain portions of the picture; points require a somewhat higher intensity than lines; a picture executing at 30 times per second (see SYNC) takes a higher intensity than one executing at 60 times per second. The display of many items at high intensity at the same point on the screen may damage the phosphor.

LIGHT PEN ENABLE (0-1) - Controls the light pen enable (on or off) of those portions of the picture until the next such instruction is executed. A user may wish to obtain light pen hits only on certain portions of his displayed picture.

WARNING

> The internal structure of this feature is far different than under DOS, requiring one or two more locations in the display file (see Chapter 7 for details) when the light pen is turned on. Those who have adjusted their display file arrays under DOS to exactly the right size must readjust the display file size for use in RSX Graphics.

BLINK (0-1) - Controls whether that portion of the displayed image up to the next BLINK command will blink at 4 times per second.

DASH (0-3) - Controls whether lines (and the lines in characters) are dashed or solid.

SETTING	RASTERS
0	ALL ON
1	3 ON 1 OFF
2	4 ON 2 OFF
3	4 ON 4 OFF

OFFSET (0-1) - The offset area is a thin vertical rectangle of screen to the right of the normal 1024x1024 working area. When OFFSET is set to one, all coordinates are based from the lower left of the thin vertical rectangle rather than the lower left corner of the 1024 square. The offset area is normally used for control information such as light pen buttons. A light pen button is some graphic element (text string, square, etc.) that is used to notify the program that the scope user desires some particular action when a light pen hit occurs on that particular element.

ROTATE (0-1) - When on, causes the X and Y axes to be swapped, causing a rotation of 90 degrees counter-clockwise. This is mostly used for rotating text strings for labeling graphs, etc. Note, do not rotate arbitrary vectors.

NAME REGISTER (0-127) - This feature is used to identify portions of the picture upon detection of a light pen hit. The NAME REGISTER is set to different values for different portions of the picture by the user. When a light pen hit occurs, the NAME REGISTER setting is returned, allowing easy identification of the element receiving the light pen hit.

WARNING

The NAME REGISTER settings 120-127 are used by the TRACKING routine; it is recommended that the user not use these values.

SYNC (0-1) - This feature is no longer under user control under the RSX system; SYNC is always on. The SYNC parameter code (see Table 2-1) and its corresponding argument are accepted by PRAMTR and ignored so that existing DOS programs using SYNC can be run without causing argument errors. SYNC locks the execution of the display processor to power line frequency.

NOTE

The following discussion assumes 60 cycle power. For those users with 50 cycle power, the numbers should be changed accordingly.

This means that the user's display is executed 60 or 30 times per second. This synchronization prevents a continuous change of picture intensity with picture size, prevents phosphor damage from very small display files, and prevents noise in the power lines from making the picture 'swim'. A disadvantage is that 30 times-per-second execution causes the picture to 'flicker' because the phosphor of the tube becomes appreciably dimmer in the 1/30 of a second as it waits to be illuminated again. Another interesting problem occurs if the picture is right on the border between execution at 60 times and 30 times. Here the execution rate may well depend on the loading of the RSX system. This type of picture instability is very uncomfortable for the viewer. It is recommended that if this condition should arise, the user put some sort of non-visible display code in his picture to force execution entirely into the 30 times per second domain. The calling sequences for one hardware feature, or for some number of hardware features is shown in the calls below.

```
CALL PRAMTR (SELECT,VALUE[,PNAME(1)[,CNAME]])

CALL PRAMTR (SELECT,VALUE1,VALUE2,,,[,PNAME(1)[,CNAME]])
```

where: SELECT argument, described below, tells the graphics system which of the hardware features are desired.

VALUE is the value for each hardware feature selected.

PNAME(1) is the address of the display file into which the display code is placed. If PNAME(1) is not provided, the code is placed in the display file into which code was last placed.

In Table 2-1 each hardware feature is given a code. The SELECT argument is the sum of the codes of the features desired. Allowable ranges for the values for each feature are discussed earlier in this paragraph and are summarized in Table 2-1. There must be one and

exactly one VALUE argument for each feature requested in the SELECT argument. The values must furthermore be in the order given in Table 2-1.

Table 2-1

Display Parameter Settings

Parameter	Code	Settings
SCALE	1	0(SMALL) TO 15(LARGE)
INTENSITY	2	0(LOW) TO 7(HIGH)
LIGHT PEN	4	0(OFF) AND 1(ON)
BLINK	8	0(OFF) AND 1(ON)
DASH	16	0(SOLID) TO 3(FINEST DASH)
OFFSET	32	0(OFF) AND 1(ON)
ROTATE	64	0(NORMAL) AND 1(X AND Y AXES SWAPPED)
NAME REG.	128	0(LOW) TO 127(HIGH)
SYNC	256	0(OFF) AND 1(ON)

The PRAMTR subroutine adds one to six commands to the display file, depending on the features requested. Only one to four locations were required under DOS. Differences in the length of the generated code may occur with the SYNC feature, which will generate no code under RSX. Differences will definitely occur (more code under RSX) when a request is made to enable the light pen. Rather than simply enabling the light pen, it is necessary under RSX to make a subroutine call to the VT-15 handler. This subroutine call requires more code in the display file. Therefore, it may be necessary to increase array lengths in moving programs from DOS to RSX.

Each of the features requested from PRAMTR remains in effect until specifically changed, even if the display processor moves into other display files. Note that PLOT and REPLOT with a select argument of 2 are entirely equivalent to PRAMTR. Under DOS the feature requested would remain in effect even when the display processor started in executing the beginning of the 'main' file again. Under RSX the display processor always enters the 'main' file with the features set to specific default values. The default values are 4 for intensity, and zero for all other features (except for SYNC which is always on).

The user should be very careful in the use of the PRAMTR subroutine. A miscount of the number of VALUE arguments can have disastrous effects.

The following sample code shows the use of the PRAMTR subroutine for the specification of a single feature:

```
CALL PRAMTR (2,7,IHOUSE(1),IEDIT)
```

This call sets the intensity of the display to its highest value, 7, at the current end of the code in the display file IHOUSE. An edit address is returned into the variable IEDIT. The following is a multiple-feature statement:

```
C  CODE SETTINGS FOR ASSIGNMENT BITS
       ISCALB=1
       INTB=2
       LPENB=4
```

```
      CALL PRAMTR (ISCALB+INTB+LPENB,0,4,1)
C   OR, THE SAME THING WITH A NUMBER INSTEAD
      CALL PRAMTR (7,0,4,1)
```

In this case the scale is set to 0, the intensity is set to 4, and the light pen is enabled. The code is placed in the display file to which code was last added. In general, code will be placed in a single display file rather than a piece here and a piece there, so that PNAME will often be defaulted, i.e., not provided.

In XVM systems, a special marker no-op will be placed in the display file if CNAME was specified to the PRAMTR CALL. At this point the display file is one location longer than in previous systems. If a display element other than a PRAMTR or a PLOT (2,,,[CNAME]) is added to the display file, this special marker is written over, reclaiming the space. (This other group will serve to terminate the PRAMTR group for purposes of REPLOT'ing and DELETE'ing).

2.7 GRAPH SUBROUTINE

The GRAPH subroutine adds to the specified display file the code necessary to display an array of positive integer data in graphical form. One coordinate of the display (usually Y) is set equal to each data point in turn. The other coordinate is automatically incremented by the hardware after each data point. From the discussion of the SCALE feature, this increment is equal to the SCALE setting, so that for a scale of 0 the whole graph is collapsed onto a single X value. For a scale of 1 the increment is one raster unit; for a scale of 2 the increment is two raster units, etc. The beam is left positioned one increment past the last data point. Note that the axes and labeling for the graph must be provided separately; this routine deals only with data. The calling sequence is:

```
CALL GRAPH (DATA(1),N[,A[,PNAME(1)[,CNAME]]])
```

where: DATA(1) is the address of the positive integer data to be plotted. This data will be truncated to ten bits without any warning to the user.

N is the number of data points to be plotted.

A is the axis. For a normal Y vs. X plot, A is zero; for an X vs. Y plot, A is nonzero. When A is not provided as an argument, the plot is the normal Y vs. X.

PNAME(1) is the display file into which the graphical data is to be placed. If a PNAME(1) argument is not provided, the data is placed into the display file into which code was last written.

CNAME is the output argument for the return of an edit address.

The code generation is handled differently than under DOS to allow editing (REPLOT). Under RSX a two location bookkeeping header is placed prior to the graphical code. Under both DOS and RSX, each data point requires one core location to display it, using the hardware graph-point instruction. The bookkeeping header under RSX is a display skip followed by a count of the total number of locations

placed into the display file. Note, prior to XVM systems, the bookkeeping header was placed only if there were 7 or more data points. This means that XVM/RSX display files will grow by two locations, compared to previously, if GRAPH calls with less than 7 points are used. The following sample code plots 20 points from array IX into the present display file. The points are spaced along the X axis in increments of 2 raster units.

```
CALL PRAMTR (1,2)

CALL GRAPH (IX(1),20)
```

2.8 BLANK SUBROUTINE

The BLANK subroutine changes code in a display file to prevent the displaying of any copy of the specified display file. The length of the display file is not changed by this operation. The calling sequence is:

```
CALL BLANK (PNAME(1))
```

where: PNAME(1) is the display file to be BLANK'ed.

NOTE

> The PNAME(1) argument must be provided for this call, and cannot be defaulted. The PNAME(1) provided here does not count as 'subpicture into which code was last placed'. That remains what it was prior to this call.

The operation of the BLANK subroutine is to swap the contents of the first location (after the DJMS entry point) and last location of the referenced display file. Thus, the first instruction executed by the VT-15 processor upon entering the display file is the DJMP* to exit from the display file.

Some restrictions should be noted. As under DOS, PNAME should be a defined display file at the time that BLANK is called. As under DOS, the DYSET-DYLINK routines described in Chapter 6 should not operate on BLANK'ed files. As under DOS, BLANK'ing a file that has already been BLANK'ed is a no-op. In contrast to DOS, code can be added to a BLANK'ed subpicture. REPLOT'ing on the BLANK'ed subpicture, not possible under DOS, should be done with care. Clearly, the first group of commands in the display file, which now is holding the return jump, cannot be changed.

WARNING

> BLANK requires an I/O operation to the handler to prevent the VT-15 from executing beyond the end of the BLANK'ed file. This I/O operation cannot be immediately honored if there is an outstanding LTORPB (see Chapter 5). Upon the completion of the LTORPB, the BLANK subroutine will be completed.

The use of the BLANK routine:

```
CALL BLANK (IPIC(1))
```

2.9 UNBLNK SUBROUTINE

The UNBLNK subroutine reverses the action of the BLANK subroutine, allowing the subpicture to again be displayed. The calling sequence is:

```
CALL UNBLNK (PNAME(1))
```

PNAME(1), as in the BLANK subroutine, must be provided as an argument, and is not remembered as the default subpicture. The effect of this call is to swap back the locations swapped by BLANK, restoring the display file to its original configuration. If the file has not been BLANK'ed the call to UNBLNK is a no-op. Unlike the BLANK subroutine, UNBLNK is not affected by outstanding calls to LTORPB.

The following sample code will reverse the action of the example given for BLANK:

```
CALL UNBLNK (IPIC(1))
```

2.10 POINT SUBROUTINE

The POINT routine places in the specified display file the necessary code to absolutely position the beam. The final point may optionally be intensified. The calling sequence is:

```
CALL POINT (IX,IY[,INT[,PNAME(1)[,CNAME]]])
```

where: IX is the positive integer absolute X-position to which the beam is to be moved.

IY is the positive integer absolute Y-position to which the beam is to be moved. IX and IY are truncated to ten bits without any warning to the user.

INT=nonzero, point is intensified; INT=0, point is not intensified. If INT is not provided the point is not intensified.

PNAME(1) is the display file address for that display file to contain the code. When PNAME(1) is not provided, the code is placed in that subpicture into which code was last placed.

CNAME, if present, is used for the return of an edit address.

The code generated by a call to this routine requires two display file locations. The first is the hardware command to position the beam along the Y-axis, and the second to position the beam along the X-axis. Intensification is controlled by a bit in the second instruction. The following sample code:

```
CALL POINT (512,512,0,IFILE(1),IEDIT)
```

positions the beam to the center of the screen (512,512). The point is not intensified. The code is added to the display file beginning at IFILE. An edit address is returned into IEDIT.

2.11 ANY SUBROUTINE

The ANY subroutine places a user-provided array of display code into the specified subpicture file. The calling sequence is:

```
CALL ANY (ARRAY(1),N[,PNAME(1)[,CNAME]])
```

where: ARRAY(1) is the starting address of an integer array of VT15 display command code provided by the user and is represented as a subscripted variable.

N is the number of elements of that array that are to be moved into the display file.

PNAME(1) is the starting address of the display file into which the display commands are to be placed. PNAME(1) and ARRAY(1) may not refer to the same array. When PNAME(1) is not provided by the user, the display code is placed into the display file into which code was last placed.

CNAME, when present, is the output argument used for the return of an edit address.

The user code is placed unchanged into the display file. The user code is preceded by a two-word bookkeeping header identical to that described in section 2.7 for the GRAPH subroutine.

The graphics system does not allow the user to access all of the hardware features of the VT15. The intent of the ANY subroutine is to allow users to write specialized code using "unused" features, while still providing the overall convenience of the FORTRAN environment. (The user should be careful when using ANY, as the possibility of the VT15 processor "escaping" because of undebugged code is increased considerably.) Examples of ANY subroutines are single-character handling programs, such as editors, that can have one character right justified per word. This word, when executed by the VT15 processor, displays the single character. Applications of this type can include display of user terminal input, where the characters are relatively few in number and can easily be changed.

2.12 CIRCLE SUBROUTINE

The CIRCLE subroutine is used by Graphics programs to generate code to approximate arcs and circles. In RSX PLUS III, and XVM/RSX the calling arguments remain the same. However, at the conclusion of the arc or circle, the beam is returned to the center of the circle, not left at the edge as in RSX PLUS. The calling sequence is:

Form: CALL CIRCLE (R,THETA,GAMME,ANG,ISUB)

Where:
- r is the radius of the circle in floating-point raster units
- theta is the starting angle in floating-point degrees
- gamma is the ending angle in floating-point degrees
- ang is the angle subtended by each side of the polygon in floating-point degrees
- isub is the name of the integer subpicture in which the circle or arc is placed

See example in Figure 2-2.

The call to the CIRCLE subroutine has no effect if ANG is less than 0.001 degrees (absolute) or if R is less than one raster unit. The difference between GAMMA and THETA is reduced modulo 360, and both are measured counter-clockwise from the positive X axis. If ANG is positive, circles are drawn counter-clockwise from THETA to GAMMA. A full circle is drawn if THETA and GAMMA are within 0.001 degrees (modulo 360 degrees). The maximum number of polygon sides allowed is 360, even at the expense of not completing the requested circle or arc. It is possible for GAMMA to be less than THETA. If the user wishes, for example, he can draw an arc counter-clockwise from 20 degrees around to 10 degrees. Note that the previous contents of the display file ISUB are destroyed by this call.

2.13 ROTATE SUBROUTINE

The ROTATE subroutine is a FORTRAN subroutine which performs rotations of arrays of X-Y-Z data about the X,Y, or Z axes, or combinations of these axes. The user can control the angle of rotation. The user's array is modified by the ROTATE subroutine which uses the same left-handed system that is used throughout the graphic software:

X, horizontal movement, positive to the right

Y, vertical movement, positive is up

Z, perpendicular to the screen, positive into screen

The calling sequence is:

```
CALL ROTATE (N,IA,IB,IC,X,Y,Z,SINA,COSA)
```

where: N is the number of data points to be rotated.

IA nonzero, rotate about Z axis.

IB nonzero, rotate about Y-axis.

IC nonzero, rotate about X-axis.

X is the address of the array of X-positions.

Y is the address of the Y-position array.

Z is the address of the Z-position array.

SINA is the sine of the angle through which the arrays are to be rotated, in single precision real format.

COSA is the cosine of the same angle.

When two axes of rotation are specified, the rotation occurs about the 45 degree line of the plane defined by those two axes. When all three axes are specified, the rotation occurs about a line 45 degrees to all axes.

WARNING

The values in the X, Y, Z arrays must be in floating-point format. The specification of these three arrays in the argument string must be in the form XARRAY rather than XARRAY(1). In a multi-axis rotation, this routine rotates about one axis by the angle specified, and then by the same angle about the other axis. The resulting overall angle of rotation is not that angle originally specified.

The operation of the ROTATE routine replaces the X, Y, and Z values in the arrays, which are taken to be the values before rotation, with the values corresponding to the positions after rotation. (Since the initial values are replaced, beware of accumulated rounding errors.) The user has the responsibility of converting this data back to integer format and making those calls to the graphics system which are necessary to display this data. The coordinate point 0,0,0 is taken to be the center of the rotation. The user can control the displayed center point by making an initial set point, and then displaying the figure in relative form. Care should be taken in rotating large, or off-center figures. It is a hardware feature of the VT-15 that displayed items that are in part off-screen are not intensified. Thus, the edge of the screen will not serve to 'clip' figures that rotate partially off-screen.

