H-81-0021

Graphic 8™

COMPUTER GRAPHICS DISPLAY SYSTEM

FORTRAN SUPPORT PACKAGE [FSP]

PROGRAMMER'S REFERENCE MANUAL

Information Products Division
Federal Systems Group

SANDERS DANIEL WEBSTER HIGHWAY, SOUTH—NASHUA, NEW HAMPSHIRE 03061

Sanders Associates, Inc. reserves the right to modify the products described in this manual and to make corrections or alterations to this manual at any time without notice.

First Edition - April 1981

Reprint - August 1981

Reprint - November 1981

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

APPENDICES

SECTION 1

INTRODUCTION

This Programmers Reference Manual for the Fortran Support Package (FSP) is provided by Sanders in support of its GRAPHIC 8 interactive display terminal.

1.1 SUBROUTINE CONCEPT

FSP is a collection of Fortran-callable subroutines. The routines require little knowledge of the GRAPHIC 8 terminal, yet allow the user maximum utilization of its interactive capabilities. FSP can be tailored to a Fortran system by excluding specific modules at assembly time.

1.2 HOST COMPUTERS

FSP is designed to run in any host computer which supports Fortran and has a minimum word length of 16 bits. The actual hardware method by which the GRAPHIC 8 terminal is connected to the host is of no concern to FSP since it is I/O independent. I/O considerations such as parallel or serial interfaces, half or full-duplex, selector or multiplexer channels, etc., are incorporated in the customer-supplied I/O driver and hardware interface, leaving FSP computer independent. Depending on the host computer, Sanders, by special request, will supply the I/O driver (software).

1.3 STRUCTURE

FSP employs the distributed processing approach, because it requires and makes extensive use of the Graphic Control Program (GCP), which is resident in read-only memory in the GRAPHIC 8.

Figure 1-1 shows that the application program uses FSP by making calls to the various subroutines. FSP formats GCP compatible messages and transmits them to the GRAPHIC 8 terminal via the MSGOUT subroutine (provided by the customer). GCP in the GRAPHIC 8, processes the message to produce the desired results.

FSP also receives and interprets messages from GCP in response to a POLL request. These messages may contain keyboard, and PED* information.

1.4 FSP/GCP LINK CONTROL

As mentioned above, GCP sends messages to FSP only when polled. Each message (input or output) contains a header word to identify the message, then the remainder of the message. FSP may send a message to the GCP at any time.

---

* - PED = position entry device

1.5 ERROR DETECTION/RECEIVING

Errors generated in running FSP are detected and an error code is displayed in the upper left corner of the display screen. This error display area can be turned on or off (displayed or not displayed) by user calls to routines ENBERR, to turn error display on, or DSAERR to turn error display off. See Section 3 for a more detailed description of these routines.

Error detection is also available under program control. When the user calls EVENT, the routine which polls the terminal for an event or request response, the routine sends back an event code indicating an error has been detected. The user can now call subroutine GETERR to retrieve the error code. See Section 13 for a detailed description of the GETERR routines.

Error codes are defined in Appendix C.

1.6 COORDINATE SYSTEM

The user can define the limits of the coordinate system he will use by calling subroutine SCALE with parameters defining the lower left and the upper right coordinates of the screen. FSP converts these floating point coordinates to integer display coordinates as the various FSP routines are called. It is the display coordinates that are passed to the GCP program. Without a call to SCALE, the user coordinate system is the same resolution as the display coordinate system. The lower left point is defined as (0., 0.) and the upper right point as (+1023.,+1023.). See paragraph 3.2 for a detailed description of subroutine SCALE.

1.7 USE OF LABELED COMMON

FSP uses labeled common. The user would be careful not to use these common block names within his program. These common blocks and their dimensions are as follows:

Common Block Name	Common Block Length (Words)
TERMB	283
COORD	9
PVMD	9
LAYOT	516
MAST	9
PERIPH	16
LMEM	11

1.8 PAGING CONCEPT

A GRAPHIC 8 may be configured to have up to four 32K banks of memory for a total of 128K of memory.

GCP and the memory required to support it occupies approximately 9K of space in memory bank 0 and leaves approximately 23K of space for the user's refresh program. The entire 32K in memory banks 1, 2, and 3 is available for refresh. The approximate total useable refresh space in a 128K system therefore is 119K. The following chart summarizes the amount of user refresh program space available for the various memory configurations:

User Refresh Space	Total Memory
23K	32K
55K	64K
87K	96K
119K	128K

FSP uses a paging and mark approach wherein the following definitions are used:

"Page" Definition

- A page is a contiguous block of memory locations.
- A page may range in size from 4 memory locations to 32K-4 memory locations.
- A maximum of 255 pages may be defined.
- A page is referred to by a numeric value which ranges from 1 to 255.
- A page normally contains refresh commands generated by the various calls to FSP.
- Pages are defined by a call to LAYOUT in the host but physically exist in the memory of the GRAPHIC 8.
- A page may not cross a 32K bank boundary.
- Page 1 exists entirely in memory bank 0.
- Page 1 is always refreshed and can be thought of as the "mainline" refresh program.

1.9 DISTRIBUTED PROCESSING

1.9.1 FSP PROCESSING

1. Floating point conversion.
   Scaling: conversion of user floating point coordinates to display coordinates.
2. Clipping of off screen data.
3. Smoothing: the removing of unneeded points in defining a continuous line.
4. Formatting and transmitting the message to the GRAPHIC 8 terminal.
5. Receiving and converting all messages from the GRAPHIC 8 terminal to a manageable form for Fortran. This includes converting screen coordinates to floating point user coordinates.
6. Controls refresh file management, LAYOUT.