2.14 EXAMPLES

The first example (Figure 2-1) is a simple program utilizing some of the calls described in this chapter to display four squares and a text string. The DINIT and LTORPB calls used in the examples are described in Chapter 5. Figure 2-2 is an example that highlights the CIRCLE subroutine.

```
C       SIMPLE DEMO TO DISPLAY FOUR SQUARES
C
        LOGICAL IB(6)
        DIMENSION MAIN(40)
        DIMENSION ISQ(10)
        DIMENSION TXT(5)
        DATA TXT(1),TXT(2),TXT(3),TXT(4)/5HHERE ,
        15HARE 4,5H SQUA,5HRES   /
C
C
        ISQ(1)=0
        MAIN(1)=0
C  SQUARE SUBROUTINE
        CALL LINE (100,0,1,ISQ(1))
        CALL LINE (0,100)
        CALL LINE (-100,0)
        CALL LINE (0,-100)
C  THE MAIN CALLING ROUTINE
C   FIRST THE TEXT, POSITION BEAM
        CALL POINT (100,100,0,MAIN(1))
C
C AND TURN ON THE PROCESSOR
        CALL DINIT(MAIN(1))
C
C  MAKE SCALE EQUAL TO 1 FOR TEXT
        CALL PRAMTR (1,1)
C
C  THE TEXT
        CALL TEXT(TXT(1),18)
C
C SCALE BACK TO ZERO  FOR SQUARES
        CALL PRAMTR (1,0)
C
C  NOW FOUR TIMES, A SET BEAM, AND CALL SQUARE
        CALL POINT (200,500)
        CALL COPY (0,ISQ(1))
        CALL POINT (200,800)
        CALL COPY (0,ISQ(1))
        CALL POINT (700,500)
        CALL COPY (0,ISQ(1))
        CALL POINT (700,800)
        CALL COPY (0,ISQ(1))
C
C  NOW WAIT FOR AN INTERRUPT TO LEAVE PICTURE
        CALL LTORPB(LPX,LPY,NAME,IB,IW,1)
        STOP
        END
```

Figure 2-1
Four Square Display Example

```
C   TRY A CIRCLE
C
      DIMENSION IM(20),IC(1000)
C
      IM(1)=0
      IC(1)=0
C
C   PLACE BEAM IN SCREEN CENTER
C
      CALL POINT (512,512,1,IM(1))
C
C   AND PLACE SOME ITEM IN IC SO WE CAN CALL IT
C    SINCE WE ARE GOING TO TURN ON THE SCOPE!
C
      CALL LINE (10,0,0,0,IC(1))
C
C   LARGE ARRAY FOR CIRCLE CALLED FROM IM
C
      CALL COPY (0,IC(1),IM(1))
C
C   TURN ON THE VT-15 PROCESSOR
C
      CALL DINIT (IM(1))
C
C   GET CIRCLE PARAMETERS FROM TTY
C    12 IS INPUT LUN SLOT
C
 23   READ (12,100) IR,IT,IG,ID
C
C   CONVERT INTEGER TO FLOATING
C
       R=IR
       T=IT
       G=IG
       D=ID
C
C   PLACE LINES TO APPROX. CIRCLE IN IC
C
      CALL CIRCLE (R,T,G,D,IC)
C
C   GO WAIT FOR NEXT CIRCLE
C
      GO TO 23
C
 100  FORMAT (4I3)
      STOP
      END
```

Figure 2-2
CIRCLE Subroutine Example

The next four examples deal with the technical and conceptual differences between the DOS and RSX graphics packages. The example in Figure 2-3 is copied from the DOS graphics manual and will not present a picture under RSX. The example in Figure 2-4 shows the minimum modification that has to be made to the example in Figure 2-3 to run under RSX. The example in Figure 2-5 shows how an RSX graphics programmer might generate the same picture without the calls of Chapter 3. The example in Figure 2-6 shows how the program might be coded so that all the graphics code is in one display file. The example in Figure 2-7 demonstrates the manipulation of PRAMTR settings. The example in Figure 2-8 illustrates the use of the ROTATE subroutine. Figure 2-8A is the FORTRAN listing of the example. Figure 2-8B is the MACRO listing of the random number generator designed for the example.

```
C
C  ARRAY INITIALIZATION
      INTEGER SINWV(300),Y(200)
      DIMENSION TITL(10),MAINFL(20)
      DATA TITL(1),TITL(2),TITL(3),TITL(4)/5HTHIS ,
     1 5HIS A ,5HSINE ,4HWAVE/
C
C  SET UP INTEGER ARRAY OF VALUES TO BE PLOTTED
C
10    X=0
      DO 20 I=1,200
      Y(I)=IFIX(SIN(X)*256.)+512
      X=X+.0628
20    CONTINUE
C
C  SET UP SUBPICTURE TO PLOT THOSE VALUES
C
      SINWV(1)=0
      CALL PRAMTR(3,0,7,SINWV(1))
      CALL LINE(1000,0,1)
      CALL LINE(-1000,0,0)
      CALL LINE(0,250,0)
      CALL LINE(0,-500,1)
      CALL LINE(0,250,0)
      CALL PRAMTR (1,4)
      CALL GRAPH (Y(1),100,0)
      CALL GRAPH (Y(101),100,0,SINWV(1))
C
C  SET UP MAIN FILE TO DISPLAY THE GRAPH
C  (MAIN FILE CALLS BELOW, DESCRIBED IN CHPT. 3)
C
      MAINFL(1)=0
      CALL DINIT (MAINFL(1))
      CALL SETPT (10,512)
      CALL PLOT (0,0,SINWV(1))
      CALL SETPT (100,100)
      CALL PLOT (2,1,1)
      CALL PLOT (3,TITL(1),19)
      CALL DCLOSE
      STOP
      END
```

Figure 2-3
DOS Sine Wave Program Example

```
C
C  ARRAY INITIALIZATION
          LOGICAL IB(6)
          INTEGER SINWV(300),Y(200)
          DIMENSION TITL(10),MAINFL(20)
          DATA TITL(1),TITL(2),TITL(3),TITL(4)/5HTHIS ,
          1 5HIS A ,5HSINE ,4HWAVE/
C
C  SET UP INTEGER ARRAY OF VALUES TO BE PLOTTED
C
10        X=0
          DO 20 I=1,200
          Y(I)=IFIX(SIN(X)*256.)+512
          X=X+.0628
20        CONTINUE
C
C  SET UP SUBPICTURE TO PLOT THOSE VALUES
C
          SINWV(1)=0
          CALL PRAMTR(3,0,7,SINWV(1))
          CALL LINE(1000,0,1)
          CALL LINE(-1000,0,0)
          CALL LINE(0,250,0)
          CALL LINE(0,-500,1)
          CALL LINE(0,250,0)
          CALL PRAMTR (1,4)
          CALL GRAPH (Y(1),100,0)
          CALL GRAPH (Y(101),100,0,SINWV(1))
C
C  SET UP MAIN FILE TO DISPLAY THE GRAPH
C  (MAIN FILE CALLS BELOW, DESCRIBED IN CHPT. 3)
C
          MAINFL(1)=0
          CALL DINIT (MAINFL(1))
          CALL SETPT (10,512)
          CALL PLOT (0,0,SINWV(1))
          CALL SETPT (100,100)
          CALL PLOT (2,1,1)
          CALL PLOT (3,TITL(1),19)
C
C  THE ONLY CHANGE NECESSARY IS A WAIT
C  OF SOME SORT TO RETAIN THE PICTURE
C  THE "END" KILLS THE PICTURE UNDER RSX
C  USE LTORPB FOR THE WAIT
C
          CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  ANY PUSH BUTTON OR LIGHT PEN HIT WILL
C  CAUSE THE PROGRAM TO EXIT
C
C  IF YOU FORGET DCLOSE, THE HANDLER WILL TURN OFF TUBE
          CALL DCLOSE
          STOP
          END
```

Figure 2-4
DOS Sine Wave Program Converted To RSX

```
C
C  ARRAY INITIALIZATION
        LOGICAL IB(6)
        INTEGER SINWV(300),Y(200)
        DIMENSION TITL(10),MAINFL(20)
        DATA TITL(1),TITL(2),TITL(3),TITL(4)/5HTHIS ,
        1 5HIS A ,5HSINE ,4HWAVE/
C
C  SET UP INTEGER ARRAY OF VALUES TO BE PLOTTED
C
10      X=0
        DO 20 I=1,200
        Y(I)=IFIX(SIN(X)*256.)+512
        X=X+.0628
20      CONTINUE
C
C  SET UP SUBPICTURE TO PLOT THOSE VALUES
C
        SINWV(1)=0
C
C  PLACE ABSOLUTE BEAM POSITION WITH REST OF GRAPH
C
        CALL POINT (10,512,0,SINWV(1))
        CALL PRAMTR (3,0,7)
        CALL LINE(1000,0)
        CALL LINE(-1000,0,0)
        CALL LINE(0,250,0)
        CALL LINE(0,-500)
        CALL LINE(0,250,0)
        CALL PRAMTR (1,4)
        CALL GRAPH (Y(1),100)
        CALL GRAPH (Y(101),100)
C
C  SET UP 'MAIN' FILE TO CALL THE GRAPH
C
        MAINFL(1)=0
        CALL DINIT (MAINFL(1))
        CALL COPY (0,SINWV(1),MAINFL(1))
        CALL POINT(100,100)
        CALL PRAMTR (1,1)
        CALL TEXT (TITL(1),19)
C  THE ONLY CHANGE NECESSARY IS A WAIT
C  OF SOME SORT TO RETAIN THE PICTURE
C  THE "END" KILLS THE PICTURE UNDER RSX
C  USE LTORPB FOR THE WAIT
C
        CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  ANY PUSH BUTTON OR LIGHT PEN HIT WILL
C  CAUSE THE PROGRAM TO EXIT
C
C  IF YOU FORGET DCLOSE, THE HANDLER WILL TURN OFF TUBE
        CALL DCLOSE
        STOP
        END
```

Figure 2-5
RSX Sine Wave Program Eliminating Chapter 3 Calls

```
C
C   ARRAY INITIALIZATION
        LOGICAL IB(6)
        INTEGER SINWV(300),Y(200)
        DIMENSION TITL(10)
        DATA TITL(1),TITL(2),TITL(3),TITL(4)/5HTHIS ,
        1 5HIS A ,5HSINE ,4HWAVE/
C
C   SET UP INTEGER ARRAY OF VALUES TO BE PLOTTED
C
10      X=0
        DO 20 I=1,200
        Y(I)=IFIX(SIN(X)*256.)+512
        X=X+.0628
20      CONTINUE
C
C   ONE FILE FOR EVERYTHING
C
C   ZERO TOP-OF-FILE POINTER
        SINWV(1)=0
C
C   ESTABLISH ABSOLUTE BEAM POSITION
        CALL POINT (10,512,0,SINWV(1))
C
C   SCALE 0, INTENSITY 7
        CALL PRAMTR (3,0,7)
C
C   HORIZONTAL AXIS
        CALL LINE(1000,0)
C
C   MOVE BEAM BACK
        CALL LINE(-1000,0,0)
C
C   MOVE BEAM UP
        CALL LINE(0,250,0)
C
C   TURN ON VT-15 ANY OLD TIME
        CALL DINIT (SINWV(1))
C
C   VERTICAL AXIS
        CALL LINE(0,-500)
C
C   MOVE BEAM BACK
        CALL LINE(0,250,0)
C
C   SET SCALE TO FOUR FOR GRAPH POINTS
        CALL PRAMTR (1,4)
```

Figure 2-6
Sine Wave Program Written For Single Display File
(Sheet 1 of 2)

```
C
C  NOW TWO GROUPS OF 100 POINTS EACH
      CALL GRAPH (Y(1),100)
      CALL GRAPH (Y(101),100)
C
C  ABSOLUTE BEAM POSITION FOR TITLE
      CALL POINT(100,100)
C
C  SCALE BACK TO 1 FOR TITLE TEXT
      CALL PRAMTR (1,1)
C
C  AND PLACE TEXT
      CALL TEXT (TITL(1),19)
C
C  WAIT FOR ANY INTERRUPT BEFORE EXITTING,
C   A PUSH BUTTON WILL DO
      CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  IF YOU FORGET DCLOSE, HANDLER WILL TURN OFF TUBE
      CALL DCLOSE
      STOP
      END
```

Figure 2-6 (Cont.)
Sine Wave Program Written For Single Display File (Sheet 2 of 2)

```
C
C  TO SHOW SOME OF PRAMTR SETTINGS
C
      LOGICAL IB(6)
      DIMENSION IFILE(100)
C
      IFILE(1)=0
C
C
C  POSITION BEAM
      CALL POINT (300,300,0,IFILE(1))
C
C  NOW A LINE, PRAMTR SETTINGS OFF
C  ON ENTERING MAIN FILE, SO LINE NORMAL
      CALL LINE (200,0)
C
C  TURN ON DASH AND BLINK
      CALL PRAMTR (24,1,1)
C
C  AND A LINE TO DEMONSTRATE IT
      CALL LINE (0,200)
C
C  NOW TURN OFF DASH AND BLINK, AND
C  TURN ON OFFSET AND ROTATE
      CALL PRAMTR (120,0,0,1,1)
C
C  BEAM POSITION. NOTE THAT ROTATE
C  AFFECTS ONLY RELATIVE POSITIONING
      CALL POINT (10,300)
C
C  AND ROTATED LINE IN OFFSET AREA
      CALL LINE (200,0)
C
C  REMEMBER TO TURN ON SCOPE
      CALL DINIT (IFILE(1))
C
C  WAIT BEFORE EXITTING TO LEAVE PICTURE
      CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  WHEN WE GET IT, EXIT
      STOP
      END
```

Figure 2-7
Demonstration of PRAMTR Settings

```
C
C  ROTATING LINE EXAMPLE
C
      LOGICAL ID,LTORPB,IPB(6)
      DIMENSION X(10),Y(10),Z(10),MAIN(1000)
      DIMENSION IT(2),IT2(2)
      DIMENSION IQ(7),JX(10),JY(10)
C
C  EQUATES FOR TIME DELAYS
      IT(1)=2
      IT(2)=1
      IT2(1)=1
      IT2(2)=2
C
C  SIN AND COS OF 4.5 DEGREES
      SINQ=.07846
      COSQ=.99692
C
C  ADDITIONAL CORRECTION FOR 3 AXIS
      CORR=.00032
C
C  AXIS SELECTION ARRAY
      IQ(1)=1
      IQ(2)=5
      IQ(3)=4
      IQ(4)=3
      IQ(5)=7
      IQ(6)=2
      IQ(7)=6
C
C  MUST START TUBE NOW, SINCE WE CAN'T START
C  WHILE A LTORPB IS OUTSTANDING
      CALL DINIT (MAIN(1))
C
C  GET INITIAL BUTTON SETTTINGS
      CALL GETPSH (IPB)
C
C  SELECT 6 POINTS FOR ROTATION
 1    SH=0.
C  HOLD POINT FURTHEST FROM ORIGIN
      DO 100 I=1,6
      CALL RANDM (ZZ)
      X(I)=400.*ZZ-130.
      IF(X(I).LT.70.) X(I)=X(I)-140.
      CALL RANDM (ZZ)
      Y(I)=400.*ZZ-130.
      IF(Y(I).LT.70.) Y(I)=Y(I)-140.
      CALL RANDM (ZZ)
      Z(I)=400.*ZZ-130.
      IF(Z(I).LT.70.) Z(I)=Z(I)-140.
C
C  SUM OF SQUARES
      S=X(I)*X(I)+Y(I)*Y(I)+Z(I)*Z(I)
C
C  KEEP LARGEST
      IF(S.GT.SH) SH=S
C
```

Figure 2-8A
ROTATE Subroutine Example (FORTRAN Listing) (Sheet 1 of 3)

```
 100    CONTINUE
C
C  EACH AXIS IS +-(70. TO 270.)
C  NOW RATIO UP IF GOT SMALL PATTERN
C  HAVE TO KEEP ALL LINES ON SCREEN.
       R=SQRT(250000./SH)
C
C  NOW RATIO UP EVERYONE
       DO 101 I=1,6
       X(I)=X(I)*R
       Y(I)=Y(I)*R
       Z(I)=Z(I)*R
 101   CONTINUE
C
C  ROTATE ABOUT EACH A 7 AXIS COMBINATIONS
       DO 20 II=1,7
C
C
C  ROTATE ABOUT Z?
       IA=IQ(II).AND.4
C
C  ROTATE ABOUT Y?
       IB=IQ(II).AND.2
C
C  ROTATE ABOUT X?
       IC=IQ(II).AND.1
C
C  CORRECT ROTATE ROUTINE FOR MULTI-AXIS
       ZZ=3.
       IF(IA.EQ.0) ZZ=ZZ-1.
       IF(IB.EQ.0) ZZ=ZZ-1.
       IF(IC.EQ.0) ZZ=ZZ-1.
       ZZQ=SQRT(ZZ)
       SINCL=SINQ/ZZQ
       IF(ZZ.GT.2.5) SINCL=SINCL+CORR
       COSCL=1.-(1.-COSQ)/ZZ
C
C  REINIT MAIN FILE
       MAIN(1)=0
C
C
C  SET UP INTENSITY AND SCALE
       CALL PLOT (2,3,0,6)
C
C  NOW ROTATE 80 TIMES FOR IMAGE
       DO 10 IL=1,80
       CALL ROTATE (6,IA,IB,IC,X,Y,Z,SINCL,COSCL)
C
C  MAKE ALL INTEGER AT ONCE TO HELP ROUNDOFF
       DO 121 IO=1,6
       JX(IO)=X(IO)
       JY(IO)=Y(IO)
 121   CONTINUE
C
C  SET POINT FOR INITIAL POINT
       IX=JX(1)+512
       IY=JY(1)+512
       CALL SETPT(IX,IY)
```