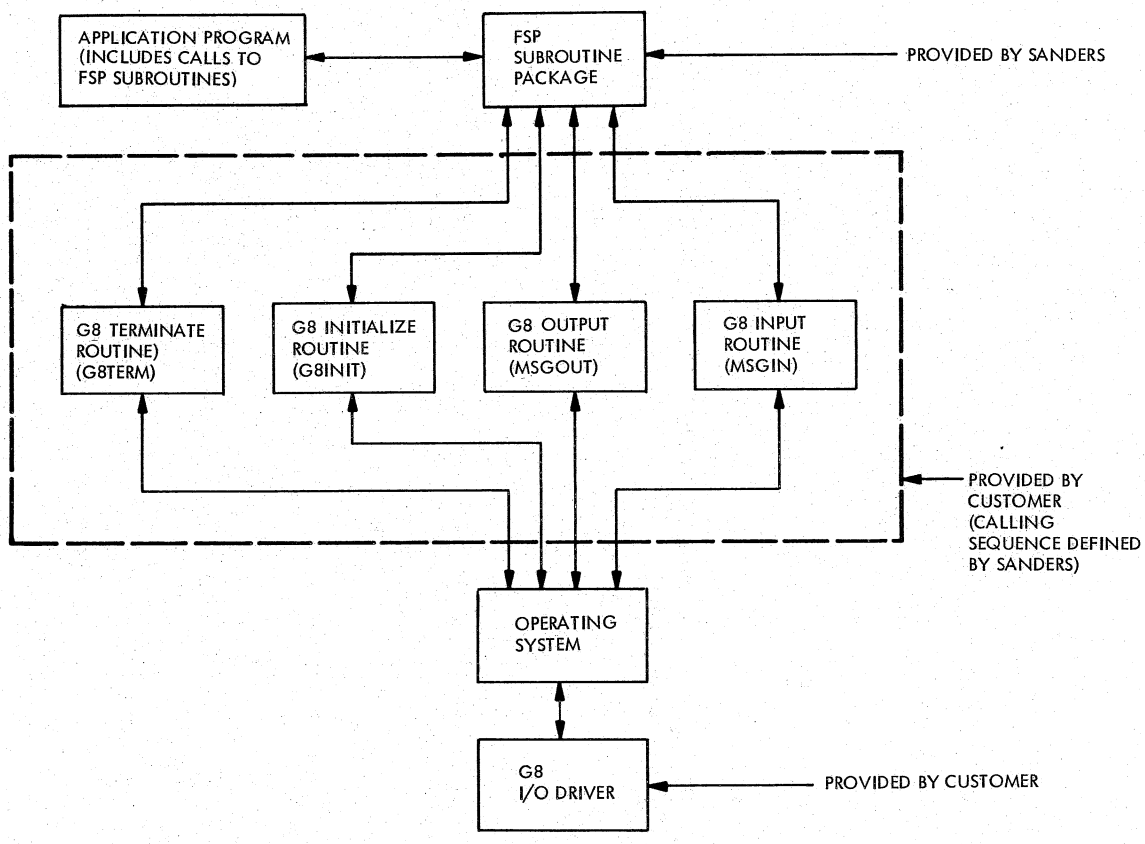

APPLICATION PROGRAM (INCLUDES CALLS TO FSP SUBROUTINES)
FSP SUBROUTINE PACKAGE
PROVIDED BY SANDERS
G8 TERMINATE ROUTINE) (G8TERM)
G8 INITIALIZE ROUTINE (G8INIT)
G8 OUTPUT ROUTINE (MSGOUT)
G8 INPUT ROUTINE (MSGIN)
PROVIDED BY CUSTOMER (CALLING SEQUENCE DEFINED BY SANDERS)
OPERATING SYSTEM
G8 I/O DRIVER
PROVIDED BY CUSTOMER
H-80-0021-001

Figure 1-1

1.9.2 GCP PROCESSING

1. Receives messages from the host computer.

2. Processes messages from the host computer.

3. Handles PED manipulations and symbol.

4. Displays alphanumeric keyboard inputs on the screen in a predefined scratchpad area.

5. Handles editing of text displayed in the scratchpad.

6. Formats all messages to the host computer.

7. Services all display interrupts.

8. Services all display peripheral devices.

9. Performs validation test and diagnostics.

1.10 FEATURES OF FSP

The standard features of FSP are specified below:

1. Fortran-callable subroutines.

2. Distributed processing: Some features are performed in the host computer, others in the GRAPHIC 8 terminal.

3. FSP is machine independent.

4. Refresh paging mechanism for organizing refresh data. This includes refresh subroutine capability.

5. The conversion of user coordinates to refresh coordinates and vice versa.

6. Modifying images presently displayed (selective updating).

7. Each copy of FSP in the host supports one GRAPHIC 8 controller with four display monitors, four keyboards, four trackballs or data tablets.

8. Operator interactions with application program:

   a) Alphanumeric keyboard

   b) Function keys

   c) Trackball, forcestick, or data tablet

9. Generation of all refresh instructions including image generation commands i.e., (MOVE, DRAW, CIRCLE, Point Plotting, Ellipse, Polygon Fill, Color/Gray Level changes).

10. Smoothing of user data to minimize the number of coordinates necessary for presenting a continuous line.

11. Local PED operation performed at the terminal:

    a. PED symbol locally updated at the terminal.

    b. Symbol may be user defined or the default symbol.

12. Lcoal keyboard manipulations performed at the terminal:

    a. Characters typed directly into a refresh scratchpad.

    b. Scratchpad area can be edited from the keyboard.

13. Mass transfer of existing refresh data to the terminal. This allows for off line generated refresh code to be passed directly to the GRAPHIC 8 terminal and inserted into the refresh memory without any additional processing.

14. Mass transfer of pixel data (useful in image data processing).

15. For inserting refresh code, two modes of operation exist:

    a. Initial or additional data.

    b. Editing data (selective updating).

16. Displayed images can be scaled, rotated and translated in two dimensions. Subroutines exist for manipulating the 2D Coordinate Converter hardware option.

17. Displayed images can be scaled, rotated and translated in three dimensions. Nine subroutines exist for manipulating the 2D/3D Coordinate Converter hardware option.

18. A maximum of 3 sections of split screen imaging.

19. Set of colors may be defined using the Red, Green, Blue or Hue, Lightness, Saturation, Model.

20. Defined convex polygons may be filled.

SECTION 2

GRAPHICS FSP SUBROUTINE LIBRARY

Setup Routines

1. INIT
2. LAYOUT
3. SCALE
4. ENBBOX
5. DSABOX
6. ENBERR
7. DSAERR
8. THEEND

Status Routines

1. TPARM
2. LMARGN
3. STATUS
4. LAMPON
5. LAMPOF
6. COLORI
7. GRAYI
8. SCOLOR
9. SGRAY

Image Generation Routines

1. MOVE
2. DRAW
3. TEXT
4. NEWLIN
5. CIRCLE
6. CLIPSE
7. XYPLOT
8. HTPLOT
9. VTPLOT
10. FILL

Page Management Routines

1. ADDREF
2. UPDATE
3. ERASEP
4. PICTUR
5. GETMRK
6. MOVEIM
7. COPYIM

Event Routine

1. EVENT

PERIPHERAL DEVICE ROUTINES

Alphanumeric/Function Keyboard Routines

1. ENBPAD
2. DSAPAD
3. GETTXT
4. GETKEY

Trackball/Forcestick/Data Tablet Routines

1. ENAPED
2. DSAPED
3. ENACUR
4. DSACUR

Packed Vector Routines

1. ENBPMD
2. PDRAW
3. PMOVE
4. DSAPMD

Two Dimensional Scale, Rotate & Translate Routines

1. CC2DBL
2. MOV2D
3. DRAW2D
4. T2D2D
5. MTRX2D
6. V2DBOX

Three Dimensional Scale, Rotate & Translate Routines

1. INIT3D
2. SCAL3D
3. CCBLK
4. MOVE3D
5. DRAW3D
6. T3D2D
7. MTRX3D
8. VIEWPT
9. VIEWBX

Image Control Routines

1. CLIP
2. SMOOTH
3. SPLIT

Data Transfer Routines

1. REFDAT
2. REQIM
3. GETIM
4. MOVDAT
5. GETERR

SECTION 3

SETUP ROUTINES

The following subroutines are described in this section:

INIT - Initialize the terminal to FSP mode.

LAYOUT - define FSP memory layout in the GRAPHIC 8 terminal.

SCALE - Define user coordinate system.

ENBBOX - Turn border display on.

DSABOX - Turn border display off.

ENBERR - Turn error display on.

DSAERR - Turn error display off.

THEEND - Terminate FSP mode.

The purpose of the routines in this section is to set up and define the general characteristics of the GRAPHIC 8. The GRAPHIC 8 is notified that it will be communicating with a host application program that is using the Fortran Support Package (FSP) and is placed in FSP mode by the user's call to INIT. The GRAPHIC 8 memory is allocated according to the specifications defined by the user in the call to LAYOUT. The viewable area or boundary (commonly called window) that the user specifies (by the call to SCALE) maps the user's coordinates to display coordinates. Only objects with coordinates within this user defined viewing area are displayed.