Figure 2-8A (Cont.)
ROTATE Subroutine Example (FORTRAN Listing) (Sheet 2 of 3)

```
C
C  FIVE LINES CONNECTING 6 POINTS
        DO 122 IO=1,5
        IX=JX(IO+1)-JX(IO)
        IY=JY(IO+1)-JY(IO)
        CALL PLOT (1,IX,IY)
 122    CONTINUE
C
C  TIME DELAY BETWEEN EACH ROTATION
C  EXCEPT IF BUTTON 1 ON
        IF(IPB(1)) GO TO 10
        CALL MARK(IT,IEV)
        CALL WAITFR(IEV)
 10     CONTINUE
C
C  AND A BIGGER ONE BETWEEN EACH PICTURE
C  EXCEPT IF PUSH BUTTON 2
C  FIRST FIND OUT ABOUT BUTTON STATES
        CALL GETPSH (IPB)
        IF(IPB(2)) GO TO 27
        CALL MARK(IT2,IEV)
        CALL WAITFR(IEV)
C
C  IF PUSH BUTTON 3, HOLD THIS PICTURE
 27     CALL GETPSH (IPB)
        IF(IPB(3)) GO TO 777
C
C  IF PUSH BUTTON 6, EXIT
        IF(IPB(6)) GO TO 999
C
C  OTHERWISE CONTINUE AS NORMAL
 20     CONTINUE
C
C  IF THRU WITH 7 AXIS TRIES, GET NEW POINTS
        GO TO 1
C
C  NOW IF BUTTON 3 ON, WAIT FOR NEW HAPPENING WHILE
C  USER ENJOYS THE PICTURE
 777    ID=LTORPB(LPX,LPY,NAME,IPB,IW,1)
C
C   IF BUTTON 6 ON EXIT
        IF(IPB(6)) GO TO 999
C
C  IF BUTTON 3 STILL ON, GO WAIT AGAIN
        IF(IPB(3)) GO TO 777
C
C  OTHERWISE REJOIN LOOP
        GO TO 20
 999    STOP
        END
```

Figure 2-8A (Cont.)
ROTATE Subroutine Example (FORTRAN Listing) (Sheet 3 of 3)

```
/
/   RANDOM NUMBER GENERATOR
/
/   CALLING SEQUENCE
/
/         CALL RANDM (Z[,INIT])
/
/   Z IS RETURNED WITH A FLOATING POINT NUMBER
/   NORMALIZED BETWEEN 0 AND 1
/
/   INIT IS AN INTEGER QUANTITY USED TO
/   CHANGE THE INTIALIZATION OF THE
/   GENERATOR. THIS ARGUMENT IS OPTIONAL
/
/   THE MULTIPLIER IS 5^15
/   THE DIVISOR IS 2^35
/   THE STARTING NUMBER IS (2^19+INIT).OR.1
/
          .GLOBL RANDM
RANDM     0
/
/   SET UP POINTERS TO HANDLE ARG.'S
/
          LAC       RANDM
          IAC
          DAC       T
          LAC*      T
          SMA                 /SKIP IF ANOTHER INDIRECT
          JMP       .+3
          DAC       POINT
          LAC*      POINT
/
/   FIRST OF TWO WORDS OF FLOATING POINT
          DAC       POINT
          IAC
/
/   AND SECOND
          DAC       POINT1
/
/   NOW FIRST OF THREE MULTIPLIES
          LAC       NH        /HIGH 1/2 OF NUMBER
          MUL
ML        244615              /LOW 1/2 OF MULTIPLIER
          LACQ                /LOW 1/2 OF ANSWER
          DAC       NH        /BUILD NEW ANSWER
/
/   NOW CHECK FOR INIT ARGUMENT
          ISZ       T
          LAC       T         /DOES POINTER=JMP
          XOR*      RANDM     /LAST 13 BITS
          AND       (17777    /KEEP LOW 13
          SNA                 /SKIP IF ARGUMENT
          JMP       NOARG     /NOT ONE
/
/   THERE IS INIT ARGUMENT, IS THIS FIRST TIME
          LAC*      T         /
          SMA                 /SKIP IF EXTRA INDIRECT
```

Figure 2-8B
ROTATE Subroutine Example (Random Number Generator Coding) (Sheet 1 of 2)

```
        JMP     .+3
        DAC     T
        LAC*    T
        DAC     T       /NOW POINTS TO INIT
        LAC     FIRST   /MASK SO WE CAN OR NL
        AND*    T
NOARG   DZM     FIRST   /SET NO LONGER FIRST TIME
        XOR     NL      /XCPT FIRST, AC=0
        DAC     NL      /STORE RESULT
/
/  NOW LOW 1/2 NUMBER, HIGH 1/2 MULTIPLIER
        MUL
MH      343277          /HIGH 1/2 OF 5^15
        LACQ            /LOW 1/2 OF MULT.
        TAD     NH      /ASSEMLY OF HIGH 1/2 NEW NUMB.
        DAC     NH      /HOLD FOR NOW
/
/  NOW LOW 1/2 TIMES LOW 1/2
/
        LAC     ML
        MUL
NL      1               /INIT'S AT 1 AS DEFAULT
        TAD     NH      /HIGH 1/2 MULT TO HIGH 1/2 NUM
        AND     (377777 / MAKE IT POSITIVE
        DAC     NH      /THIS IS THE NEW HIGH HALF
        LACQ            /GET NEW LOW HALF
        DAC     NL      /PLACE IT
        LAC     NH      /NOW NORMALIZE
        NORM            /TO FLOATING POINT FORMAT
        DAC*    POINT1  /HIGH ORDER MANTISSA IN SECOND WORD
        LACQ            /NOW MAKE 9 BITS LOW ORDER MANTISSA
        AND     (777000 /WHICH WILL BE IN TOP OF FIRST WORD
        DAC     POINT1  /USE THIS AS TEMPORARY TO HOLD IT
        LACS            /GET STEP COUNT TO COMPUTE EXPONENT
        AAC     -34     /NEXT TWO TO NORMALIZE TO 1, AND MAKE
        CMA!IAC         /FORTRAN EXPECTED TWO-COMPLEMENT EXPONENT
        AND     (777    /STRIP TO LOW NINE BITS
        TAD     POINT1  /PUT TOGETHER TWO HALVES OF FIRST WORD
        DAC*    POINT   /AND RETURN IT TO CALLER
        JMP*    RANDM
POINT   0
POINT1  0
NH      2
FIRST   777776
T       0
```

Figure 2-8B (Cont.)
ROTATE Subroutine Example (Random Number Generator Coding) (Sheet 2 of 2)

CHAPTER 3

MAIN DISPLAY FILE ROUTINES

The calls presented in this chapter are supported by RSX graphics to maintain DOS graphics compatibility. The calls need not be used at all for graphics programs being created under RSX. The PLOT calls do, however, serve as a model for the REPLOT calls of Chapter 4.

Under DOS there are two distinctly different types of display files, main files and subpicture files. It is the intent of the RSX graphics package to remove this distinction as much as possible without making DOS programs incompatible with the RSX graphics system.

Under DOS the calls that operated on subpicture files had distinctly different capabilities than the calls that operated on main files. The main file was that file incorporating the VT-15 processor loop; subpictures were called as subroutines from it.

Under RSX all the code generating capability of the package is included in the subpicture calls, and they may indeed operate on the 'main' file. The main file is linked as a display subroutine from the VT-15 handler. The subpicture files are, in turn, linked as display subroutines from the main file.

Of the original DOS calls described in the 'MAIN DISPLAY FILE ROUTINES' chapter of the DOS graphics manual, the calls DINIT and DCLOSE are found in Chapter 5, Input-Output. The calls DELETE, REPLOT, and RSETPT are discussed in Chapter 4, Code Modification. The two remaining calls described in this chapter are: SETPT and PLOT.

SETPT - places in the main file the code to absolutely position the beam. The resulting point is not intensified.

PLOT - accesses the code generating capability (by a selection argument) of the subpicture calls LINE, TEXT, COPY, PRAMTR, GRAPH, POINT, and ANY. The GRAPH, POINT and ANY subroutine accessed by PLOT were not originally provided in the DOS graphics package. They have been added to the RSX graphics package to provide an internal compatibility between PLOT and REPLOT in the RSX system. The last file given as an argument to the routine DINIT is generally referred to as the main file. The code generated by PLOT is placed in the main file.

3.1 SETPT SUBROUTINE

The SETPT subroutine generates the code to absolutely position the beam. The resulting point is not intensified. The calling sequence is:

```
CALL SETPT (IX,IY[,CNAME])
```

where: IX is a positive integer absolute X-position to which the beam is to be located.

IY is a positive integer absolute Y-position to which the beam is to be located.

CNAME is the argument for the return of an edit address.

Both IX and IY are truncated to ten bits without any warning to the user. The code is placed in the display file last given as an argument to the DINIT routine. This file is referred to throughout this document as the 'main' file.

The operation of this subroutine is to place two locations of display code in the main file. The first location contains the VT-15 instruction to place the display beam at the correct absolute Y-position. The second contains the corresponding instruction for X-positioning.

3.2 PLOT SUBROUTINE

The PLOT subroutine accesses the display code generation capability of the subroutines COPY, LINE, PRAMTR, TEXT, POINT, GRAPH, and ANY. The first argument to all PLOT calls is a selection argument to describe which subroutine is to be accessed, as follows:

CODE	SUBROUTINE
0	COPY
1	LINE
2	PRAMTR
3	TEXT
4	POINT
5	GRAPH
6	ANY

Note that the subpicture routines can also be used to access the main file by supplying the mainfile as the file address.

3.2.1 Access Subroutine COPY

The calling sequence to access the COPY subroutine is:

```
CALL PLOT (0,RST,PNAME(1)[,CNAME])
```

where: the code 0 is the select argument indicating that this call to PLOT generates COPY code.

RST is the argument indicating whether the hardware SAVE-RESTORE option is to be used in the generated code.

PNAME1(1) is the address of the display file to be called as a subroutine.

CNAME is the output argument for the return of an edit address.

The code generated by the PLOT call is placed into the main file. See paragraph 2.5 for a more complete description of the code generated for the COPY subroutine. In the following call:

```
CALL PLOT (0,0,IHOUSE(1),IEDIT)
```

The first zero indicates COPY, the second that the hardware SAVE-RESTORE is not to be used, IHOUSE(1) is the subroutine called, and an edit address is returned to IEDIT.

3.2.2 Access Subroutine LINE

The calling sequence to access the LINE subroutine is:

```
CALL PLOT (1,IX,IY[,INT[,CNAME]])
```

where: the code 1 indicates the LINE code generating routine is to be accessed.

IX is the X-displacement in raster units.

IY is the Y-displacement in raster units.

INT indicates whether the line is to be intensified. (INT=nonzero, the line will be visible; INT=0, the line will not be visible. If INT is omitted, INT=nonzero is assumed by default.)

CNAME is the output argument for the return of an edit address.

The code generated by this call is placed in the main file. The square brackets indicate that the user may provide all five arguments, he may omit CNAME, or he may omit both INT and CNAME. See paragraph 2.3 for a more complete description of the code generated by the LINE subroutine. In the following call:

```
CALL PLOT (1,100,-200)
```

the 1 indicates that LINE code is to be generated. The delta X is 100 raster units and the delta Y is -200 raster units. The line is intensified because the INT argument is omitted. The code goes into the main file. No edit address is returned.

3.2.3 Access Subroutine PRAMTR

The calling sequences for a single feature specification, and multiple feature specification are shown, respectively, in the two calls that follow:

```
CALL PLOT (2,SELECT,VALUE[,CNAME])

CALL PLOT (2,SELECT,VALUE1,VALUE2,,,,[,CNAME])
```

where: The code 2 indicates the PRAMTR code is to be generated.

SELECT indicates which hardware features are to be activated.

VALUE indicates what setting is to be placed in the hardware instruction.

CNAME is the output argument for the return of an edit address.

See paragraph 2.6 for a more complete discussion of PRAMTR code generation, limits for VALUE arguments, etc. The following call illustrates the setting of the BLINK feature:

```
CALL PLOT (2,8,1)
```

The following call illustrates the simultaneous setting of the name register to a value of 47, and the intensity to 6:

```
CALL PLOT (2,130,6,47,JEDIT)
```

On completion of execution of the call, the argument JEDIT contains the address of the first of the two locations of code generated by this call.

Under XVM, if a CNAME argument is provided to this type 2 PLOT call, a special marker no-op is placed in the display file to terminate the display group. When additional information is placed in this display file by a Chapter 2 or Chapter 3 call, the marker no-op is reclaimed as display space unless the call is a PRAMTR call, or a type 2 PLOT call. In these two latter cases, the display file will remain one location longer because of the imbedded no-op. The display file is also a location longer if it is terminated by a type 2 PLOT call.

This marker no-op is necessary because the new CNAME format cannot distinguish one two-location parameter graphics element from two one-location parameter graphics elements for the purposes of REPLOTing.

3.2.4 Access Subroutine TEXT

The calling sequence to access the TEXT subroutine is:

```
CALL PLOT (3,STR(1),N[,CNAME])
```

where: The code 3 indicates that TEXT code is to be generated.

STR(1) is the address of the string of 5/7 ASCII to be displayed.

N is the number of characters to be displayed.

CNAME, when present, contains an edit address upon completion of the call.

See paragraph 2.4 for a more complete description of the assumptions and limitations of this type of code generation. The following call displays 15 characters of a string located in the array ZOT:

```
CALL PLOT (3,ZOT(1),15,ISAVIT)
```

The 3 indicates the TEXT code will be generated. The character string is in the array ZOT. The pointer (edit address) to the location of this group of commands is in the variable ISAVIT.

3.2.5 Access Subroutine POINT

The calling sequence to access the POINT subroutine is:

```
CALL PLOT (4,IX,IYE,INTE,CNAME))
```

where: The code 4 is the select argument to access the POINT code generating routine.

IX and IY are the integer X and Y values to which the beam is to be positioned.

INT is the intensity control argument. The point is intensified only if the INT argument is provided, and nonzero.

CNAME, when present, is for the return of an edit address.

This routine is somewhat redundant with SETPT. The intent is to supply intensified point capability without destroying compatibility with existing code using SETPT. See paragraph 2.10 for a more complete description of POINT code generation. The following sample call generates the code to position the beam to 200,300, and to intensify the resulting point:

```
CALL PLOT (4,200,300,1)
```

3.2.6 Access Subroutine GRAPH

The calling sequence to access the GRAPH subroutine is:

```
CALL PLOT (5,DATA(1),NE,AE,CNAME))
```

where: The code 5 indicates that GRAPH code is to be generated.

DATA(1) is the address of the array of positive integer data provided by the user.

N is the number of data points to be displayed.

A specifies the axis of the plot. It is a normal Y versus X plot unless the argument A is present and nonzero.

CNAME is the output argument for the return of an edit address.

See section 2.7 for a more complete description of GRAPH code generation.

3.2.7 Access Subroutine ANY

The calling sequence to access the ANY subroutine is:

```
CALL PLOT (6,ARRAY(1),N[,CNAME])
```

where: The code 6 indicates that the ANY subroutine is to be accessed.

ARRAY(1) is the user provided array of VT15 code.

N is the number of elements of the array to be transferred to the main file.

CNAME, when present, is the output argument used for the return of an edit address.

See section 2.11 for a more complete description of the ANY routine.

CHAPTER 4

CODE MODIFICATION ROUTINES

The code modification routines provide a limited means of modifying existing VT15 code, rather than completely recreating the whole display file. Instead of working on a specific display file, these routines use an edit address, CNAME, which is the output argument returned at the time of graphics code generation. CNAME provides the starting address of a group of VT15 commands.

The general mode of operation of the code modification routines is to replace an old group of commands with a new group. In deciding whether there is room for the new command group, the modification routines insert display no-ops immediately following the old command group. A brief description of the routines follows:

DELETE - replaces a group of commands with display no-ops.

RSETPT - accesses the SETPT code generating routine. The code is generated, if it fits, at the address provided in the CNAME argument.

REPLOT - accesses the code generating routines LINE, TEXT, COPY, PRAMTR, GRAPH, POINT and ANY. The code is placed at the CNAME address, if it fits.

A comprehensive programming example using the code modification routines is located at the end of Chapter 5.

4.1 GENERAL

Under XVM/RSX, the CNAME pointer has a new format. It used to be a 15-bit address pointer with a three-bit count field. It is now a 17-bit address pointer. The graphics system derives the count of in-core display items by examining the display code.

When REPLOT is used to place parameter instruction code (type 2) over nonparameter code, a problem arises. Marker no-ops are used to terminate parameter code, because the parameter groups have variable length. When a REPLOT occurs, marker no-ops must be inserted if parameter instructions precede or follow the new group. One marker no-op is required for each parameter-to-parameter interface. The REPLOT can fail if insufficient space is available for the marker no-ops.

When a group of commands is replaced by a group having the same number of commands, the new group is written over the old. If the new group is smaller than the old, the new group is placed so that it starts at the same location as the old. The extra locations at the end of the new group are filled with display no-ops. If the new group is larger than the old and there is a sufficient number of display no-ops following the old group, the new group is placed so that it starts at the same location as the old. If the new group is larger and there is not a sufficient number of display no-ops, the display file is not modified.

If the user has made REPLOT or RSETPT a FORTRAN function, he receives a logical FALSE indication if the edit has failed or a logical TRUE indication if it has succeeded. Otherwise, there is no indication. (MACRO programs receive an AC value of -1 if the edit has succeeded; zero if not.)

If the user wishes to replace two small command groups with a larger one, he should first DELETE the second command group. On editing the first group, the graphics system uses the display no-ops already in place for the second command group.

WARNING

The code modification routines require an I/O CAL to be issued to the VT15 handler. If there is an outstanding LTORPB, the CAL cannot be immediately honored. When the LTORPB is completed, the modification call is completed.