The status of FSP's error message area and border are controlled by the user's calls to ENBERR and ENBBOX, which enable them to be displayable, and calls to DSABOX and DSAERR to turn them off.

When the host application program has completed its task, it must call THEEND to notify the GRAPHIC 8 that it is no longer communicating with a FSP host application and to place it in teletypewriter emulation mode.

3.1 INITIALIZE THE TERMINAL TO FSP MODE

NAME: INIT

FUNCTION: Initializes FSP. This must be the first FSP routine called.

CALLING FORMAT: CALL INIT (IUNIT, IOPT, IFACE)

DESCRIPTION OF PARAMETERS:

IUNIT = Integer variable containing the logical unit number assigned to the GRAPHIC 8 I/O driver.

IOPT = Integer variable supplied by caller indicating option status.

0 = NO OPTIONS makes option memory space available for user refresh

1 = OPTIONS reserves option memory space

IFACE = Integer variable containing the type of hardware interface between the host and the GRAPHIC 8 terminal.

1 = Parallel
2 = Serial

DETAILED DESCRIPTION:

In addition to reinitializing internal FSP variables, the following visuals can be observed:

- The screen is cleared. The INIT routine causes the customer-supplied G7INIT routine described in paragraph 14.1 to be called as follows:

  CALL G7INIT (IUNIT)

  This subroutine is responsible for activating the system mode of GCP.

- A full screen border is placed on the screen to outline the displayable area.

- A two digit "error message" is displayed in the upper left corner of the screen. A successful call results in "00" being displayed.

3.2 DEFINE FSP MEMORY LAYOUT IN THE GRAPHIC 8 TERMINAL

NAME: LAYOUT

FUNCTION: Partitions the memory in the GRAPHIC 8 into pages. This routine must be the second FSP routine called (INIT is the first).

CALLING FORMAT: CALL LAYOUT (NPAGES, LNGARY)

DESCRIPTION OF PARAMETERS:

Three distinct functions can be performed by LAYOUT depending on the value of the NPAGES.

NPAGES = 1 to 255 ...User specifies memory layout
= 0 ...FSP automatically performs memory layout
= -1 ...User requests a description of how FSP would allocate memory but no allocation is made.

USER ALLOCATION:

NPAGES = an integer variable supplied by the caller indicating the number of graphic pages desired. Each element of the length array (LNGARY) contains the length in words of the corresponding graphic page.

$1 \leq NPAGES \leq 255$

LNGARY = An integer array supplied by the caller whose length is equal to NPAGES. Each element of the array must be filled in by the caller with the length in "words" of the corresponding page, i.e.,

```
LNGARY(1) = Length of page 1
LNGARY(2) = Length of page 2
    .              .
    .              .
    .              .
LNGARY(NPAGES) = Length of page NPAGES
```

The maximum size of page 1 is _____ words; the maximum size of all other pages is 32763 words.

AUTOMATIC ALLOCATION:

NPAGES = 0 Supplied by caller as an integer variable set to zero to indicate automatic allocation requested. The actual number of pages created will be returned to the caller in NPAGES.

LNGARY = A four word integer array supplied by the caller and filled in by LAYOUT. LAYOUT automatically creates 1 to 4 graphic pages, depending on the installed memory configuration. The mark length values returned in LNGARY are as follows:

- 32K systems LNGARY(1) = Length of page 1
  LNGARY(2) = -1 (no page 2 defined)
  LNGARY(3) = -1 (no page 3 defined)
  LNGARY(4) = -1 (no page 4 defined)

- 64K systems LNGARY(1) = Length of page 1
  LNGARY(2) = Length of page 2
  LNGARY(3) = -1 (no page 3)
  LNGARY(4) = -1 (no page 4)

- 96K systems LNGARY(1) = Length of page 1
  LNGARY(2) = Length of page 2
  LNGARY(3) = Length of page 3
  LNGARY(4) = -1 (no page 4)

- 128K systems LNGARY(1) = Length of page 1
  LNGARY(2) = Length of page 2
  LNGARY(3) = Length of page 3
  LNGARY(4) = Length of page 4

ALLOCATION REQUEST:

NPAGES = -1 Supplied by caller as an integer variable set to -1 to indicate configuration request; no pages allocated. The actual number of pages allowable will be returned to the caller in NPAGES.

LNGARY = A four word integer array supplied by the caller and filled in by LAYOUT. No pages are allocated and the data returned is the same as for the automatic allocation.

DETAILED DESCRIPTION:

The memory of the GRAPHIC 8 must be divided into graphic pages by using the LAYOUT subroutine before the subroutines described in the remaining sections can be used. User pages are numbered starting at 1. Page 1 is the "mainline refresh" page and all graphic orders in it are displayed. Graphic orders in pages 2 through 255 are displayed only through calling the PICTUR subroutine (see paragraph 6.4). The mark values for each graphic page created by this call are set to zero. Pages are allocated starting at the lowest memory allowable location of the first 32K memory bank and work upwards. A page is not allowed to cross 32K memory banks and LAYOUT will assign memory accordingly. If the user at some later time wishes to reallocate his pages, he must reinitialize the graphics package by calling INIT, followed by a call LAYOUT.

Example:

```
C
C    ALLOCATE 20,200 WORDS OF THE
C       GRAPHIC 8 MEMORY
C      INTO 7 PAGES USING LAYOUT
C      WHERE
C       PAGE 1 = 10,000 WORDS
C       PAGE 2 =  2,000 WORDS
C       PAGE 3 =    200 WORDS
C       PAGE 4 =  1,500 WORDS
C       PAGE 5 =  1,500 WORDS
C       PAGE 6 =  3,000 WORDS
C       PAGE 7 =  2,000 WORDS
     LNGARY (1) = 10000
     LNGARY (2) =  2000
     LNGARY (3) =   200
     LNGARY (4) =  1500
     LNGARY (5) =  1500
     LNGARY (6) =  3000
     LNGARY (7) =  2000
C
C    INITIALIZE FSP AND CALL LAYOUT FOR SEVEN PAGES
C
     CALL INIT (5,0,2)
     CALL LAYOUT (7, LNGARY)
```

3.3 DEFINE USER COORDINATE SYSTEM

NAME: SCALE

FUNCTION: Allows the caller to define the X, Y coordinates (in floating point) of the lower left and upper right coordinates of the screen. FSP maps these user coordinates to display coordinates as the various FSP routines are called.

CALLING FORMAT: CALL SCALE (XL, YL, XU, YU)

DESCRIPTION OF PARAMETERS:

(XL, YL) = Floating point variables containing the X and Y values to be assigned to the lower left corner of the displayable area.

(XU, YU) = Floating point variables containing the X and Y values to be assigned to the upper right corner of the displayable area.

DETAILED DESCRIPTION:

All calls to FSP subroutines in which X, Y coordinates are supplied convert the floating point user coordinate into an interger display coordinate. It is the display coordinate which is then placed in the currently opened page.

Without a call to SCALE, the user coordinate system is equal to the default display coordinate system, i.e.,

XL, YL = 0.,0.
XU, YU = +1023.,+1023.

3.4 TURN BORDER DISPLAY ON

NAME: ENBBOX

FUNCTION: Allows the caller to display a rectangular border around the displayable area on the selected monitor.

CALLING FORMAT: CALL ENBBOX (IDISP)

DESCRIPTION OF PARAMETERS:

IDISP = An integer variable indicating on which monitors the border is to be presented

0 - none	8 - #1
1 - #4	9 - #1 & 4
2 - #3	10 - #1 & 3
3 - #3 & 4	11 - #1, 3 & 4
4 - #2	12 - #1 & 2
5 - #2 & 4	13 - #1, 2, & 4
6 - #2 & 3	14 - #1, 2, & 3
7 - #2, 3, & 4	15 - #1, 2, 3, & 4 (default)

DETAILED DESCRIPTONS:

This routine allows the caller to selectively display the border on any or all monitors. The default condition for FSP is to have the borders dislayed on all monitors.

3.5 TURN BORDER DISPLAY OFF

NAME: DSABOX

FUNCTION: Allows the caller to remove the rectangular border from selected monitors.

CALLING FORMAT: CALL DSABOX (IDISP)

DESCRIPTION OF PARAMETERS:

IDISP = An integer variable indicating on which monitors the border indicators are to be removed. See ENBBOX for the associated monitor values.

DETAILED DESCRIPTIONS:

Removes outline around the displayable area on selected display monitors.

3.6 TURN ERROR DISPLAY ON

NAME: ENBERR

FUNCTION: Allows the caller to display the error code on the upper left position of the selected monitors.

CALLING FORMAT: CALL ENBERR (IDISP)

DESCRIPTION OF PARAMETERS:

IDISP = An integer variable indicating on which monitors the error is to be displayed. See ENBBOX for the associated display monitor values.

DETAILED DESCRIPTION:

The error display is two digits in the upper left hand corner of the selected monitors. The initial value displayed is "ØØ". If an error condition is detected, the error number is displayed and an error event is created. The error numbers are listed in Appendix C.

3.7 TURN ERROR DISPLAY OFF

NAME: DSAERR

FUNCTION: Allows the caller to remove the error display area from the selected monitors.

CALLING FORMAT: CALL DSAERR (IDISP)

DESCRIPTION OF PARAMETERS:

IDISP = An integer variable indicating on which monitors the error is to be removed. See ENBBOX for the associated display monitor values.

DETAILED DESCRIPTION:

Removes the error display from the requested monitors. Error events are still generated regardless of the status of the error display.

3.8 TERMINATE FSP MODE

NAME: THEEND

FUNCTION: Causes the GRAPHIC 8 terminal to return to the teletypewriter emulation mode of GCP. All screens are cleared before FSP is terminated.

CALLING FORMAT: CALL THEEND

DETAILED DESCRIPTION:

When the host application program is through with its FSP processing requirements, it must issue the THEEND call to notify the GRAPHIC 8 terminal that it is no longer communicating with FSP and to place it in teletypewriter emulation mode. In the emulator mode, the display operator could then cause another graphics job to be run which would issue an INIT call to put the terminal back into the FSP mode of operation.

SECTION 4

STATUS ROUTINES

The following routines described in this section allows the caller to define various attributes concerning the display data and selectively turn on or off the lights on the function keys.

TPARM -- Set text parameters for character, size, spacing, rotation and line spacing

LMARGN - Set value for left margin

STATUS - Set blinking, line style, display monitor selection and color/gray index

LAMPON - Turn keyboard function key on

LAMPOF - Turn keyboard function key off

COLORI - Define the look-up table with selectable levels of color in either RGB or HLS format

GRAYI -- Define the look-up table with selectable gray levels

4.1 SELECT TEXT PARAMETERS

NAME: TPARM

FUNCTION: Allows the caller to select text writing parameters: character size, spacing, orientation and line spacing

CALLING FORMAT: CALL TPARM (ICSIZE, CSPAC, RLSPAC, ICROT)

DESCRIPTION OF PARAMETERS:

ICSIZE = Integer variable selecting the character size desired. Character sizes are 1, 2, and 3 with the ratio of sizes being 1:2:3.

CSPAC = Real variable containing the horizontal spacing (vertical for rotated text) between the start positions of two adjacent characters. If a value of 0 is entered, then the spacing will default to the value appropriate for the specified size.

RLSPACE = Real variable supplied by the caller containing the vertical spacing (horizontal for rotated text) between start positions of two adjacent lines of text. If a value of 0 is entered, then the spacing will default to the value appropriate for the specified size.

ICROT = integer variable indicating the character orientation

0 = normal (horizontal) (default)
1 = rotate 90° CCW

DETAILED DESCRIPTION:

This routine generates the graphic orders containing the caller specified line spacing, character size, spacing, and orientation, and places them at the mark position in the currently opened page. Since the GRAPHIC 8 uses a firmware character generator to display characters, scaling has no impact on these text parameters.

4.2 SET LEFT (LOWER) MARGIN

NAME: LMARGN

FUNCTION: Allows the user to set the left margin (or lower margin for rotated text) to the current position. The margin is set to the X position for non-rotated text and the Y position for rotated text.

CALLING FORMAT: CALL LMARGN

PARAMETER DESCRIPTION: None

DETAILED DESCRIPTION:

The position should be set to that desired for the margin before the call to LMARGN is made. Internally, LMARGN inserts a character instruction containing an ASCII STX into refresh at the current mark of the currently opened page.

4.3 SET DISPLAY STATUS

NAME: STATUS

FUNCTION: Allows the caller to control blinking, intensity, line style, and monitor selection.

CALLING FORMAT: CALL STATUS (IBL, INDX, IVT, IDISP)

DESCRIPTION OF PARAMETERS:

IBL = Integer variable controlling blinking
0 = stop blinking (default)
1 = start blinking

INDX = Integer variable selecting the color or gray level index

Index values	Bits/pixel
0-1	1 + blink
0-4	2
0-8	3 + blink
0-16	4
0-128	5 + blink
0-256	6

IVT = Integer variable selecting line style
0 = solid vectors (default)
1 = dotted vectors
2 = dashed vectors
3 = dot-dashed vectors

IDISP = Integer variable selecting which monitors are to be used. See ENBBOX (Section 3.4) for the associated values. In addition, if set to A-1 the monitor is not changed. The monitors specified in the previous call to STATUS remain in effect.

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders containing the caller specified display attributes for blinking, color or gray level index, line style, and monitor selection. These orders are placed in the currently opened page at the current location. The display will remain in the specified status until changed by another call to STATUS.

4.4 TURN KEYBOARD LAMP ON

NAME: LAMPON

FUNCTION: Allows the caller to turn on a selected lamp on the selected keyboard.

CALLING FORMAT: CALL LAMPON (KBD, LAMP)

DESCRIPTION OF PARAMETERS:

KBD = Integer variable specifying on which keyboard lamps are to be lit.
1 - Keyboard 1
2 - Keyboard 2
3 - Keyboard 3
4 - Keyboard 4

LAMP = Integer variable specifying which of the lighted function keys is to be lighted. NOTE: If LAMP = -1 then all lamps are turned on.

DETAILED DESCRIPTON:

Lamps are numbered 0-31. The top row is numbered 16-31, left to right. The lamp number is the same as the key number.

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MATRIX KEYS:

7	8	9	15
4	5	6	14
1	2	3	13
10	0	11	12

4.5 TURN KEYBOARD LAMP OFF

NAME: LAMPOF

FUNCTION: Allows the caller to turn off a selected lamp on a selected keyboard.

CALLING FORMAT: CALL LAMPOF (KBD, LAMP)

DESCRIPTION OF PARAMETERS:

KBD = Integer variable specifying on which keyboard lamps are to be extinguished.
- 1 - Keyboard 1
- 2 - Keyboard 2
- 3 - Keyboard 3
- 4 - Keyboard 4

LAMP = Integer variable specifying which of the lighted function keys is to be turned off. NOTE: If LAMP = -1, then all lamps are turned off.