4.2 ~~DELETE~~ DDLETE (NAME CHANGED TO AVOID CONFLICT WITH FILE DELETE CALL

DELETE replaces the group of commands pointed to by CNAME with display no-ops. The calling sequence for the subroutine and function calls is:

```
CALL ~~DELETE~~ DDLETE (CNAME)
I=~~DELETE~~ DDLETE(CNAME)
```

CNAME is the edit address obtained when the group to be deleted was created. In the function call form, both I and DELETE must be declared as logical variables.

4.3 REPLOT

The REPLOT routine allows the code-generating capability of subpicture calls COPY, LINE, PRAMTR, TEXT, POINT, GRAPH and ANY to be used to edit graphics code over existing graphics code. The form of the REPLOT call is the same as that for the PLOT call described in Chapter 3. The first argument of REPLOT calls is a selection argument to describe which routine is to be selected:

Code	Routine
0	COPY
1	LINE

2	PRAMTR
3	TEXT
4	POINT
5	GRAPH
6	ANY

The selection argument is followed by those arguments required by the code-generating routines, as indicated in Chapter 3. The difference between PLOT and REPLOT is that there are no optional arguments for REPLOT calls, because the final argument, CNAME, must be provided in the REPLOT call.

A list of the calling sequences for both subroutine and function calls is given below for each of the code-generating subroutines. A logical FALSE indication is returned to the function if the group does not fit or if CNAME is zero.

```
CALL REPLOT (0,RST,PNAME1(1),CNAME)
I=REPLOT(0,RST,PNAME(1),CNAME)

CALL REPLOT (1,IX,IY,INT,CNAME)
I=REPLOT(1,IX,IY,INT,CNAME)

CALL REPLOT(2,SELECT,VALUE1,VALUE2,,CNAME)
I=REPLOT(2,SELECT,VALUE1,VALUE2,,CNAME)

CALL REPLOT (3,STR(1),N,CNAME)
I=REPLOT(3,STR(1),N,CNAME)

CALL REPLOT(4,IX,IY,INT,CNAME)
I=REPLOT(4,IX,IY,INT,CNAME)

CALL REPLOT(5,DATA(1),N,A,CNAME)
I=REPLOT(5,DATA(1),N,A,CNAME)

CALL REPLOT(6,ARRAY(1),N,CNAME)
I=REPLOT(6,ARRAY(1),N,CNAME)
```

4.4 RSETPT

The RSETPT routine uses the code-generating capability of the SETPT routine to edit over previously existing code. The operation is similar to the REPLOT call:

```
CALL RSETPT (IX,IY,CNAME)
I=RSETPT(IX,IY,CNAME)
```

CHAPTER 5

INPUT-OUTPUT

5.1 GENERAL

The RSX graphics system allows the FORTRAN user limited access to the RSX VT15 handler. The available calls and a brief description of each are listed below:

VTUNIT - provides a logical unit number (LUN) to the VT15 handler for the scope.

DINIT - requests that the provided display file be called by the VT15 as a subroutine from the handler loop. The file provided is established as the main file.

DCLOSE - requests that the present main file be detached from the VT15 execution loop (i.e., turned off).

CINIT - requests that the provided file be called as a subroutine from the VT15 execution loop. This file is not established as the main file. This file and all related calls are displayed on both scopes for the VT15 processor.

CCLOSE - requests that the last CINITed file be detached from the VT15 execution loop.

LTORPB - provides the user with the capability of obtaining status and interrupts from the light pen and push buttons on the scope.

GETPSH - reads the present push-button settings.

TRACK - allows the user to input X-Y coordinate data in a limited manner with the light pen.

The RSX VT15 handler provides for a maximum configuration of two VT15 processors, each with two scopes. Each of the four scopes can have independent operation of light pen and push buttons. Each of the four scopes can display a different picture, or image. With two scopes on one processor, the amount of material that can be displayed on the screen before "flicker" occurs is the sum of the displayed elements on both scopes. (The CINIT display of identical pictures on both scopes is considered a single execution.)

WARNING

If there is an outstanding LTORPB, the calls of this chapter (except for VTUNIT and LTORPB) cannot be immediately honored.

5.2 VTUNIT CALL

VTUNIT is used to notify the handler which logical scope number is to be used. Logical numbers 0 and 1 are on the first VT-15 processor, and 2 and 3 are on the second. If VTUNIT is never called (DOS does not have this call), logical unit number 0 is used. If used, VTUNIT calls should be made the first call to the graphics system. The calling sequence is:

```
CALL VTUNIT(N)
```

The unit number N must be provided as an argument.

The 'normal' LUN slots for the scopes are 24-27. The graphics system adds 24 to the provided number, and passes the result to the handler.

There is no 'assignment' implicit in this call. If, for example, one user requests unit 0 and starts up a picture on the scope, and another user now requests unit 0 and starts a picture, the second user's picture will be shown. No one will get an error message or any other indication. If protection against this type of situation is desired, the RSX ATTACH command can be used to lock out other jobs.

5.3 DINIT CALL

This is the traditional DOS call to start up the picture. It serves basically the same function under RSX. The calling sequence is:

```
CALL DINIT (MAIN(1))
```

Here MAIN is the array containing the file to be the 'main' file. Under RSX this file is called as a subroutine by the VT-15 processor from a loop in the VT-15 handler. This change is made necessary because the VT-15 processor is essentially a shared device. Calls to the PLOT routine will now place code in this file, that is, it is now the 'main' file. When the VT-15 processor enters the 'main' display file, the intensity setting is 4, SYNC is on, and all other parameter settings are 0. The beam position is undefined unless TRACKing is in force; in this case only, the beam position is the center of the light-pen TRACKing symbol (see Section 5.9).

5.4 DCLOSE CALL

The call to DCLOSE is used to stop the execution of the present main file by the VT15 processor. The handler removes the call to this routine from its loop. The resulting file can be used as a subpicture file or restarted as a main file. The assignment of the main file for PLOT calls is not changed by DCLOSE. The calling sequence is:

```
CALL DCLOSE
```

5.5 CINIT CALL

The CINIT routine is similar to the DINIT routine in that it starts up a display image by requesting that the handler call the provided file as a subroutine. (There is no effect on the main file assignment for the purposes of PLOT calls.) Here, however, the resulting image is shown on both scopes (if present). This call can be used, for example, for some feature common to both displays, such as light pen buttons, grids, etc. The execution time (VT15) of a CINIT file is half that required for each scope to separately display the same image. The VT15 processor enters the CINIT file with the same default parameter settings as for the main file. The position of the beam on entry to the file can be anything. A recommended procedure is to issue an absolute beam positioning call early in the file. The calling sequence is:

```
CALL CINIT (IFILE(1))
```

5.6 CCLOSE CALL

The CCLOSE routine is similar to the DCLOSE routine, except that it acts on files that have been CINITed. The calling sequence is:

```
CALL CCLOSE
```

5.7 LTORPB CALL

This routine allows the user to obtain information and interrupts from the light pen and push buttons. While the argument structure and information passed are similar to that under DOS, the use of this routine is different under the multiprogrammed RSX system. Under DOS, for example, it is possible to establish a tight loop with an LTORPB waiting for a push-button "hit" to occur. This is a poor procedure for a multiprogrammed system, but is quite satisfactory for a stand-alone system.

Another condition to avoid is a tight loop consisting of a light-pen hit, a program reenable of the hit element, another light pen hit, etc. See the example program in Figure 5-1 for a more complete explanation.

A call to LTORPB is necessary to activate the light pen. At all other times, the light pen is left inactive to reduce system loading. The calling sequence is:

```
I=LTORPB(IX,IY,NAMR,IBT,IWICHC,WAIT3)
```

where: IX is the absolute X (horizontal) coordinate of the end of the vector that caused the light pen interrupt. It is meaningless when there has not been a light pen interrupt.

IY is the absolute Y (vertical) coordinate of the end of the vector that caused the light pen interrupt. It is meaningless if there has not been a light pen interrupt.

NAMR is the setting of the name register at the time of the light pen interrupt. It is meaningless if there has not been a light pen interrupt.

IBT is the name of a logical six member array into which the on-off state of each push button is placed. The values are meaningless if there has not been an interrupt. Upon a light pen interrupt, the push buttons are read correctly.

IWICH will be 1 if a light pen hit has occurred, 2 if a push button hit has occurred, and 3 if both (just barely likely).

WAIT, when present and nonzero, indicates that the handler is to wait until an interrupt occurs before returning to the user.

LTORPB, IBT and I must be declared as logical variables in a TYPE statement when used in function calls. When I is true, an interrupt has occurred; when false, one has not.

If the WAIT argument is used, another calling sequence is possible:

```
CALL LTORPB (IX,IY,NAMR,IBT,IWRCH,1)
```

For this subroutine call, only IBT is required to be declared a logical variable in a type statement. In this case it is known that an interrupt has occurred when control returns to the user. The following example shows the use of LTORPB in an IF statement:

```
IF(LTORPB(LPX,LPY,NAME,IBT,ILB,0)) GO TO 100
```

If an interrupt occurred, go to statement 100. Note that the form of the WAIT argument gives an immediate return.

The LTORPB call issues an I/O CAL to the handler to enable light pen and push button hits. If WAIT was requested, control returns to the user at the time of the interrupt. The status, of course, is that at interrupt time. Interrupts that occurred prior to the initial LTORPB will have been ignored. Interrupts are not enabled when control returns to the user.

The situation is somewhat different when the LTORPB is specified with an immediate return. Initially, interrupts will not be enabled. The first LTORPB enables the interrupts, and returns to the user. This first LTORPB cannot have detected an interrupt. LTORPB's will be issued at some interval, each asking if an interrupt has occurred. After one of these, the interrupt will finally occur, status will be recorded at this time, and the interrupts will be disabled. The next LTORPB will notify the user that the interrupt has occurred, return the stored status, and leave the interrupts disabled.

It is possible for the user to issue an immediate return LTORPB, do some work, and then issue an LTORPB with a WAIT. The situation is then identical with that if the LTORPB with the WAIT had been initially issued.

5.8 GETPSH CALL

The GETPSH call immediately returns the present state of the push buttons. The calling sequence is:

```
CALL GETPSH (IBT)
```

where: IBT is a logical array of size six into which the push button settings are returned.

5.9 TRACK CALL

The TRACK routine has been reduced from a many optioned routine under DOS to a much simpler function under RSX. This is primarily due to a necessity to keep interrupt level time to a minimum. (Consider four scope users TRACK'ing at once). The program provides an initial position of a tracking symbol. The scope user moves the symbol to the desired position with the light pen. The scope user then hits a push button to signal the end of tracking. The program then receives the final X-Y position of the tracking symbol. The user program does not receive control for the duration of this procedure. The calling sequence is:

```
CALL TRACK (IX,IY[,IOPT,IARRAY])
```

where: IX and IY are both the original X-Y co-ordinates for the tracking symbol, and the variables to receive the final co-ordinates.

IOPT and IARRAY are arguments under DOS; they have no function here, but can be supplied without causing errors.

In contrast to DOS, no modification is made to the user 'main' file in tracking. The whole mechanism exists in the handler. The size of the tracking symbol is an assembly variable in the handler. It might prove convenient to re-assemble for a 21" scope, as the symbol was designed on a 17" scope. The following sample code shows the use of TRACK:

```
        IX=512
        IY=512
C  START IN CENTER OF SCREEN
        CALL TRACK (IX,IY)
        CALL POINT (IX,IY,0,IFF(1))
C  USE RETURNED VALUES FOR ABSOLUTE POSITION IN
C     THE DISPLAY FILE IFF
```

During TRACKing, the beam position upon entry to the user 'main' file is the center of the TRACKing symbol. If it is desired to have some graphics element automatically follow the TRACKing symbol, the first code in the 'main' file can be a COPY call to a subroutine containing the graphics element. Upon termination of TRACKing, the COPY call can

be DELETEd. The graphics element, along with position information returned from TRACK, can be permanently linked into the picture. At some future time, the top of the 'main' file can be REPLOTTed to link another following item during TRACKing.

5.10 COMPREHENSIVE EXAMPLE

The following example program (Figure 5-1) uses a considerable portion of the RSX capabilities of the graphics system. It shows some of the techniques to establish an interface between the system and the external user of the scope which minimizes system loading.

```
C
C       THIS IS A MORE GENERALIZED PROGRAM FOR THE
C  MANIPULATION OF ITEMS ON THE SCREEN. IN STATE 1 OF THE
C  PROGRAM, THE WORDS  'ADD'  'MOVE'  'KILL'  WILL
C  APPEAR ON THE TOP OF THE SCREEN. THE PROGRAM WILL ACT ON A
C  LIGHT PEN HIT ON ONE OF THESE WORDS. A PUSH BUTTON HIT ON
C  BUTTON NUMBER 6, THE RIGHTMOST, WILL CAUSE THE PROGRAM TO
C  EXIT FROM STATE 1.  WHEN THE PROGRAM GOES FROM STATE 1 TO
C  ONE OF THE ADD-MOVE-KILL ACTION ROUTINES, THESE WORDS
C  ARE REMOVED FROM THE SCREEN, TO HELP THE USER STAY
C  SYNCHRONIZED WITH THE PROGRAM, AND TO LIMIT UNWANTED
C  LIGHT PEN HITS.
C
C       THE 'ADD' INTERRUPT WILL CAUSE THE TRACKING SYMBOL TO
C  APPEAR. THE USER MOVES THE TRACKING SYMBOL TO THE DESIRED
C  POSITION, AND TERMINATES TRACKING BY HITTING A PUSH
C  BUTTON (NUMBER 5 IS RECOMMENDED SINCE IT IS NOT USED FOR
C  ANYTHING ELSE).  THE USER THEN HITS A BUTTON FROM 1-4 TO
C  SELECT THE DESIRED ITEM TO BE PLACED AT THE CO-ORDINATES
C  SPECIFIED BY TRACKING. THE ITEMS ARE, IN ORDER, SQUARE,
C  X, TRIANGLE, AND CIRCLE.  A HIT ON BUTTON 6 AT THIS POINT
C  WILL RETURN THE PROGRAM TO STATE 1 WITH NO ACTION TAKEN.
C  LIGHT PEN HITS WILL BE IGNORED. AFTER PLACEMENT OF THE
C  ITEM, THE PROGRAM RETURNS TO STATE 1 TO WAIT FURTHER
C  COMMANDS. A MAXIMUM OF 22 ITEMS CAN BE PLACED. THIS
C  SOMEWHAT ARBITRARY NUMBER IS FOR 11 O'S AND
C  11 X'S.
C
C       THE 'MOVE' INTERRUPT WILL ENABLE THE LIGHT PEN TO
C  RECEIVE A HIT ON THE ITEM THE USER WISHES TO MOVE. A HIT
C  ON PUSH BUTTON 6 RETURNS THE PROGRAM TO STATE 1.  OTHER
C  PUSH BUTTONS ARE IGNORED.  WHEN THE ITEM IS SELECTED, THE
C  TRACKING SYMBOL WILL APPEAR. MOVE THE TRACKING SYMBOL TO
C  THE DESIRED POSITION. A PUSH BUTTON HIT WILL TERMINATE
C  TRACKING, AND THE ITEM WILL BE MOVED TO THAT FINAL
C  POSITION. THE PROGRAM WILL THEN RETURN TO STATE 1.
C
C       THE 'KILL' INTERRUPT WILL ENABLE THE LIGHT PEN TO
C  RECEIVE A HIT ON THE ITEM THE USER WISHES TO REMOVE. A HIT
C  ON PUSH BUTTON 6 RETURNS THE PROGRAM TO STATE 1.  WHEN
C  A LIGHT PEN HIT OCCURS, THE ITEM IS REMOVED, AND
C  THE PROGRAM RETURNS TO STATE 1.
C
C       REMEMBER THROUGHOUT THAT A SEPARATE PUSH BUTTON
C  HIT IS NECESSARY TO TERMINATE TRACKING!
C
      LOGICAL IB(6)
      INTEGER CIRC(50),SQ(10),TRI(10)
      INTEGER X(10),DUM(10),CLEAR(2)
      DIMENSION IS(22),LOC(22),LINK(22),TA(2),TM(2),TK(2)
      DIMENSION MAIN(200),LB(30)
C
C  SET UP TEXT STRINGS FOR ADD,MOVE,KILL
      DATA TA(1)/5HADD  /
      DATA TM(1)/5HMOVE /
```

Figure 5-1
Comprehensive Example (Sheet 1 of 7)

```
      DATA TK(1)/5HKILL /
C
C  VT-15 INSTRUCTION SO BUTTONS ALWAYS OFF
C   THE # DESIGNATES OCTAL - VERY CONVENIENT!
      CLEAR(1)=#231374
C
C  NOW ZERO ALL FILE TOPS
      CIRC(1)=0
      SQ(1)=0
      TRI(1)=0
      X(1)=0
      DUM(1)=0
      MAIN(1)=0
      LB(1)=0
C
C  NOW ESTABLISH ELEMENT SUBROUTINES
C
C  THE CIRCLE
      CALL CIRCLE (25.,0.,360.,20.,CIRC)
C
C  THE SQUARE
      CALL LINE (25,-25,0,SQ(1))
      CALL LINE (0,50)
      CALL LINE (-50,0)
      CALL LINE (0,-50)
      CALL LINE (50,0)
C
C  THE TRIANGLE
      CALL LINE (-27,-16,0,TRI(1))
      CALL LINE (27,47)
      CALL LINE (27,-47)
      CALL LINE (-54,0)
C
C  THE X
      CALL LINE (25,-25,0,X(1))
      CALL LINE (-50,50)
      CALL LINE (50,0,0)
      CALL LINE (-50,-50)
C
C  AND THE FILE FOR LIGHT PEN BUTTONS
C
C  MAKE SCALE 1, AND TURN ON LIGHT PEN
C  IT WOULD PROBABLY RE BETTER TO PUT THIS
C  INTO THE OFFSET AREA.
      CALL PRAMTR (5,1,1,LB(1))
C
C  FIX THE BEAM NEAR SCREEN TOP
      CALL POINT (100,950)
C
C  SET NAME REGISTER TO 41 SO WE KNOW IT'S ADD
      CALL PRAMTR (128,41)
C
C  AND NOW THE 'ADD'
 512  CALL TEXT (TA(1),3)
C
C  SAME FOR MOVE AND KILL...
      CALL POINT (300,950)
      CALL PRAMTR (128,42)
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 2 of 7)