DETAILED DESCRIPTION:

Lamps are numbered 0-31. The top row is numbered 16-31, left to right. The lamp number is the same as the key number.

FUNCTION KEYS

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MATRIX KEYS:

7	8	9	15
4	5	6	14
1	2	3	13
10	0	11	12

4.6 DEFINE COLOR INDEX TABLE

NAME: COLORI

FUNCTION: Allows the caller to define the look-up table of selectable colors.

CALLING FORMAT: CALL COLORI (IDISP, INUM, ISNDX, ARRAY1, ARRAY2, ARRAY3, IMODEL)

DESCRIPTION OF PARAMETERS:

IDISP = Integer variable specifying the selected monitors (see ENBBOX routine (Section 3.4) for associated values).

INUM = Integer variable supplied by the caller defining the number of color indices to be changed.

ISNDX = Integer variable supplied by the caller specifying the first index value to be changed.

ARRAY 1 = Real array containing either the desired RED value or HUE. The RED value may vary from 0 to 1 with 1 corresponding to maximum RED. Hue will vary from 0° to a maximum of 360°.

ARRAY 2 = Real array containing either the desired Green value or Lightness. All values of the elements may vary from 0 to 1 with 1 corresponding to maximum green or lightness.

ARRAY 3 = Real array containing either the desired Blue value or Saturation. All values of the elements may vary from 0 to 1 with 1 corresponding to maximum blue or saturation.

IMODEL = Integer variable supplied by the caller defining the color model desired.
0 = RGB entries in ARRAY (1-3)
1 = HLS entries in ARRAY (1-3)

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders and index table to modify the look-up table. After the instruction has been completed there will be a jump inserted to transfer control back to master refresh. This call should not be in the normal refresh cycle but put off in a page set aside for certain items that should only be executed once.

The routine uses either RGB model or HLS model. If HLS is specified then the routine uses the algorithm that was published in the Status Report of the Graphic Standards Planning committee, Computer Graphics of ACM Vol. 13, No. 3, Aug. 1979. If RGB is specified the values will be put ito the corresponding Red, Green, and Blue Bits in the look-up table.

NOTE

Calls to GRAYI, COLORI, MOVDAT, and SPLIT should not be inserted inline in a currently open page. Instead a temporary page should be set aside to contain nothing but the GRAYI, COLORI, MOVDAT, or SPLIT call. The effect is to execute the instruction in this page only once rather than each refresh cycle. These instructions change values in the internal look-up table and once they are changed they do not have to be executed on each refresh cycle.

NOTE (Cont)

Following is an example that demonstrates how this should be implemented. Page 9 has been set aside as the temporary page. RED, GREEN, and BLUE have previously been set up as real arrays.

```
CALL UPDATE (9,0)
CALL ERASEP
CALL COLORI (15, 1, 0, RED, GREEN, BLUE, 0)
CALL ADDREF (1)
```

At the end a call is made to ADDREF to reopen the previous open page and continue inserting graphic orders.

4.7 DEFINE GRAY LEVEL INDEX TABLE

NAME: GRAYI

FUNCTION: Allows the user to define the look-up table of selectable gray levels.

CALLING FORMAT: CALL GRAYI (IDISP, INUM, ISNDX, ARRAY)

DESCRIPTION OF PARAMETERS:

IDISP = Integer variable specifying the monitors to be used (see ENBBOX routine (Section 3.4) for associated values).

INUM = Interger variable supplied by the caller defining the number of gray level indices to be changed.

ISNDX = Integer variable supplied by the caller specifying the first index value to be changed.

ARRAY = Real array containing the desired gray level values. The elements may vary from 0 to 1 with 0 corresponding to black and 1 corresponding to white.

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders and index table to modify the look-up table. After the instruction a jump will be inserted to transfer control back to master refresh. This call should not be in the normal refresh cycle but put in a page set aside for certain items to be executed only once. When this routine is call the graphic orders will be put down at the currently opened page at the current mark, therefore a call to ADDREF must be made before this call. The routine will refresh itself and when the jump instruction is executed, the normal refresh cycle will takeover. (See example under COLORI Section 4.5.)

4.8 SELECT COLOR

NAME: SCOLOR

FUNCTION: Allows the caller to select 1 of 256 colors as defined by the last COLORI instruction executed.

CALLING FORMAT: CALL SCOLOR (ICOLOR)

DESCRIPTION OF PARAMETERS:

ICOLOR = Integer variable from Ø to 256 supplied by the caller defining the color desired.

DETAILED DESCRIPTION:

This routine generates a graphic order to change the color of all following images and places it at the mark position of the currently opened page.

4.9 SELECT GRAY LEVEL

NAME: SGRAY

FUNCTION: Allows the caller to select 1 of 256 gray levels as defined by the last GRAYI instruction executed.

CALLING FORMAT: CALL SGRAY (IGRAY)

DESCRIPTION OF PARAMETERS:

IGRAY = Integer variable supplied by the caller defining the gray level desired.

DETAILED DESCRIPTION:

This routine generates a graphic order to change the gray level of all following images and places it at the mark position in the currently opened page.

SECTION 5

IMAGE GENERATION ROUTINES

The subroutines described in this section permit the application programmer to describe and draw objects in user coordinates.

MOVE - Move beam to the position specified

DRAW - Draw a line

TEXT - Display text characters

NEWLIN - Execute a carriage return, line feed to left margin

CIRCLE - Draw a circle

ELIPSE - Draw an ellipse

XYPLOT - Plot series of X, Y points

HTPLOT - Tabular plot in X direction

VTPLOT - Tabular plot in Y direction

FILL - Fill a convex polygon

5.1 MOVE BEAM TO THE POSITION SPECIFIED

NAME: MOVE

FUNCTION: Generates either an absolute or relative move graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL MOVE (X, Y, MODE)

DESCRIPTION OF PARAMETERS:

X, Y =

Absolute mode (MODE - 0 or 2)

Absolute X, Y coordinate of the desired position. The coordinate is in the user coordinate system.

Relative mode (MODE - 1)

Relative X, Y coordinate (deltas) to be moved from the current position. These relative values are also in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic orders to be generated.

0 = X, Y supplied is absolute and absolute graphic orders are to be generated

1 = X, Y supplied is relative and relative graphic orders are to be generated

2 = X, Y supplied is absolute but relative graphic orders are to be generated relative to the last absolute coordinate with MODE = 0

DETAILED DESCRIPTION:

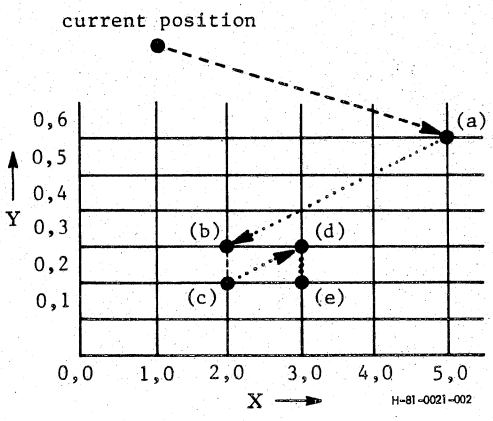
Mode = 2 is provided to allow a user whose data base contains only absolute X, Y coordinates to produce relative graphic orders without calculating the deltas. The position is changed to the specified X, Y values but nothing will be seen on the monitor.