```
      CALL TEXT (TM(1),4)
      CALL POINT (500,950)
      CALL PRAMTR (128,43)
      CALL TEXT (TK(1),4)
C
C  TURN OFF LIGHT PEN, AND REPLACE SCALE 0
C  ALTERNATELY WOULD COULD USE SAVE-RESTORE
C  IN THE COPY CALL TO LB(1)
      CALL PRAMTR (5,0,0)
C
C  NOW THE DUMMY FILE WHICH IS CALLED WHEN THAT
C  SLOT HAS NO ELEMENT
      CALL LINE (10,0,0,DUM(1))
C
C  NOW SET UP 'MAIN' FILE
      CALL ANY (CLEAR(1),1,MAIN(1))
C
C  HORIZONTAL AXIS
      CALL POINT (0,512)
      CALL LINE (1023,0)
C
C  VERTICAL AXIS
      CALL POINT (512,0)
      CALL LINE (0,1023)
C
C  MAKE ITEMS BRIGHTER THAN AXES
      CALL PRAMTR (2,6)
C
C  CALL TO THE BUTTONS FILE
      CALL COPY (0,LB(1))
C
C  NOW LIGHT PEN ENABLE FOR ITEMS, RETURN ADDR SINCE
C  THIS WILL BE EDITTED ON AND OFF. ON IS BIGGER THAN OFF!
C  SO WE MAKE ON FIRST TO GET ENOUGH SPACE. TURNED OFF AGAIN
C  AT LINE TAGGED 30.
      CALL PRAMTR (4,1,MAIN(1),ITEMS)
C
C  REMEMBER TO TURN ON SCOPE
      CALL DINIT(MAIN(1))
C
C  NOW WE ARE GOING TO ESTABLISH LINKAGES FOR 22 ITEMS.
C  A PRAMTR TO SET NAME REG. FOR LIGHT PEN HITS, A SET POINT
C  TO POSITION THE ITEM, AND A COPY TO LINK IT TO FILE.
C  AT THIS TIME THERE ARE NO ITEMS, AND THE COPY WILL BE TO
C  A DUMMY ROUTINE.  THE REAL THINGS WILL BE EDITTED IN LATER.
C  THE EDIT ADDRESSES FOR POINT AND COPY WILL BE PLACED IN
C  THE ARRAYS LOC AND LINK. IS(N) IS AN ARRAY TO TELL WHEN THE
C  SLOT IS OCCUPIED BYA REAL ITEM; IT WILL BE ZEROED NOW.
C
      DO 13 I=1,22
C
C  SET UP NAME REG=SLOT NUMBER
      CALL PRAMTR (128,I)
C
C  ZERO OCCUPIED INDICATOR
      IS(I)=0
C
C  ABSOLUTE BEAM POSITION COMMAND
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 3 of 7)

```
        CALL POINT (1,1,0,MAIN(1),LOC(I))
C
C  AND COPY COMMAND
        CALL COPY (0,DUM(1),MAIN(1),LINK(I))
 13     CONTINUE
C
C  ENTER STATE 1, WAIT FOR HIT
C  BUT FIRST PREVENT LIGHT PEN HIT ON ITEMS!!!
C  THE NAME REGISTER WOULD PREVENT US FROM THINKING
C  THAT AN ITEM HIT WAS AN ADD-MOVE-KILL HIT, BUT
C  SOME AMOUNT OF SYSTEM COULD BE USED UP BY THE
C  LOOP: HIT ON ITEM, WE THROW IT AWAY, HIT ON ITEM....
C
 30     CALL REPLOT (2,4,0,ITEMS)
C
C  NOW SINCE WE ARE ENTERING STATE 1, PUT THE
C  ADD-MOVE-KILL BACK ON THE SCREEN.
        CALL UNBLNK(LB(1))
C
C  NOW WAIT UNTIL A HIT HAPPENS, THEN WE GET CONTROL
        CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  IF LAST PUSH BUTTON EXIT
        IF(IB(6)) GO TO 99
C
C  IF NOT A LIGHT PEN HIT, GO WAIT AGAIN
        IF(IW.GT.1) GO TO 30
C
C  IF WRONG NAME REGISTER, ALSO WAIT AGAIN
        IF(NAME.LT.41) GO TO 30
        IF(NAME.GT.43) GO TO 30
C
C  AHA, GOT A LEGAL ONE, SO TURN OFF ADD-MOVE-KILL
        CALL BLANK (LB(1))
C
C  NOW GO TO CORRECT ACTION ROUTINE
        IF(NAME.EQ.43) GO TO 43
        IF(NAME.EQ.42) GO TO 42
C
C
C   'ADD'
C
C  DEFAULT I=0 IF NO MORE EMPTY SLOTS
        I=0
C
        DO 41 IJ=1,22
C  FIND OUT IF SLOT EMPTY, AND SAVE
C  ITS NUMBER IF IT IS
        IF(IS(IJ).EQ.0)  I=IJ
 41     CONTINUE
C
C  IF NO EMPTIES, BACK TO STATE 1
 513    IF(I.EQ.0) GO TO 30
C
C  HAVE AN EMPTY, CONTINUE
        IS(I)=1
C  SET SWITCH, SAYING WE'RE TAKING THE SLOT
C
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 4 of 7)

```
C  SET UP TRACKING SO USER CAN SAY 'WHERE'
        IX=512
        IY=512
C  INITIALIZE IN SCREEN CENTER
C
        CALL TRACK (IX,IY)
C
C  NOW HAVE THE CO-ORDINATES, EDIT INTO PLACE
        CALL REPLOT (4,IX,IY,0,LOC(I))
C
C  NOW WAIT FOR USER PUSH BUTTON TO SAY 'WHICH'
 413    CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  IF BUTTON 6, BACK TO STATE 1, CLEARING IS(I)!!
        IF(IB(6)) GO TO 419
C
C  IF NOT PUSH BUTTON HIT, GO WAIT AGAIN
        IF(IW.LT.2) GO TO 413
C
C  IF 1-4 ALL OFF, GO WAIT AGAIN ALSO
        IF(IB(1)) GO TO 418
        IF(IB(2)) GO TO 418
        IF(IB(3)) GO TO 418
        IF(IB(4)) GO TO 418
        GO TO 413
C
C  NOW EDIT THE COPY TO CORRECT ITEM
 418    IF(IB(1)) CALL REPLOT (0,0,SQ(1),LINK(I))
        IF(IB(2)) CALL REPLOT (0,0,X(1),LINK(I))
        IF(IB(3)) CALL REPLOT (0,0,TRI(1),LINK(I))
        IF(IB(4)) CALL REPLOT (0,0,CIRC(1),LINK(I))
C
C  DONE, GO BACK TO STATE 1
        GO TO 30
C
C  REMEMBER RETURN TO STATE 1 ON BUTTON 6...
 419    IS(I)=0
        GO TO 30
C
C
C    'MOVE'
C
C  TURN ON LIGHT PEN FOR ITEMS
 42     CALL REPLOT (2,4,1,ITEMS)
C
C  WAIT FOR USER TO SELCT ITEM.
 423    CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  CHECK FOR BUTTON 6 PUSH TO STATE 1
        IF (IB(6)) GO TO 30
C
C  IF BUTTON HIT, WRONG, GO WAIT AGAIN
        IF(IW.GT.1) GO TO 423
C
C  IF WRONG NAME REG. GO WAIT AGAIN
        IF(NAME.EQ.0) GO TO 423
        IF(NAME.GT.22) GO TO 423
C
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 5 of 7)

```
C  GOT ONE, NOW SET UP TRACKING SO USER CAN SAY 'WHERE'
        IX=512
        IY=512
        CALL TRACK(IX,IY)
C
C  WHEN RETURN, HAVE THE X AND Y, SO EDIT IN
        CALL REPLOT (4,IX,IY,0,LOC(NAME))
C
C  DONE, GO TO STATE 1 TO WAIT FOR MORE COMMANDS
        GO TO 30
C
C
C    'KILL'
C
C  TURN OF LIGHT PEN FOR ITEMS
 43     CALL REPLOT (2,4,1,ITEMS)
C
C  WAIT FOR USER TO SELECT ITEM
 433    CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C  CHECK FOR PUSH BUTTON 6 RETURN TO STATE 1
        IF (IB(6)) GO TO 30
C
C  IF BUTTON HIT, WRONG, GO WAIT AGAIN
        IF(IW.GT.1) GO TO  433
C
C  IF WRONG NAME REG. GO WAIT AGAIN
        IF(NAME.EQ.0) GO TO 433
        IF (NAME.GT.22) GO TO 433
C
C  GOT THE ONE,  ZERO ITS FLAG
        IS(NAME)=0
C
C  AND REMOVE LINKAGE TO ITEM
        CALL DELETE (LINK(NAME))
C
C  AND HOP BACK TO STATE 1
 514    GO TO 30
C
C
C  HERE IS EXIT FROM STATE 1 VIA PUSH NUMBER 6.
 99     STOP
C
C       NOTE THAT THIS PROGRAM HAS A BUG IN IT! WHEN 22
C  ITEMS HAVE BEEN PLACED ON THE SCREEN, AND THE USER
C  ATTEMPTS TO PLACE THE IMPOSSIBLE 23RD, A LIGHT PEN
C  SELECT LOOP OCCURS. I.E. 'ADD' GIVES A LIGHT
C  PEN INTERRUPT, AND THE PROGRAM IMMEDIATELY RETURNS
C  TO STATE 1 - - TO GET THE SAME INTERRUPT AGAIN.
C  TRY IT AND NOTE THE SYSTEM LOADING.
C
C       ONE OF THE MANY WAYS TO FIX THIS
C  WOULD BE TO TURN OFF 'ADD' WHEN FULL UP.
C  THE INITIAL CALL AT 512 WOULD RETURN CNAME
C
C512    CALL TEXT (TA(1),3,LB(1),ITFLL)
C
C  AND AT 513, RATHER THAN GOING TO 30 ON FULL,
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 6 of 7)

```
C  GO VIA A
C
C       CALL DELETE (ITFLL)
C
C  AND FINALLY, A LINE OF CODE WOULD BE
C  INSERTED PRIOR TO 514 TO RESTORE THE ADD
C  UPON A SUCCESSFUL 'KILL'.
C
C       CALL REPLOT (3,TA(1),3,ITFLL)
C
C
C
        END
```

Figure 5-1 (Cont.)
Comprehensive Example (Sheet 7 of 7)

CHAPTER 6

RELOCATING ROUTINES

The routine DYSET is used to convert display files from their absolute executable form to a relocated form. The display files in relocated form are then stored on a mass storage medium. In the general case, these relocated display files generated by one user can then be placed in different arrays for other graphics programs generated by a second user. The routine DYLINK is then called to convert the display files back to the absolute executable form.

The DYSET-DYLINK routines under RSX are completely different than those under DOS. Files stored (DYSET) under one system cannot be brought back (DYLINK) under the other.

The basic intent of these routines is to allow storage of scope images. A scope image, or picture, may arise as a result of considerable work on the part of the scope user interacting with a program. Simply rerunning the program will not give the same picture, so it is necessary to store the actual display files. With careful programming it is also possible to bring together portions of stored images into a composite image.

The mass storage files may not be modified between the action of DYSET and DYLINK. Display code normally generated by the graphics system will survive the DYSET-DYLINK procedure. This includes BLANK'ed files.

6.1 DYSET ROUTINE

The DYSET routine converts all direct memory references to a relative form, where the address is relative to the first location of the display file. Indirect memory references must refer to display files provided as arguments to DYSET. These references are replaced by a logical number pointing to the requisite file name. Executable files have a positive number corresponding to their position in the argument list, text files a similar negative number. Each executable file is 'labelled' by placing its logical number in the second location of the file. The text files cannot be so conveniently labeled; the call to DYLINK must preserve the logical numbers by providing the text file arguments in the same order. The logical file number approach means that, in contrast to DOS, no additional display file space is required to DYSET files.

The possible calling sequences are;

```
CALL DYSET (PNAME1(1),PNAME2(1),,PNAMEN(1))

CALL DYSET (PNAME1(1),,,,PNAMEN(1),-1,STR1(1),,,,STRN(1))
```

where: PNAME1(1) to PNAMEN(1) are display files containing executable code.

STR1(1) to STRN(1) are arrays containing text strings.

-1 is a delimiter separating the executable files from the text strings.

All file arguments are of the usual form FILE(1). It is the user's responsibility to insure that all files, executable and text, referenced from the provided executable files, are included in the argument list. The VT-15 processor must not be allowed to execute a file in relocated form. The integrity of the entire system is at stake. DCLOSE and CCLOSE should be called before any call to DYSET.

6.2 DYLINK ROUTINE

The DYLINK routine reverses the action of the DYSET routine. The calling sequences are:

```
CALL DYLINK (PNAME1(1),PNAME2(1),,PNAMEN(1))

CALL DYLINK (PNAME1(1),,,,PNAMEN(1),-1,STR1(1),,,,STRN(1))
```

where: PNAME1(1) to PNAMEN(1) are the files containing executable code that are to be linked.

STR1(1) to STRN(1) are the names of the files containing text information that are to be linked.

-1 is a delimiter separating the executable and text files.

All files specified as arguments to a call to DYLINK must have been provided as arguments to a single call to DYSET. Otherwise there might be conflicts in the assignment of logical file numbers. The list of text files must appear exactly in the order that it appeared in the corresponding call to DYSET, even if some files are not referenced by the executable files. This is necessary to maintain the logical numbers of the text files; note that the text files will likely have different array names under DYLINK than they had under DYSET.

It is not necessary that all the PNAME's (executable files) provided in the call to DYSET be provided in the DYLINK call, or that they appear in the same order. That is, the user can 'link' a subset of what he 'set'. It is still necessary that all files which are referenced by the executable code appear in the argument list to DYLINK. Again the VT-15 processor cannot execute relocated files. It is recommended that the files be brought from mass storage, DYLINK'ed, and only then should the VT-15 be turned on.

If the programmer wishes, for example, to display portions of three separate stored pictures, he must first bring all the relevant files

into core. (Text files that are not referenced do not have to be core-resident, but they must appear in the argument string.) Then DYLINK is called three separate times, once for each group of files that was DYSET together. At this point the files are equivalent to ordinary display files. The COPY routine can be called so that all files are displayed. Note, however, that no mechanism exists for passing edit addresses (CNAME's) from one program to the next. In general the DYLINK'ed files cannot be edited (REPLOT'ed).

6.3 NON-STANDARD DISPLAY FILES

Certain restrictions should be noted if code not generated by the graphics system is to be passed through the DYSET-DYLINK procedure. It is not guaranteed that following these restrictions will ensure success. Try examples of the expected type of non-standard code when the system is not running important tasks.

The DYSET-DYLINK routines simulate the path of the VT-15 processor through the display file. This means that non-executable information, suitably protected with a display jump around it, can be placed in the display file. Similarly, a character string direct instruction could refer to a text string contained in the same display file. Again the text string would require a display jump around it.

Two separate indicators are used to decide what is the end of the display file. Whichever comes first will be honored. One indicator is the pointer in the first location of the display file. It is assumed to hold the difference in location between the last executable location of the display file, and itself. The second indicator is the occurrence of a display jump indirect. (Note, however, that the display jump indirect will not end the file if it is in the third location of the file, where it is placed in a BLANK'ed file).

As a consequence of the end of file assumption, it is not possible to have embedded subroutines in display files. The DYSET-DYLINK routines would stop at the end of the first embedded subroutine, and not complete the file. It is possible, however, to place non-executable information at the end of the display file, making sure, of course, that this information is stored on the mass storage device. The DYSET-DYLINK routines would stop upon finding the jump indirect at the end of the file, before acting on the non-executable information.

6.4 PROGRAM EXAMPLE

The program in Figure 6-1 illustrates the use of the DYSET-DYLINK routines.