Example:
```
C  The following call produces an absolute graphic order
C  which moves              to (1,1)
                    CALL  MOVE (1.,1.,0)
C  The following call produces a relative graphic order
C  which draws              from absolute (1,1) to absolute
C  (3,3).  The deltas computed are (2,2).
                    CALL DRAW (3.,3.,2)
C  The following call produces another relative graphic
C  order
C  which draws              from absolute (3,3) to absolute (6,7).
C  The deltas computed are (3,4).
                    CALL DRAW (6.,7.,2)
C
```

The end result of the above example is that absolute coordinates were used to create a relative entity (entity consisting of an absolute move and two relative vectors).

Example

```
a)  CALL MOVE (5.,6.,0)     !ABS
b)  CALL MOVE (-3.,-3.,1)   !REL
c)  CALL MOVE (2.,2.,3)     !ABS
d)  CALL MOVE (3.,3.,2)     !REL
e)  CALL MOVE (3.,2.,2)     !REL
```



• Beam Position After Move

--------Absolute Move

.....Relative Move

5.2 DRAW A LINE

NAME: DRAW

FUNCTION: Generates either an absolute or relative draw graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL DRAW (X, Y, MODE)

DESCRIPTION OF PARAMETERS:

X, Y =

Absolute mode (MODE - 0, 2)

Absolute X, Y coordinate of the end point of a line to be drawn. The coordinate is in the user coordinate system.

Relative mode (MODE - 1)

Relative X, Y coordinate to be used in drawing a line from the current position to a new position. These relative values are also in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic orders to be generated.

0 = X, Y supplied is absolute and an absolute draw graphic order is to be generated

1 = X, Y supplied is relative and a relative draw graphic order is to be generated

2 = X, Y supplied is absolute but a relative draw graphic order is to be generated

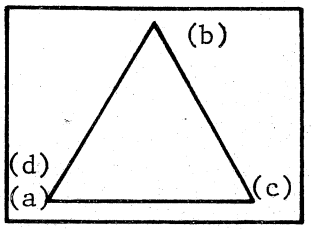
DETAILED DESCRIPTON:

The behavior of this routine is almost identical to the MOVE subroutine except that DRAW graphic orders are produced rather than MOVE graphic orders.

The attributes of the line drawn as a result of this call are determined by previous user calls to the STATUS routine which sets up (1) the type of line (solid, dotted, dashed, dot-dashed), (2) blink or no blink (3) intensity level Ø-7. The gray level may also have been set by a call to GRAYI. If the system has a color monitor the color will have been set by a call to COLORI.

Example:

```
C
C     DRAW A TRIANGLE USING ABSOLUTE COORDINATES
C
```



H-81-0021-003

```
      CALL MOVE (1.,1.,0)  (a)
      CALL DRAW (3.,4.,0)  (b)
      CALL DRAW (5.,1.,0)  (c)
      CALL DRAW (1.,1.,0)  (d)

C
C     DRAW THE SAME TRIANGLE USING RELATIVE COORDINATES
C

      CALL MOVE (1.,1.,0)  (a)
      CALL DRAW (2.,3.,1)  (b)
      CALL DRAW (2.,-3.,1) (c)
      CALL DRAW (-4.,0.,1) (d)
```

5.3 DISPLAY TEXT CHARACTERS

NAME: TEXT

FUNCTION: Generates text graphic orders and places them starting at the mark position of the currently opened page.

CALLING FORMAT: CALL TEXT (N, IARRAY)

DESCRIPTION OF PARAMETERS:

N = An integer variable supplied by the caller indicating the number of text characters to be displayed.

$$1 \leq N \leq 86$$

IARRAY = An integer array supplied by the caller in which each element of the array contains a 7 bit ASCII character code right adjusted in the element (see Appendix B for character codes).

NOTE
The 8th bit is always set.

DETAILED DESCRIPTION:

When the currently opened page is displayed, the GRAPHIC 8 displays text starting at the current position in either a horizontal or vertical direction with character size and spacing determined by a previous user call to TPARM. The text intensity, blink or no blink, and color (color monitors only) has also been determined by calls to other FSP routines. The current position after the text is displayed is after the last text character drawn (blanks included). If N is odd, a null character is stored as the last text character.

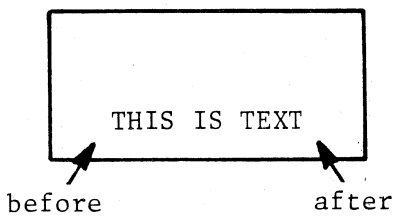
NOTE

The TEXT routine is not normally used directly in a user program. Instead it is strongly recommended that the user write a routine that will convert a literal character string into a form usable by TEXT. The calling sequence should be:

```
CALL SETEXT ('ABCDE',ICOUNT)
```

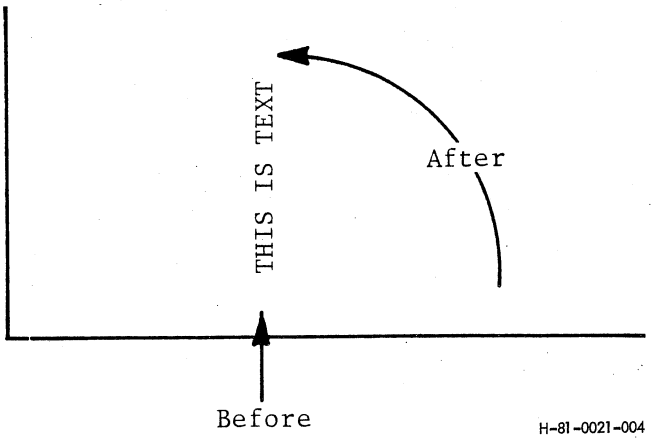
where 'ABCDE' is the literal character string and ICOUNT is a count of the number of characters in the string. Since this routine is necessarily dependent upon the word size of the host computer, it must be individually written. This routine has been written for several host computers and can thus be supplied by Sanders Associates for these hosts.

```
C
C     NON-ROTATED TEXT
C


      CALL TPARM (1,0.,0.,0)
      CALL MOVE (1.,1.,0)
      CALL SETEXT ('THIS IS TEXT',12)
```



X, Y Position

```
C
C     ROTATED TEXT
C


      CALL TPARM (1,0.,0.,1)
      CALL MOVE (1.,1.,0)
      CALL SETEXT (THIS IS TEXT;12)
```



X, Y Position

5.4 NEW LINE

NAME: NEWLIN

FUNCTION: NEWLIN performs the analoguous function of typing a carriage return, line feed at a terminal. The position is changed to that set by the last call to LMARGN and the line spacing is updated by the amount specified in the last call to TPARM.

CALLING FORMAT: CALL NEWLIN

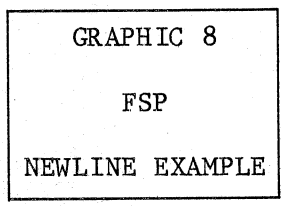
PARAMETER DESCRIPTION: NONE

DETAILED DESCRIPTION:

NEWLIN inserts a text instruction containing a carriage return, line feed into the currently opened page at the current mark. This instruction will change the position to the left margin (lower margin if characters are rotated). The left margin should have been set up prior to this call using subroutine LMARGN. The line spacing is also updated by the amount specified in the last call to TPARM.