```
C
C  EXAMPLE PROGRAM FOR DYSET-DYLINK
C
      LOGICAL IB(6)
      DIMENSION LAT0(40),LAT1(20),LAT2(20)
      DIMENSION NEW0(40),NEW1(20),NEW2(20)
      DIMENSION TXT01(2),TXT02(2),TXTN1(2),TXTN2(2)
C
      DATA TXT01(1),TXT01(2)/5HI AM ,4HBOXA/
      DATA TXT02(1),TXT02(2)/5HI AM ,4HBOXB/
C
C  INIT THE DISPLAY FILES
C
      LAT0(1)=0
      LAT1(1)=0
      LAT2(1)=0
C
C  BOXB
C
      CALL TEXT (TXT02(1),9,LAT2(1))
      CALL LINE (100,0)
      CALL LINE (0,100)
      CALL LINE (-100,0)
      CALL LINE (0,-100)
C
C  BOXA
C
      CALL LINE (300,0,1,LAT1(1))
      CALL LINE (0,300)
      CALL LINE (-300,0)
      CALL LINE (0,-300)
      CALL COPY (0,LAT2(1))
C
C  A 'MAIN' FILE
C
      CALL DINIT(LAT0(1))
      CALL PRAMTR (7,0,6,1,LAT0(1))
      CALL POINT (20,20)
      CALL TEXT (TXT01(1),9)
      CALL COPY (1,LAT1(1))
      CALL POINT (534,20)
      CALL TEXT (TXT01(1),9)
      CALL COPY (0,LAT1(1))
C
C  NOW LEAVE PICTURE ON UNTIL LIGHT PEN HIT
C
 20   CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
      IF (IW.GT.1) GO TO 20
C
C  CLOSE OUT AND DYSET
      CALL DCLOSE
      CALL DYSET (LAT0(1),LAT1(1),LAT2(1),-1,TXT01(1),TXT02(1))
C
C  NOW OUTPUT THE 5 FILES TO LUN 5
      CALL ENTER (5,1HF,1H1,IEV)
      CALL WAITFR (IEV)
      J=LAT0(1)+1
```

Figure 6-1
DYSET-DYLINK Example (Sheet 1 of 3)

```
      WRITE (5) (LAT0(I), I=1,J)
      CALL CLOSE (5,1HF,1H1,IEV)
      CALL WAITFR (IEV)
C
      CALL ENTER (5,1HF,1H2,IEV)
      CALL WAITFR (IEV)
      J=LAT1(1)+1
      WRITE (5) (LAT1(I), I=1,J)
      CALL CLOSE (5,1HF,1H2,IEV)
      CALL WAITFR (IEV)
C
      CALL ENTER (5,1HF,1H3,IEV)
      CALL WAITFR (IEV)
      J=LAT2(1)+1
      WRITE (5) (LAT2(I), I=1,J)
      CALL CLOSE (5,1HF,1H3,IEV)
      CALL WAITFR (IEV)
C
      CALL ENTER (5,1HF,1H4,IEV)
      CALL WAITFR (IEV)
      WRITE (5) (TXT01(I), I=1,2)
      CALL CLOSE (5,1HF,1H4,IEV)
      CALL WAITFR (IEV)
C
      CALL ENTER (5,1HF,1H5,IEV)
      CALL WAITFR (IEV)
      WRITE (5) (TXT02(I), I=1,2)
      CALL CLOSE (5,1HF,1H5,IEV)
      CALL WAITFR (IEV)
C
C  NOW GET THEM ALL INTO NEW FILES
C
      CALL SEEK (5,1HF,1H1,IEV)
      CALL WAITFR (IEV)
      READ (5) J,(NEW0(I+1), I=1,J)
      NEW0(1)=J
      CALL CLOSE (5,1HF,1H1,IEV)
      CALL WAITFR (IEV)
C
      CALL SEEK (5,1HF,1H2,IEV)
      CALL WAITFR (IEV)
      READ (5) J,(NEW1(I+1), I=1,J)
      NEW1(1)=J
      CALL CLOSE (5,1HF,1H2,IEV)
      CALL WAITFR (IEV)
C
      CALL SEEK (5,1HF,1H3,IEV)
      CALL WAITFR (IEV)
      READ (5) J,(NEW2(I+1), I=1,J)
      NEW2(1)=J
      CALL CLOSE (5,1HF,1H3,IEV)
      CALL WAITFR (IEV)
C
      CALL SEEK (5,1HF,1H4,IEV)
      CALL WAITFR (IEV)
      READ (5) (TXTN1(I), I=1,2)
      CALL CLOSE (5,1HF,1H4,IEV)
      CALL WAITFR (IEV)
```

Figure 6-1 (Cont.)
DYSET-DYLINK Example (Sheet 2 of 3)

```
C
      CALL SEEK (5,1HF,1H5,IEV)
      CALL WAITFR (IEV)
      READ (5) (TXTN2(I), I=1,2)
      CALL CLOSE (5,1HF,1H5,IEV)
      CALL WAITFR (IEV)
C
C  WE GOT THEM BACK, SO DYLINK
      CALL DYLINK (NEW1(1),NEW0(1),NEW2(1),-1,TXTN1(1),TXTN2(1))
C
C  AND TURN ON THE TUBE
      CALL DINIT (NEW0(1))
C
C  WAIT FOR LIGHT PEN HIT TO EXIT
 30   CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
      IF(IW.GT.1) GO TO 30
C
C  EXIT
      STOP
      END
```

Figure 6-1 (Cont.)
DYSET-DYLINK Example (Sheet 3 of 3)

CHAPTER 7

VT-15 HANDLER

This chapter provides a brief description of the external interface of the RSX VT-15 handler. The handler is designed for the VT-15 scope and will not handle I/O for other devices. The user calls to the handler are made via GET's and PUT's. In general, PUT's are used to issue VT-15 IOT's, and GET's are used to obtain status and 'interrupts' from the light pen, push buttons, etc.

The handler occupies 2000 octal locations for a one scope configuration, and 2400 octal locations for two scopes or more. In addition to the GET and PUT previously mentioned, the I/O handler honors DETACH, ATTACH, HINF, ABORT, and DISCONNECT & EXIT. The HINF function returns a value of 13 octal.

Paragraph 7.3 of this chapter also deals with the use of both the handler and the FORTRAN package from a MACRO program and includes an example program.

7.1 DESCRIPTION OF PUT

The format of the PUT CAL:

```
        CAL     PUTCAL  /THE CAL IS ISSUED TO THE CAL PARAMETER
/                       /BLOCK AT ADDRESS PUTCAL
/
/  SOMEWHERE IN THE CODE IS THE CONTROL BLOCK
PUTCAL  3100            /CODE FOR THE PUT FUNCTION
        EV              /ADDRESS OF THE EVENT VARIABLE
        UNIT            /LOGICAL UNIT NUMBER, USUALLY 24-27
        PUTTAB          /ADDRESS OF CONTROL TABLE
/
/  AND SOMEWHERE ELSE IN THE CODE, THE CONTROL TABLE
PUTTAB  TYPE            /CODE FOR TYPE OF PUT
        ARG             /ADDITIONAL ARGUMENT, OPTIONAL,
/                       /  DEPENDING ON WHAT KIND
/                       /  OF PUT IS BEING ISSUED
```

The following is a list of the legal types of PUT functions and their action:

TYPE 0 - requests that the VT-15 handler stop the VT-15 processor associated with the scope specified by the provided LUN. The stop is

accomplished by the handler issuing a 703044 or 703444. Use this function with care if there are two scopes on the VT-15 processor, since it will turn off both scopes.

TYPE 1 - requests that the VT-15 handler resume operation of the VT-15 processor associated with the scope specified by the provided LUN. When the VT-15 processor is stopped, it remembers its PC, and can be continued (resumed) from that location. The handler issues a 703024(703424) to set the initial conditions, and then a 703064(703464) to resume execution.

TYPE 2 - requests that the handler place a subroutine call to the address provided in the optional argument of the argument table in the handler VT-15 display loop. The VT-15 processor is then started in that display loop, and will call the provided address as a subroutine. (This CAL is issued by the DINIT routine.) The image shown by the called code will be displayed on the scope specified by the provided logical unit number (LUN). Note that while the user provided IFILE(1) to the graphics system, it then passes the address of IFILE(2) to the handler.

TYPE 3 - requests that the handler place a subroutine call to the provided address in the display processor loop. Again the VT-15 is started at the top of the display loop (even if already running). The image will be shown on the scope specified by the provided LUN, and the other scope (if present) on the same VT-15 processor. (This CAL is issued by CINIT.)

TYPE 4 - requests that the subroutine linkage established by the last TYPE 2 PUT be removed. This request is honored only if that PUT was issued from the partition of the present CAL. If this is the last subroutine in the display loop, the VT-15 processor is stopped. (This call is issued by DCLOSE.)

TYPE 5 - requests that the subroutine linkage established by the last TYPE 3 PUT be removed. This request is honored only if the linkage to be removed is to the partition of the present CAL. If this is the last subroutine in the display loop, the VT-15 processor is stopped. (This PUT is issued by CCLOSE.)

TYPE 6 - requests that the argument provided as the optional argument be used as the SIC (set initial conditions) word the next time this VT-15 processor is started or resumed. Prior to starting the display, the IOT 703024 or 703424 is issued to establish various initial conditions. See the GRAPHIC-15 Reference Manual for a description of the IOT instruction bits.

TYPE 7 - requests that the VT-15 processor be started at the address supplied as the optional argument. This allows the user to circumvent the display loop in the VT-15 handler.

WARNING

Very short display files at high intensities executed without SYNC can damage the scope phosphor.

Now that the handler is no longer in the display execution loop, the user now has the responsibility of either running in SYNC (place a

236000 (octal) in the display loop) or maintaining the display file at an appropriate size and intensity. (This CAL nullifies the effect of all type 2 and type 3 PUTs previously issued to this VT15 processor.) The user also has the responsibility of shutting down the scope software when he is done.

7.2 DESCRIPTION OF GET

The format of the GET CAL to the RSX system is as follows:

```
        CAL     GETCAL  /ISSUE A CAL TO THE CAL PARAMETER
/                       /BLOCK AT ADDRESS GETCAL
/
/  SOMEWHERE ELSE IN THE CODE, CAL PARAMETER BLOCK
GETCAL  3000            /FUNCTION CODE FOR A GET
        EV              /ADDRESS OF EVENT VARIABLE
        UNIT            /UNIT NUMBER OF SCOPE
        GETTAB          /ADDRESS OF CONTROL TABLE
/
/  AND SOMEWHERE ELSE IN THE CODE, CONTROL TABLE
GETTAB  CODE            /CODE FOR THE TYPE FO GET
        GBUF            /ADDRESS OF A BUFFER FOR ARGUMENTS
        ARG             /OPTIONAL ARGUMENT
/
/  SOMEWHERE ELSE
GBUF    0               /BUFFER OF 2-10 LOCATIONS FOR
/                       /ARGUMENTS AND DATA
/                       /BACK AND FORTH. DEPENDS ON TYPE
```

The following is a list of the legal types of GET functions and a brief description of their action:

Type 7 - This type is discribed first, because types 0 to 6 are subsets of type 7. The third word of the control table (GETTAB+2) describes which interrupts are to be acted on and what status is to be returned in the argument buffer on an interrupt. The first word of the argument buffer is returned with this descriptor word with only those bits set for which an interrupt actually occurred. This means that an information receiving routine, separate from the requesting routine, can determine which interrupt occurred and which status words are being returned. The returned status words are then placed in order of increasing bit number in the argument buffer after the leading descriptor word. If a staus word is omitted in the bit specifications, no open word is left in the argument buffer. The arguments are packed from the top down.

Bits 0 to 4 of the word describe which interrupts are to be acted on:

Bit 0 is the internal stop interrupt.

Bit 1 is the push-button interrupt.

Bit 2 is the light-pen interrupt.

Bit 3 is the edge-flag interrupt.

Bit 4 is the external stop interrupt.

WARNING

The code modification routines request external stops from the VT-15 handler. See the GRAPHIC-15 Reference Manual for a more complete description of these interrupts.

Bits 5-11 (decimal) describe what status words are to be returned in the argument buffer:

Bit 5 returns a word containing (left-justified) a bit for the state of each push button.
Bit 6 returns the X-position of the beam.
Bit 7 returns the Y-position of the beam.
Bit 8 returns the display program counter.
Bit 9 returns READ-STATUS 1.
Bit 10 returns READ-STATUS 2.
Bit 11 returns the NAME-REGISTER.

See GRAPHIC-15 Reference Manual for a description of READ-STATUS 1 and 2.

TYPE 0 - reads all of the status registers immediately without waiting for an interrupt.

TYPE 1 - waits for an internal stop interrupt, and returns all status registers when one happens.

TYPE 2 - waits for a push button interrupt, and returns all registers when that happens.

TYPE 3 - waits for a light pen interrupt and returns all status registers.

TYPE 4 - waits for an edge interrupt and returns all registers.

TYPE 5 - waits for an external stop interrupt and returns all registers.

TYPE 6 - waits for light pen or push buttons. It returns push buttons, X-position, Y-position, and NAME-REGISTER. (This is used by the LTORPB routine.)

TYPE 10 - (octal) performs TRACK'ing. The initial positive integer X and Y positions are placed in the first two words of the argument buffer. When the scope user signals an end of TRACK'ing by generating a push button interrupt, the final X and Y positions are returned into the first two words of the argument buffer.

TYPE 11 - is not implemented at present.

TYPE 12 - returns into the first word of the buffer the address of the subroutine in the handler (for the provided scope unit) to enable the light pen. Normally, the light pen is activated by this routine when the user has issued a GET that requests a light pen interrupt. In the user display code, instead of the hardware instruction to enable the light pen, is a DJMS* to this subroutine. In this way, the light pen is enabled only in those portions of the display file that the user wishes, and only when a GET on the light pen (usually an LTORPB in the FORTRAN environment) has been issued.

7.3 MACRO PROGRAMMING

If the MACRO user wishes to issue calls to the FORTRAN callable routines, he must simulate the calling sequence. If, for instance, the FORTRAN call is:

```
CALL LINE (IX,IY,INT,MAIN(1))
```

Then the corresponding MACRO call is:

```
.GLOBL LINE
JMS*    LINE
JMP     .+5
.DSA    IX
.DSA    IY
.DSA    INT
MAIN
```

If a variable is to be floating point instead of integer, the calling sequence is the same, but the variable is two core locations in floating point format. If MAIN is to be provided instead of MAIN(1), the argument would remain MAIN. With the original VPR.32 MAIN would require .DSA MAIN as an argument.

A user may also write his own graphics code, run under protect-relocate, and call the handler for I/O functions. In this case, he assumes the responsibility of relocating those addresses in graphics code that reference memory. This is best explained by the example in Figure 7-1. If the user runs in executive mode, ignoring both the graphics package and the handler, the coding techniques are exactly those described in the GRAPHIC-15 Reference Manual. This last technique is not recommended, since I/O rundown cannot be done in a reasonable manner.

The following program shows the use of the handler and code relocation under protect-relocate:

```
/
/  MACRO EXAMPLE CORRESPONDING ROUGHLY
/  TO THE FOUR SQUARE FORTRAN EXAMPLE
/
/  GRAPHICS EQUALITIES
DJMS=640000
DJMPI=620000
CHARS=40000
/
/  SQUARE SUBROUTINE
/
SQ      0
        420144          /+X=100 DECIMAL
        424144          /+Y=100 DECIMAL
        430144          /-X
        434144          /-Y
CSQR    SQ              /MUST RELOCATE VT-15 MEMORY
/                       /REFERENCES BECAUSE WE'RE RELOCATED
/                       /AND IT'S NOT
/
/  MAIN FILE
/
MAIN    0
        140144          /SET Y=100
        144144          /SET X=100
        200021          /SET SCALE=1
CTXT    STR             /RELOCATE TEXT STRING REFERENCE
        200020          /SCALE=0 FOR SQUARES
        140764          /SET Y=500
        144310          /SET X=200
CCSQ    SQ              /CALL TO SQUARE MUST BE RELOC'ED
        141440          /SET Y=800
        144310          /SET X=200
CCSQ1   SQ              /AGAIN RELOC IT
        140764          /Y=500
        145274          /X=700
CCSQ2   SQ              /AS BEFORE
        141440          /Y=800
        145274          /X=700
CCSQ3   SQ              /FINAL CALL TO SQ SUBROUTINE
CMR     MAIN            /RETURN JUMP FROM MAIN
/
/  CAL'S
/
/  FIRST THE ONE TO SYSTEM TO FIND
/  OUT WHAT OUR PARTITION BEGINNING
/  ADDR IS
/
WHRCAL  26
        WHREV
        0
        0
        WHRBEG
WHRBEG  -1
        -1
/
/
```

Figure 7-1
MACRO Programming Example (Sheet 1 of 3)

```
/  PUT CAL
/
PUTCAL  3100
        PEV
        30
        PUTTAB
PUTTAB  2
        MAIN
/
/  GET CAL
/
GETCAL  3000
        GEV
        30
        GETTAB
GETTAB  2
        GBUF
GBUF    0
        0
        0
        0
        0
        0
        0
        0
/
/  WAITS FOR ABOVE CAL'S
/
WATWHR  20
        WHREV
/
/
WATPUT  20
        PEV
/
/
WATGET  20
        GEV
/
/  EVENT VARIABLES
/
WHREV   0
PEV     0
GEV     0
/
/  TEXT STRING
STR     .ASCII /HERE ARE 4 SQUARES/
/
/  FIRST WE HAVE TO RELOCATE ALL VT-15
/   MEMORY REFERENCES
/
/  FIND OUT WHERE WE ARE
/
START   CAL     WHRCAL  /RETURNS PARTITION ADDR.
        CAL     WATWHR  /WAIT FOR COMPLETION
        LAC     WHREV   /IF POSITIVE OK
        SPA
        HLT             /HALT ON ERROR
```

Figure 7-1 (Cont.)
MACRO Programming Example (Sheet 2 of 3)