Example:

```
C    DISPLAY THE FOLLOWING TEXT ON THE SCREEN
C         GRAPHIC 8
C         FSP
C         NEW LINE EXAMPLE
C    DIMENSION IPGS (2)
     DATA IPGS 11000.,1000.,1
     CALL INIT (5,0,2)
     CALL LAYOUT (2,IPGS)
     CALL SCALE (0.0,0.0,1000.,1000.)
     CALL MOVE (300.,550.,0)
     CALL LMARGN
     CALL TPARM (3,0.,50.,0)
     CALL SETEXT ('GRAPHIC 8',9)
     CALL NEWLIN
     CALL SETEXT ('FSP',3)
     CALL NEWLIN
     CALL SETEXT ('NEWLINE EXAMPLE',15)
     CALL DATEND
     STOP
     END
```



5.5 DRAW A CIRCLE

NAME: CIRCLE

FUNCTION: Allows the caller to display specified quadrants of a circle centered at the point defined by the current position.

CALLING FORMAT: CALL CIRCLE (RADIUS, IQUAD)

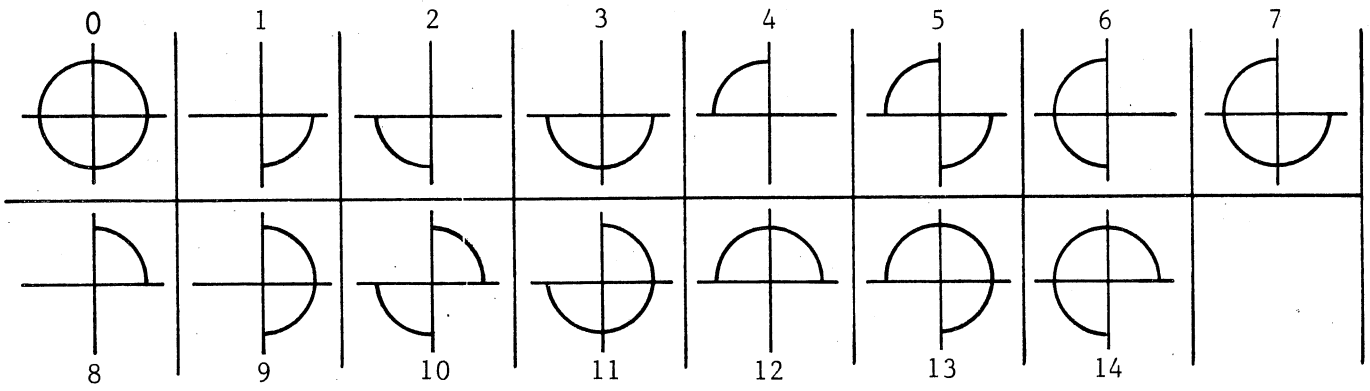
DESCRIPTION OF PARAMETERS:

RADIUS = Radius of the circle in user coordinates.

IQUAD = Which quadrants of the circle are to be displayed.
where:

Ø = turn on all quadrants
1 = turn on quadrant 4 only
2 = turn on quadrant 3 only
3 = turn on quadrants 3 and 4 only
4 = turn on quadrant 2 only
5 = turn on quadrants 2 and 4 only
6 = turn on quadrants 2 and 3 only
7 = turn on quadrants 2, 3, and 4 only
8 = turn on quadrant 1 only
9 = turn on quadrants 1 and 4 only
10 = turn on quadrants 1 and 3 only
11 = turn on quadrants 1, 3, and 4 only
12 = turn on quadrants 1 and 2 only
13 = turn on quadrants 1, 2, and 4 only
14 = turn on quadrants 1, 2, and 3 only

IQUAD



81-0021-005

DETAILED DESCRIPTION:

This routine places the necessary graphic order at the mark position of the currently opened page. When the currently opened page is displayed, the GRAPHIC 8 displays a circle or a series of quadrants (see description of IQUAD) at a distance equal to RADIUS around the current position of the beam. The current position of the beam remains unchanged.

5.6 DRAW AN ELLIPSE

NAME: ELIPSE

FUNCTION: Allows the caller to display specified quadrants of an ellipse centered at the point defined by the current X and Y position.

CALLING FORMAT: CALL ELIPSE (XSEMI, YSEMI, IQUAD)

DESCRIPTION OF PARAMETERS:

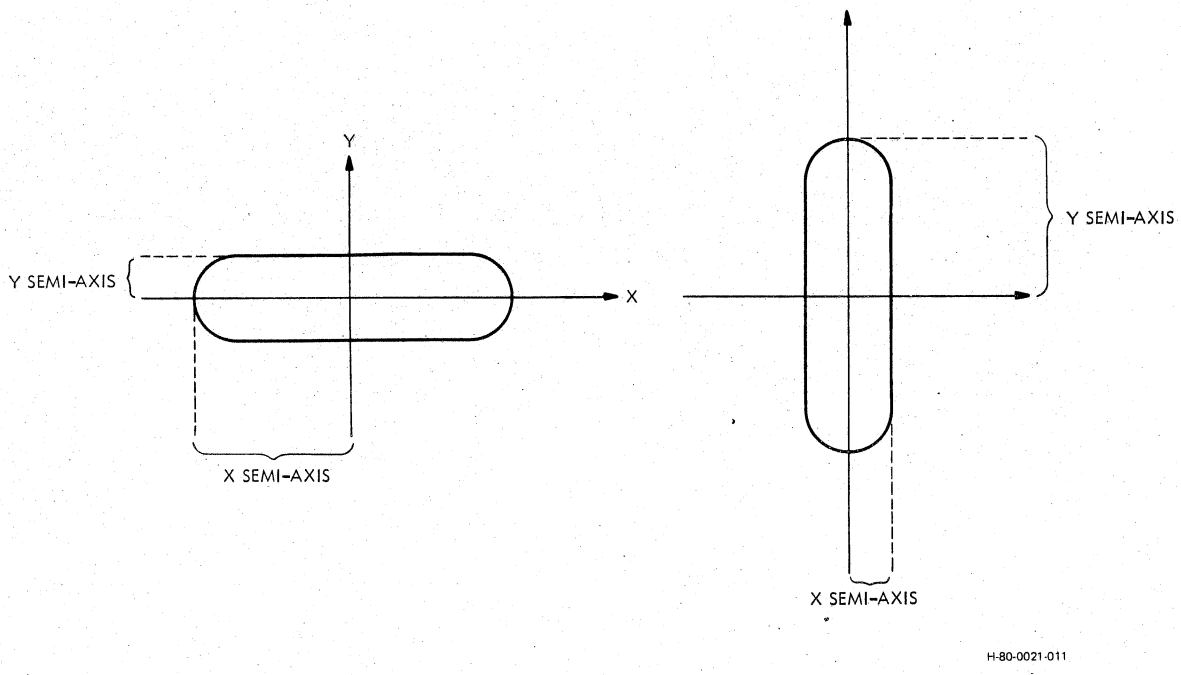
XSEMI = Length of horizontal semi-axis of the ellipse in the user coordinates

YSEMI = Length of vertical semi-axis of the ellipse in the user coordinates

IQUAD = The quadrants of the ellipse which are to be displayed. (See CIRCLE for the definition of IQUAD.)

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders and places them at the mark position of the currently opened page. The following diagrams show sample ellipses.



5.7 PLOT A SERIES OF X, Y POINTS

NAME: XYPLOT

FUNCTION: Allows the caller to plot a series of X, Y point pairs.

CALLING FORMAT: CALL XYPLOT (INUM, XARRAY, YARRAY, IMODE)

DESCRIPTION OF PARAMETERS:

INUM = Integer variable supplied by the caller specifying the number of X, Y pairs to be plotted.