```
/
/  RELOCATE RETURN JUMP FROM SQUARE SUBROUTINE
/
        LAC     WHRBEG  /START OF PARTITION
        TAD     CSQR    /ADD IN RELATIVE ADDR.
        AND     (17777  /TRUNCATE TO 13 BITS
        TAD     (DJMPI  /OP CODE JUMP INDIRECT
        DAC     CSQR    /AND INTO PLACE
/
/  AND SAME FOR TEXT STRING REFERENCE
/
        LAC     WHRBEG  /ADDR. OF PARTITION START
        TAD     CTXT    /ADD RELATIVE TO START
        AND     (17777  /VT-15 HAS 13 BIT ADDR
        TAD     (CHARS  /OP CODE FOR CHARACTER STRING
        DAC     CTXT    /PLACE IT
/
/  AND FOR FOUR CALLS TO SQUARE
/
        LAC     WHRBEG
        TAD     CCSQ
        AND     (17777
        TAD     (DJMS   /SUBROUTINE CALL OP CODE
        DAC     CCSQ    /4 TIMES FOR 4 COPIES
        DAC     CCSQ1
        DAC     CCSQ2
        DAC     CCSQ3
/
/  AND RETURN FROM MAIN TO HANDLER
/
        LAC     WHRBEG
        TAD     CMR
        AND     (17777
        TAD     (DJMPI
        DAC     CMR
/
/  PUT ALT-MODE IN TEXT STRING
/
        LAC     STR+7   /END OF 4 TWO-WORD PAIRS
        AAC     372     /ALT-MODE SHIFTED UP ONE
        DAC     STR+7
/
/  AND THE PUT TO CALL OUR 'MAIN' FILE
/
        CAL     PUTCAL  /
        CAL     WATPUT  /WAIT FOR COMPLETION
        LAC     PEV     /EVENT VARIABLE
        SPA
        HLT
/
/  NOW LEAVE PICTURE RUNNING, WHILE
/  WAITING FOR A PUSH BUTTON
/
        CAL     GETCAL
        CAL     WATGET  /WAIT FOR COMPLETION
        LAC     GEV
        SPA
        HLT
        CAL     (10     /EXIT, WHICH WILL CAUSE HANDLER
/
/  TO TURN OFF TUBE IN OUR PARTITION
        .END    START
```

Figure 7-1 (Cont.)
MACRO Programming Example (Sheet 3 of 3)

CHAPTER 8

WRITING TABLET HANDLER

The writing tablet returns X-Y coordinate pairs to the user program. The pen emits a spark; the time that it takes the sound of the spark to reach microphones at the side of the tablet gives the X-Y coordinates of the pen. When the pen is pressed firmly on the tablet surface, it will emit a spark (assuming that it has been enabled). It is also possible to initialize the pen to issue a continuous series of sparks.

The modes of operation of the writing tablet are most easily described when there is only one tablet. In mode zero, the continuous series of sparks is not initialized, only single point data from pressing the pen upon the tablet (pen data) is returned to the program. In mode one (any nonzero mode setting), the pen is initialized to spark continuously to provide continuous tracking. Both the single point data and the continuous tracking data are returned (appropriately labelled) to the user program.

When two users must be simultaneously serviced (a maximum of four is possible), operation is more complicated. A mode zero user may find that his pen starts to emit a continuous spark (necessary to service some mode one user); only the pen data will be returned to the mode zero program. A mode one user may find that his data rate is halved when another user of any type is being serviced.

8.1 GETBLT ROUTINE

The GETBLT routine is called by the FORTRAN user to obtain an X-Y pair from the writing tablet. The calling sequence is:

```
CALL GETBLT (LUN,MODE,ARRAY[,IEV])
```

where: LUN is an integer expression describing which logical unit is to be referenced.

MODE is an integer expression, defining the mode of operation, as described above.

ARRAY is a three member integer array into which are returned the data and a type indicator.

IEV is an integer event variable.

Pen data is indicated by a zero word in the first location of ARRAY. A value greater than zero indicates continuous data obtained at full speed; a value less than zero, half speed. The second location contains the X coordinate, and the third location contains the Y coordinate. The coordinate data range in value from 0 to 1023 decimal.

IEV is handled in a manner identical to RSX calls such as ENTER, SEEK etc. In the general case IEV must be provided, and the calling program must issue a WAITFR to insure that the read has occurred before the data is used. If the program has controlled the timing in some other way, the IEV does not have to be provided as an argument to GETBLT.

8.2 HANDLER INTERFACE

The writing tablet handler honors a GET CAL to obtain the writing tablet data. The handler is designed for the writing tablet, and will not handle I/O intended for other devices. In addition to the GET CAL, the handler honors DETACH, ATTACH, HINF, ABORT, DISCONNECT & EXIT and CLOSE. The HINF function returns a value of 23 octal. The handler requires a partition of 1000 octal locations in the first 32K of core. The format of the GET CAL:

```
TABGET  3000            /SYSTEM CODE FOR GET
        EV              /EVENT VARIABLE ADDRESS
        LUN             /LOGICAL UNIT NUMBER
        CNTL            /CONTROL TABLE ADDRESS
/
/  AND SOMEWHERE IN CORE THE CONTROL TABLE
/
CNTL    MODE            /MODE, ZERO, OR NON-ZERO
        ARRAY           /ARRAY INTO WHICH TO GIVE DATA
/
/  AND THE ARRAY, FOR FORTRAN IT IS THE PROVIDED ARRAY
/  IN THE FORTRAN PROGRAM.  FOR MACRO IN CORE
/  SOMEWHERE
/
ARRAY   0               /DATA TYPE
RETX    0               /RETURN X VALUE
RETY    0               /RETURN Y VALUE
```

Figures 8-1 and 8-2 illustrate the use of the writing tablet GETBLT routine.

```
C
C  TEST PROGRAM TO SHOW TRACKING WITH THE WRITING TABLET
C
C       NOTE THAT A FAINT 'D' WILL APPEAR TO THE RIGHT OF THE 'P'
C WHEN THE PROGRAM IS RUNNING IN CONTINUOUS DATA MODE.
C WHILE THE REPLOT IS TAKING PLACE, THE GRAPHICS SYSTEM PLACES
C A JUMP OVER THE CODE TO BE MODIFIED. IF THE VT-15 PROCESSOR
C HAPPENS TO EXECUTE THAT JUMP, THE 'D' WILL NOT HAVE A SET-POINT.
C THE BEAM REMAINS POSITIONED AFTER THE 'P', SO THAT IS WHERE
C THE 'D' WILL BE PLACED.
C
C       THE PROGRAM CAN DEAL WITH THIS BY MOVING THE BEAM
C OFF-SCREEN PRIOR TO THE EDITED SET POINT. THE 'EXTRA' 'D'
C WILL THEN NOT BE SEEN. A CONSIDERABLE IMPROVEMENT CAN BE OBTAINED
C BY ASSEMBLING VPR.XX WITH THE QEDIT ASSEMBLY PARAMETER.
C WHEN THE SCOPE IS NOT RESTARTED FOR EACH EDIT, THE JUMP IS
C EXECUTED FAR LESS FREQUENTLY.
C
       LOGICAL IB(6)
       DIMENSION MAIN(100),ITAB(3)
C
C  ARBITRARY LUN SLOT FOR TABLET
       LUN=28
C
C  START UP MAIN FILE
       MAIN(1)=0
       CALL DINIT (MAIN(1))
C
C  SET UP A 'P' ON THE SCREEN TO FOLLOW PEN DATA
C  HITS FROM THE WRITING TABLET
C
C  FIRST THE BEAM POSITION; ALLOW FOR EDITING
       CALL POINT(250,250,0,MAIN(1),IEP)
C
C  HARDWARE SCALE OF 1 FOR BOTH CHARACTERS
       CALL PRAMTR (1,1)
C
C  SINGLE CHARACTER COMMAND FOR A 'P'
       CALL ANY(#120,1)
C
C  NOW SET UP A 'D' ON THE SCREEN TO FOLLOW CONTINUOUS
C  DATA HITS FROM THE TABLET
C
C  THE BEAM POSITION, AGAIN SETTING UP FOR LATER EDIT
       CALL POINT (750,750,0,MAIN(1),IED)
C
C  SINGLE CHARACTER COMMAND FOR A 'D'
       CALL ANY (#104,1)
C
C  HERE IS THE MAIN LOOP
C
C  FIND OUT PRESENT STATE OF PUSH BUTTONS
 44    CALL GETPSH (IB)
C
C  IF LAST BUTTON IS ON, EXIT
       IF (IB(6)) GO TO 99
C
```

Figure 8-1
Tracking With Writing Tablet (Sheet 1 of 2)

```
C  SET UP DATA MODE 0 AS A DEFAULT
       MODE=0
C
C  IF FIRST BUTTON ON, THE DATA MODE SHOULD BE NONO
       IF (IB(1)) MODE=1
C
C  GET DATA FROM TABLET
       CALL GETBLT (LUN,MODE,ITAB,IEV)
C
C  WAIT FOR COMPLETION OF REQUEST
       CALL WAITFR (IEV)
C
C  CHECK WHETHER RETURNED DATA IS PEN DATA
       IF (ITAB(1).EQ.0) GO TO 55
C
C  IT WAS CONTINUOUS DATA, SO MOVE THE 'D'
       CALL REPLOT (4,ITAB(2),ITAB(3),0,IED)
C
C  BACK TO TOP OF LOOP
       GO TO 44
C
C  WAS PEN DATA, SO MOVE THE 'P'
 55    CALL REPLOT (4,ITAB(2),ITAB(3),0,IEP)
C
C  RETURN TO LOOP TOP
       GO TO 44
C  EXIT HERE
 99    STOP
       END
```

Figure 8-1 (Cont.)
Tracking With Writing Tablet (Sheet 2 of 2)

```
C
C       PROGRAM TO INPUT AN ARBITRARY CURVE
C
        LOGICAL IB(6)
        DIMENSION MAIN (3100),ITAB(3)
C
C  START UP MAIN FILE
        MAIN(1)=0
        CALL DINIT (MAIN(1))
C
C  FIRST TIME THROUGH, DON'T WAIT ON BUTTONS
        GO TO 2
C  MAIN LOOP, DO WE GET ANOTHER CURVE?
C  WAIT FOR USER TO DECIDE
C  UNLESS PUSH BUTTON SIX SET, CONTINUE
C
 1      CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
        IF (IB(6))  GO TO 99
C
C  RE-INIT THE MAIN FILE
 2      MAIN(1)=0
C
C  MAKES POINTS BRIGHTER SO THEY WILL SHOW
        CALL PRAMTR (2,6,MAIN(1))
C
C  GET 1500 POINTS, OR STOP WHEN PEN PUSHED DOWN
        DO 33 IC=1,1500
C
C  GET THE NEXT POINT
 22     CALL GETBLT (28,1,ITAB,IEV)
C
C  WAIT
        CALL WAITFR (IEV)
C
C  IF PEN DATA, STOP GETTING POINTS
        IF (ITAB(1).EQ.0) GO TO 77
C
C  REGULAR DATA, CHECK TO SEE IF FAR ENOUGH FROM
C  LAST. OTHERWISE WE COULD OVER-INTENSIFY.
C
        IT=IX-ITAB(2)
C
C  CHECK FOR DIFFERENCE OF THREE RASTER UNITS
        IF(IT.GT.2) GO TO 66
        IF(IT.LT.-2) GO TO 66
C
C  X DIDN'T MOVE ENOUGH, CHECK Y
        IT=IY-ITAB(3)
        IF(IT.GT.2) GO TO 66
        IF(IT.GT.-3) GO TO 22
C
C  GOT A POINT FAR ENOUGH AWAY. SAVE ITS CO-ORDINATES
C  FOR THE CHECK THE NEXT TIME THROUGH
 66     IX=ITAB(2)
        IY=ITAB(3)
C
C  ADD THE DATA POINT TO THE FILE
```

Figure 8-2
Inputting With Writing Tablet (Sheet 1 of 2)

```
        CALL POINT (IX,IY,1)
C
C  LOOP CONTROL
 33     CONTINUE
C
C  TURN OFF PEN WHILE WAITING
 77     CALL CLOSE (28)
C
C  GO WAIT FOR USER TO DECIDE
        GO TO 1
C
C  EXIT HERE
 99     STOP
        END
```

Figure 8-2 (Cont.)
Inputting With Writing Tablet (Sheet 2 of 2)

CHAPTER 9

GETTING ON THE AIR WITH XVM/RSX GRAPHICS SOFTWARE

The RSX distribution tapes contain seven graphics source files. These are:

File	Description
VT.nn SRC	VT15 Display Processor I/O handler
VW.nn SRC	VW15 Writing Tablet I/O handler
VPR.nn SRC	FORTRAN-callable graphics primitive routines
CIRCLE SRC	Circle approximation routine
ROTATE SRC	Axis rotation routine
DYS.nn SRC	DYSET/DYLINK routine
TBL.nn SRC	Routine allowing FORTRAN code to access the writing tablet

To use either the VT15 or VW15 handler under RSX, the appropriate source file must be assembled, task built and installed in the system. General directions for these operations are in the System Installation Guide. Assembly parameters for both handlers are in Appendix B of Part III of this manual. For additional information, refer to section 9.1 and Appendix D of this part.

9.1 CONSOLE DIALOGUE

The console listing in Figure 9-1 shows an example procedure for installing graphics into an RSX system. The slashes (/) and text following are comments for the purposes of the manual and will not appear at the console. It is assumed that the user does not have floating-point hardware. Installation of the writing tablet handler is not shown. An expanded version of the typed FORTRAN program is provided in Figure 9-2.

In the dialogue presented, the exact version numbers of the modules and the exact program sizes may not match those obtained by every user; however, the overall flow remains accurate.

```
MCR>DTC TTO LA30 300        /TELL SYSTEM WHAT KIND OF TERMINAL
MCR>ADV VTO
MCR>ADB VWO
MCR>RCP

TYPE UNITS "NAME(BASE,SIZE)"

PARTITION
>10.6(75400,2400)
>

TYPE N TO EXIT
>

RCF OK!
MCR>
/
/                           /LOG INTO TDV, TYPICALLY AT
/                           /SOME OTHER TERMINAL, BY
/                           /TYPING CONTROL T.
/
XVM/RSX V1B000 MULTIACCESS
6/4/1976 15:30
0   USERS ALREADY LOGGED IN
SPECIFY DISK TYPE(RK,RP OR RF), UNIT AND UFD><VTX>
<VTX> UFD CREATED
TDV>ASG 19 DTO              /BRING IN FILES
TDV>FIN .LIBRX BIN          /BRING IN THE NON-FLOATING POINT LIBRARY
TDV>FIN CIRCLE BIN          /BINARY OF CIRCLE ROUTINE
TDV>FIN ROTATE BIN          /AND ROTATE.
/
/                           /IF RUNNING WITH FLOATING POINT, BRING IN
/                           /INSTEAD .LIBFX BIN,CIRCLE SRC,ROTATE SRC. THEN
/                           /COMPILE CIRCLE AND ROTATE (USE F4F).
/
TDV>FIN VT.21 BIN           /BRING IN THE VT HANDLER BINARY
TDV>FIN VPR.32 BIN          /BRING IN THE PRIMITIVES BINARY
TDV>FIN DYS.03 BIN          /AND RELOCATION ROUTINES
TDV>FIN VW.03 BIN           /AND OF VW TABLET HANDLER
TDV>FIN TBL.02 BIN          /OF GETBLT FORTRAN CALL FOR TABLET
TDV>TKB                     /TASK BUILD HANDLER
TASK BUILDER V5A            /TERMINATE WITH ALT-MODE TO TKB!
LIST OPTIONS
>EXM,SZ                     /MUST BE EXEC MODE;SZ OPTIONAL
NAME TASK
>VT....                     /HANDLERS HAVE .'S AT END OF NAME
SPECIFY DEFAULT PRIORITY
>1                          /I/O HANDLERS MUST HAVE 1
DESCRIBE PARTITION          /IN FIRST 32K; MUST BE AT LEAST 2400
>10.6
/                           /CAN REASSMBLE HANDLER 1 SCOPE (2000)
```

Figure 9-1
Simulated Console Listing (Sheet 1 of 3)

```
DESCRIBE SYSTEM COMMON BLOCKS
>                              /NONE, SO JUST HIT ALT-MODE
DEFINE RESIDENT CODE
>VT.21                         /HANDLER BINARY JUST READ IN
DESCRIBE LINKS & STRUCTURE
>                              /NO OVERLAYS, JUST HIT ALT-MODE
VT.21    51000-53320  02321

CORE REQ'D
         51000-53320  02321
TDV>INS VT....                 /INSTALL TASK IMAGE IN SYSTEM
TDV>EDI                        /CALL EDITOR TO TYPE IN PROGRAM
EDITRSX V19A
>                              /EXTRA CARRIAGE RETURN TO INPUT


INPUT
C
C               TO DISPLAY 'HI'
C
C
                LOGICAL IB(6)
                DIMENSION IFILE(100),TXT(2),ICL(2)
                DATA TXT(1)/5HHI    /
                ICL(1)=#231374
                INT=4
                LARGE=0
                IFILE(1)=0
                CALL ANY(ICL(1),1,IFILE(1))
                CALL PRAMTR (3,LARGE,INT,IFILE(1),IEDIT)
                CALL POINT (250,250)
                CALL DINIT (IFILE(1))
                CALL TEXT (TXT(1),2)
33              CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
                IF (IW.LT.2) GO TO 33
                IF (IB(6)) GO TO 99
                IF (IB(1)) INT=INT+1
                IF (IB(2)) INT=INT-1
                IF (IB(3)) LARGE=LARGE+1
                IF (IB(4)) LARGE=LARGE-1
                IF (INT.LT.0) INT=0
                IF (INT.GT.7) INT=7
                IF (LARGE.LT.0) LARGE=0
                IF (LARGE.GT.15) LARGE=15
                CALL REPLOT (2,3,LARGE,INT,IEDIT)
                GOT TO 33
99              STOP
                END            /EXTRA CARRIAGE RETURN

EDIT                           /FOR EDIT MODE
>CLOSE TESTF SRC               /CLOSE OUT WITH NAME FOR FILE
EDITRSX V19A
>E                             /IMPORTANT, MUST EXIT FROM EDITOR
```

Figure 9-1 (Cont.)
Simulated Console Listing (Sheet 2 of 3)

```
TDV>FOR BL_TESTF              /DO NOT SPECIFY EXTENSION
/
/                             /IF FLOATING POINT USE 'F4F' NOT 'FOR'
/
TDV>TKB                       /NOW TASK BUILD THE EXAMPLE PROGRAM
TASK BUILDER V5A
LIST OPTIONS
>BKR,NFP,SZ                   /BANK MODE NECESSARY FOR FORTRAN, NO FLOATING
/                             /POINT, SIZE OPTIONAL
NAME TASK
>HELLO                        /CAN BE ANY UNUSED NAME
SPECIFY DEFAULT PRIORITY
>300                          /EXACT VALUE UNIMPORTANT
DESCRIBE PARTITION
>TDV                          /GENERALLY THE ONLY ONE BIG ENOUGH
DEFINE RESIDENT CODE
>TESF,VPR.32                  /MUST TASK BUILD WITH PRIMITIVES, AND
/                             /WHATEVER ADDITIONAL (CIRCLE, ROTATE,,)
/                             /ROUTINES ARE CALLED BY THE FORTRAN PROGRAM
DESCRIBE LINKS & STRUCTURE
>                             /JUST AN ALT-MODE
TESTF    00020-00531   00512
VPR.32   00532-02564   02033
STOP     02565-02600   00014
SPMSG    02601-02724   00124
.FP      02725-02726   00002

CORE REQ'D
         00000-02726   02727
TDV>ASG 24 VT                 /STANDARD VT ASSIGNMENT
TDV>CON HELLO                 /STEP 1 FOR EXECUTION
TDV>XQT HELLO                 /STEP 2
TDV>OFF                       /BUTTON #6 EXITS PROGRAM; LOG OFF
 LOGGING OFF MULTIACCESS AT 16:07
```

Figure 9-1 (Cont.)
Simulated Console Listing (Sheet 3 of 3)

This page intentionally left blank

9.2 FULL FORTRAN EXAMPLE

Figure 9-2 is the commented version of the FORTRAN program referenced in the console example above:

```
C      THE TEXT 'HI' IS DISPLAYED ON THE SCREEN
C
C      PUSH BUTTON #1 (LEFTMOST) TO INCREASE INTENSITY
C      PUSH BUTTON #2 TO DECREASE INTENSITY
C      PUSH BUTTON #3 TO INCREASE SCALE (SIZE)
C      PUSH BUTTON #4 TO DECREASE SCALE
C      PUSH BUTTON #5 IS A NO-OP
C      PUSH BUTTON #6 (RIGHTMOST) TO EXIT
C
        LOGICAL IB(6)
        DIMENSION IFILE(100),TXT(2),ICL(1)
C
C SET UP THE 'HI' FOR THE TEXT CALL
        DATA TXT(1)/5HHI   /
C
C PUSH BUTTON CLEAR FOR THE ANY CALL
        ICL(1)=#231374
C SCOPES 0 AND 2 ONLY
C
C DEFAULT SIZE AND INTENSITY
        INT=4
        LARGE=0
C
C ZERO TOP-OF-FILE POINTER
        IFILE(1)=0
C
C START BUILDING THE DISPLAY FILE
        CALL ANY(ICL(1),1,IFILE(1))
C THIS SO BUTTONS NEVER LIGHT
C
C SET UP SIZE AND INT. GET BACK EDIT ADDRESS
        CALL PRAMTR(3,LARGE,INT,IFILE(1),IEDIT)
C
C ABSOLUTE BEAM POSITION, 250 ARBITRARY.
        CALL POINT(250,250)
C
C START UP THE VT-15
        CALL DINIT(IFILE(1))
C NOTE WE CAN HAVE SAME FILE MAIN AND SUBPICTURE
C ALSO, VT-15 CAN EXECUTE DURING FILE-BUILD
C
C NOW PLACE THE TEXT STRING
        CALL TEXT (TXT(1),2)
C
C NOW WAIT FOR INTERRUPT, THEN GET CONTROL
 33     CALL LTORPB (LPX,LPY,NAME,IB,IW,1)
C
C CHECK IF PUSH BUTTON INTERRUPT
        IF(IW.LT.2) GO TO 33
```

Figure 9-2
FORTRAN Example To Display 'HI' (Sheet 1 of 2)

```
C
C IS TIME TO EXIT
        IF(IB(6)) GO TO 99
C
C ADJUST INTENSITY AND SCALE
        IF(IB(1)) INT=INT+1
        IF(IB(2)) INT=INT-1
        IF(IB(3)) LARGE=LARGE+1
        IF(IB(4)) LARGE=LARGE-1
C
C PREVENT OUT-OF-BOUNDS VALUES
        IF(INT.LT.0) INT=0
        IF(INT.GT.7) INT=7
        IF(LARGE.LT.0) LARGE=0
        IF(LARGE.GT.15) LARGE=15
C
C EDIT NEW ONES OVER OLD
        CALL REPLOT (2,3,LARGE,INT,IEDIT)
C
C GO WAIT FOR NEXT
        GO TO 33
C
C EXIT ON LAST BUTTON
 99     STOP
        END
```

Figure 9-2 (Cont.)
FORTRAN Example To Display 'HI' (Sheet 2 of 2)

APPENDIX A

ERROR MESSAGES

A.1 ERROR MESSAGES FORMAT

Error handling under the RSX graphics system is minimal. If an error is detected, the user task is aborted and a message is printed on the user terminal. Other tasks in the system are unaffected. The message format is:

```
*** "TASKN" VPR ADDR MMMMMM CODE NNNNNN
```

where: TASKN is the task name.

MMMMMM is the address of the user call to the graphics system.

NNNNNN is an error code (refer to section A.2).

The 100000 bit of the address is set if the user is running in user mode. The address in this case is relative to the start of the task partition. A task build gives a memory map of the modules in the user partition (including FORTRAN COMMON blocks). A FORTRAN compilation with the BLO option provides a complete listing (BLS provides locations of variables and numbered statements). This information allows the user to associate the returned address with an individual call to the graphics system. If the error is in a DYSET or DYLINK call, DYS replaces VPR in the error message.

A.2 ERROR CODES

A negative error code is an RSX negative event variable for an I/O error. This error presumably occurred when the handler was doing I/O.

An error code of 0 indicates that the user provided an incorrect number of arguments in his call.

An error code of 1 indicates that the user provided an illegal selection code (first argument) to a PLOT or REPLOT call.

An error code of 2 indicates that the user provided an illegal bit in the PRAMTR (or corresponding PLOT or REPLOT) hardware feature selection argument.

An error code of 3 indicates that the user provided a CNAME with a value of zero. (It is probable that the CNAME was used to attempt an edit without first filling it with an address.)

An error code of 4 indicates that the last user call would have extended a display file across an absolute 8K boundary. The display file is left as it was prior to the erroneous call. Locations beyond the end of the display file may, however, have been modified.

An error code of 5 indicates that a user-provided count of display elements (TEXT, GRAPH or ANY) was impossible (usually occurs with a negative count).

An error code of 6 indicates the detection of any DYSET/DYLINK error, except those involving 8K boundaries. (8K boundary errors are indicated by error code 4.)

An error code of 7 indicates that a text string extended across an absolute 8K boundary.

APPENDIX B

SUMMARY OF CALLS

```
CALL ANY (ARRAY(1),NC,PNAME(1)[,CNAME]])

CALL BLANK (PNAME(1))

CALL CCLOSE

CALL CINIT (IFILE(1))

CALL CIRCLE (R,THETA,GAMMA,DEG,ISUB)

CALL COPY (RST,PNAME1(1)[,PNAME(1)[,CNAME]])

CALL DCLOSE

CALL DELETE (CNAME)
I=DELETE(CNAME)

CALL DINIT (MAIN(1))

CALL DYLINK (PNAME1(1),PNAME2(1),,PNAMEN(1))

CALL DYLINK (PNAME1(1),,,,PNAMEN(1),-1,STR1(1),,,,STRN(1))

CALL DYSET (PNAME1(1),PNAME2(1),,PNAMEN(1))

CALL DYSET (PNAME1(1),,,,PNAMEN(1),-1,STR1(1),,,,STRN(1))

CALL GETBLT (LUN,MODE,ARRAY[,IEV])

CALL GETPSH (IBT)

CALL GRAPH (DATA(1),NC,AC,PNAME(1)[,CNAME]])

CALL LINE (IDX,IDY[,INT[,PNAME(1)[,CNAME]]])

CALL LTORPB (IX,IY,NAMR,IBT,IWRCH,1)
I=LTORPB(IX,IY,NAMR,IBT,IWICH[,WAIT])

CALL PLOT (0,RST,PNAME(1)[,CNAME])

CALL PLOT (1,IX,IY[,INT[,CNAME]])

CALL PLOT (2,SELECT,VALUE[,CNAME])
```

```
CALL PLOT (2,SELECT,VALUE1,VALUE2,,,,[,CNAME])

CALL PLOT (3,STR(1),N[,CNAME])

CALL PLOT (4,IX,IY[,INT[,CNAME]])

CALL PLOT (5,DATA(1),N[,A[,CNAME]])

CALL PLOT (6,ARRAY(1),N[,CNAME])

CALL POINT (IX,IY[,INT[,PNAME(1)[,CNAME]]])

CALL PRAMTR (SELECT,VALUE[,PNAME(1)[,CNAME]])

CALL PRAMTR (SELECT,VALUE1,VALUE2,,,[,PNAME(1)[,CNAME]])

CALL REPLOT (0,RST,PNAME(1),CNAME)
I=REPLOT(0,RST,PNAME(1),CNAME)

CALL REPLOT (1,IX,IY,INT,CNAME)
I=REPLOT(1,IX,IY,INT,CNAME)

CALL REPLOT (2,SELECT,VALUE1,VALUE2,,CNAME)
I=REPLOT(2,SELECT,VALUE1,VALUE2,,CNAME)

CALL REPLOT (3,STR(1),N,CNAME)
I=REPLOT(3,STR(1),N,CNAME)

CALL REPLOT (4,IX,IY,INT,CNAME)
I=REPLOT(4,IX,IY,INT,CNAME)

CALL REPLOT (5,DATA(1),N,A,CNAME)
I=REPLOT(5,DATA(1),N,A,CNAME)

CALL REPLOT (6,ARRAY(1),N,CNAME)
I=REPLOT(6,ARRAY(1),N,CNAME)

CALL ROTATE (N,IA,IB,IC,X,Y,Z,SINA,COSA)

CALL RSETPT (IX,IY,CNAME)
I=RSETPT(IX,IY,CNAME)

CALL SETPT (IX,IY[,CNAME])

CALL TEXT (STR(1),N[,PNAME(1)[,CNAME]])

CALL TRACK (IX,IY[,IOPT,IARRAY])

CALL UNBLNK (PNAME(1))

CALL VTUNIT(N)
```

APPENDIX C

MNEMONICS COMMONLY USED IN GRAPHICS CALLS

MNEMONIC	DEFINITION
A	An integer variable or constant which indicates which axis to increment for GRAPH subroutine: 0 = increment X, set Y to data values. nonzero = increment Y, set X to data values.
ARRAY	In the GETBLT call, a three member integer array into which are returned the X-Y coordinate data and a type indicator. In the ANY call, the starting address of an integer array of VT-15 display command code provided by the user and represented as a subscripted variable.
CNAME	An integer variable that identifies the edit address (first location) which contains the display command(s) placed by the call in which CNAME is an output argument.
COSA	Floating-point cosine of angle of rotation. Used in ROTATE call.
DATA	Address which contains the array of integer data points to be plotted by the GRAPH subroutine. DATA is represented as an integer subscripted array.
DEG	Chord length of circle, expressed in Floating-point degrees of arc. Used in CIRCLE call.
GAMMA	Used in CIRCLE call to define end point of circle or arc. Expressed in floating-point degrees counter-clockwise from the positive X axis.
IA	If nonzero indicates rotation about Z axis. Used in ROTATE call.
IB	If nonzero indicates rotation about Y axis. Used in ROTATE call.
IBT	A six-element integer array which will contain a logical TRUE or FALSE for each of the six pushbuttons. An output argument of the LTORPB and GETPSH calls.
IC	If nonzero indicates rotation about X axis. Used in ROTATE call.

IDX — An integer number or variable which represents in raster units the amount the CRT beam is to be displaced from its current position in a horizontal direction. This quantity is signed to indicate the direction of displacement (i.e., + = move beam right and - = move beam left). Used in LINE call.

IDY — Same as IDX except that the indicated displacement is made in a vertical direction and the directions indicated by the sign are: + = move beam up and - = move beam down. Used in LINE call.

IEV — In the GETBLT call, an integer event variable.

INT — This integer variable indicates if the CRT beam movement is to be visible, (INT = nonzero) to draw a line, or invisible (INT = 0). Used in LINE, POINT, and PLOT calls.

IWICH — Output argument in LTORPB call. Set to 1 for light pen hit; set to 2 for push button hit; set to 3 for simultaneous light pen and push button hit.

IX — Used in POINT and SETPT calls to indicate the absolute X position to which the beam is to be moved. Absolute X coordinate of light pen hit in a LTORPB call.

IY — Used in POINT and SETPT calls to indicate the absolute X position to which the beam is to be moved. Absolute Y coordinate of light pen hit in a LTORPB call.

LUN — In the GETBLT call, an integer expression describing which logical unit is to be referenced.

MODE — In the GETBLT call, an integer expression defining the mode of operation. MODE=0, single point data; MODE=nonzero, continuous tracking.

N — Used by GRAPH subroutine to indicate the number of points to be plotted. Used by TEXT subroutine to indicate the number of characters to be displayed. Used by ROTATE subroutine to indicate number of data points to be rotated. Also used in ANY call to indicate the number of elements of the array specified that are to be moved into the display file.

NAMR — An integer which represents the contents of the name register at the time of a light pen hit (restricted to values ranging from 0 to 127). An output argument of the LTORPB call.

PNAME — The display files generated by the graphic subpicture routines are stored in dimensioned integer arrays specified by the user. The integer variable PNAME specifies the first element of the array into which commands generated by a particular call are to be stored. PNAME is always represented as a subscripted variable (e.g., PNAME(1)) except in the CIRCLE call. It will contain the length of the file and is the variable by which the file is referenced in later manipulations.

NOTE

The variable PNAME may be dropped from the statement argument lists; if dropped, the last given value for PNAME will be assumed.

PNAME1 — In a COPY call, the address of a subpicture display file called PNAME and is represented as a subscripted variable (e.g., PNAME(1)).

R — Used in CIRCLE call to specify radius of a circle in raster units.

RST — In the COPY call, this variable indicates whether the hardware SAVE/RESTORE option is to be used to save display parameters through the subroutine call. The value 0 indicates that the SAVE/RESTORE option is not to be used; a nonzero value indicates that it is to be used.

SELECT — An integer number which identifies a hardware feature(s) to be specified in the call (e.g., 1 = scale, 2 = intensity, 4 = light pen, and 8 = blink). Used in the PRAMTR call.

SINA — In the ROTATE call, represents the floating-point sine of the angle of rotation.

STR — In the TEXT call, identifies the dimensioned real array which contains the string of characters to be displayed in IOPS ASCII (Hollerith) form (five 7-bit characters per word). Represented as a subscripted variable (e.g., STR(1)).

THETA — Beginning point of circle or arc, expressed in floating-point degrees counter-clockwise from the positive X axis. Used in CIRCLE call.

VALUE — A single integer variable or constant that indicates the value or setting specified for a selected display feature in the PRAMTR call.

WAIT — An integer constant or variable, which, if nonzero, handler waits for an LTORPB interrupt before returning to user.

X — In the ROTATE call, the address of the array of floating-point X positions to be rotated.

Y — In the ROTATE call, the address of the array of floating-point Y positions to be rotated.

Z — In the ROTATE call, the address of the array of floating-point Z positions to be rotated.

APPENDIX D

ASSEMBLY PARAMETERS

The VT15 handler recognizes the assembly parameters SCOPE0, SCOPE1, SCOPE2 and SCOPE3. If no assembly parameters are specified, the handler is assembled to handle all four scopes. If any assembly parameter is specified, the resulting binary handles only those scopes specified. SCOPE1 cannot be specified if SCOPE0 is not specified and SCOPE3 cannot be specified if SCOPE2 is absent. The VT15 handler requires 2400 (octal) core locations, except for a one-scope configuration, which requires 2000 (octal) locations.

WARNING

When assembling the source file of the VT15 handler, the user must specify at least as many VT15 units as there are listed in the Physical Device List.

The FORTRAN-callable routine VPR.nn recognizes the assembly parameter QEDIT. If no parameters are provided, the FORTRAN package restarts the VT15 each time a code modification call is issued. If QEDIT is specified, this restart is not issued. The restart is present as a safety feature primarily to guard against editing different length groups over COPY calls. If speed is essential and complex editing operations are not being done, this safety feature can be removed by assembling VPR.nn with the QEDIT feature.

VPR.nn recognizes the assembly parameter SHRTAV. If this parameter is defined, the output display code consists of one location of short arbitrary vector when the absolute magnitude of both IDX and IDY is less than 32.

The other graphics programs do not have assembly parameters. Writing tablet handler VW.nn requires 1000 (octal) locations.

The following is an example of an assembly of the VT15 handler for one scope:

```
TDV>MACRO PBLX_VT.23  /PARAMETERS,BINARY,LIST,CREF
MAC-INPUT PARAMETER DEFINITIONS
SCOPE0=1
TDV>                        /AFTER CARRIAGE RETURN,CONTROL D,ALTMODE
```

INDEX