XARRAY = Real array supplied by the caller in which each element contains either the absolute X coordinate or relative X coordinate (depending on the value of IMODE).

YARRAY = Real array supplied by the caller in which each element contains either the absolute Y coordinate or relative Y coordinate (depending on the value of IMODE).

IMODE = Integer variable supplied by the caller specifying whether the XARRAY and YARRAY elements are absolute or relative.

0 = absolute
1 = relative

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders to plot a point at each of the X, Y positions specified. These graphic orders are inserted at the mark position of the currently opened page. If relative mode is used the initial X, Y position should be set by the user.

5.8 HORIZONTAL TABULAR PLOT

NAME: HTPLOT

FUNCTION: Allows the caller to plot a series of points at equal intervals along the X axis.

CALLING FORMAT: CALL HTPLOT (TAB, INUM, YARRAY, UMODE)

DESCRIPTION OF PARAMETERS:

TAB = Real variable supplied by the caller containing the X tabular increment defined in user coordinates.

INUM = Integer variable supplied by the caller specifying the number of points to be plotted.

YARRAY = Real array supplied by the caller in which each element contains either the absolute Y coordinate or the relative Y displacement.

IMODE = Integer variable supplied by the caller specifying whether the elements of YARRAY are absolute of relative.

0 = absolute
1 = relative

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders to do a tabular plot of the YARRAY values and places them at the mark position of the currently opened page. The initial X, Y position should be set by the user.

5.9 VERTICAL TABULAR PLOT

NAME: VTPLOT

FUNCTION: Allows the caller to plot a series of points at equal intervals along the Y axis.

CALLING FORMAT: CALL VTPLOT (TAB, INUM, XARRAY, IMODE)

DESCRIPTION OF PARAMETERS:

TAB = Real variable supplied by the caller containing the Y tabular increment defined in user coordinates.

INUM = Integer variable supplied by the caller specifying the number of points to be plotted.

XARRAY = Real array supplied by the caller in which each element contains either the absolute X coordinate or the relative X displacement.

IMODE = Integer variable supplied by the caller specifying whether the elements of XARRAY are absolute or relative.

0 = absolute
1 = relative

DETAILED DESCRIPTION:

This routine generates the necessary graphic orders to do a tabular plot of the XARRAY values and places them at the mark position of the currently opened page. The initial X, Y position should be set by the user.

5.10 FILL A CONVEX POLYGON

NAME: FILL

FUNCTION: Allows the caller to display a solid convex polygonal area.

CALLING FORMAT: CALL FILL (INUM, ARRAY, IMODE)

DESCRIPTION OF PARAMETERS:

INUM = Integer variable supplied by the caller specifying the number of vertices for the polygon to be displayed.

ARRAY = Real array supplied by the caller containing the X and Y values of or the intervals between adjacent vertices defined in user coordinates. The vertices may be specified in either clockwise or counterclockwise order. ARRAY will have twice as many elements as the number of vertices as it to be structured as follows:

```
ARRAY (1) = X1 or X1 interval
ARRAY (2) = Y1 or Y1 interval
ARRAY (3) = X2 or X2 interval
ARRAY (4) = Y2 or Y2 interval
 .              .
 .       =      .
 .              .
ARRAY (2*INUM) = Y (INUM) or Y (INUM) interval
```

NOTE

For fastest fill specify the vertex with the maximum Y value and proceed clockwise.

IMODE = Integer variable supplied by the caller specifying whether the elements of ARRAY are absolute points or intervals.

0 - Each vertex is described by an absolute X and Y position.

1 - Each vertex is described by a relative X and Y displacement from the current X and Y position.

DETAILED DESCRIPTION:

This routine generates the graphic orders and related vertex table and places them at the mark position in the currently opened page. The vertex table will occupy twice as many words as the number of vertices defined and care must be taken to maintain the integrity of the refresh if it becomes necessary to change the number of vertices. The current X and Y position remains unchanged after the fill function has been completed.

SECTION 6

PAGE MANAGEMENT ROUTINES

The page management routines are used to select the memory address in the GRAPHIC 8 that will be used to store the next graphic instruction. The GRAPHIC 8 memory address is calculated internally in FSP based on the current page selected and the current mark position. Each time the application program calls one of the image generation routines (MOVE, DRAW, TEXT, Plotting Routines, etc.), FSP generates the equivalent graphic controller instructions which are sent to the GRAPHIC 8 and stored in the GRAPHIC 8 memory.

The page management routines consist of the following subroutines:

ADDREF - Open page at end of page mark for adding refresh data
UPDATE - Open page at specified page mark for editing refresh data
ERASEP - Erase from current page mark to end of page
PICTUR - Graphic subroutine call
GETMRK - Get current page mark information
MOVEIM - Move a block or graphic orders
COPYIM - Copy a block of graphic orders

Refresh data refers to the block (or group) of graphic controller instructions that are used to display the desired image on the monitor.

When the application program calls the LAYOUT subroutine, the GRAPHIC 8 memory is sectioned into pages. For example, if 3 pages were selected and page 1 length was 2000, page 2 length was 1000, and page 3 length was 500, then the GRAPHIC 8 memory would look as follows:

ADDRESS (OCTAL)	GRAPHIC 8 MEMORY
0	Memory space used by GCP
4550*	Start of Page 1
14410	Start of Page 2
20330	Start of Page 3
22300	End of memory for use by FSP

*Approximate start address (actual start address is contained in location 722 (octal)).

**A mark selection of 500 would result in an error code being generated because the length of page 3 was only 500 and valid marks would be in the range of 0 to 499.

These addresses were selected for illustrative purposes and may not be the same memory addresses that would be used by an FSP program. Note that in this example all addresses above 22300 are unassigned and would be unavailable for storage of refresh data.

When refresh data is to be added to GRAPHIC 8 memory, the address is selected by adding the start address of the current page to the current mark. The following table indicates the GRAPHIC 8 memory address that would be selected, based on the current page and current mark.

CURRENT PAGE	CURRENT MARK (DECIMAL WORDS)	GRAPHIC 8 MEMORY ADDRESS (OCTAL)
1	0	4550
1	5	4562
2	0	14410
2	876	17740
3	0	20330
3	499	22276
3	500	22300

To illustrate the principles involved when using the page management routines, a simple FSP program will be reviewed in the areas related to page management. The program is given below; the image that would be displayed on a CRT for this program is shown in figure 6-1.

NOTE

Please read the subroutine descriptions for ADDREF, UPDATE, ERASEP, PICTUR, and GETMRK before continuing.

An FSP program which generates the display image in figure 6-1 is given below:

```
LINE NO.

10          Dimension IPGS (3)
20          Data IPGS /500,500,500/
30          Call INST (3, 0, 2)
40          Call LAYOUT (3, IPGS)
50          Call SCALE (0.0, 0.0, 12.0, 12.0)
60          Call ADDREF (1)
70          Call MOVE (6.0, 6.0,0)
80          Call DRAW (7.0, 5.5, 0)
90          Call DRAW (8.0, 5.5, 0)
100         Call SETEXT ('TEST ONE', 8)
110         Call ADDREF (2)
120         Call MOVE (-.5, -.5, 1)
130         Call DRAW (1.0, 0., 1)
140         Call DRAW (0., 1.0, 1)
150         Call DRAW (-1.0, 0., 1)
160         Call DRAW (0., -1.0, 1)
170         Call ADDREF (3)
```

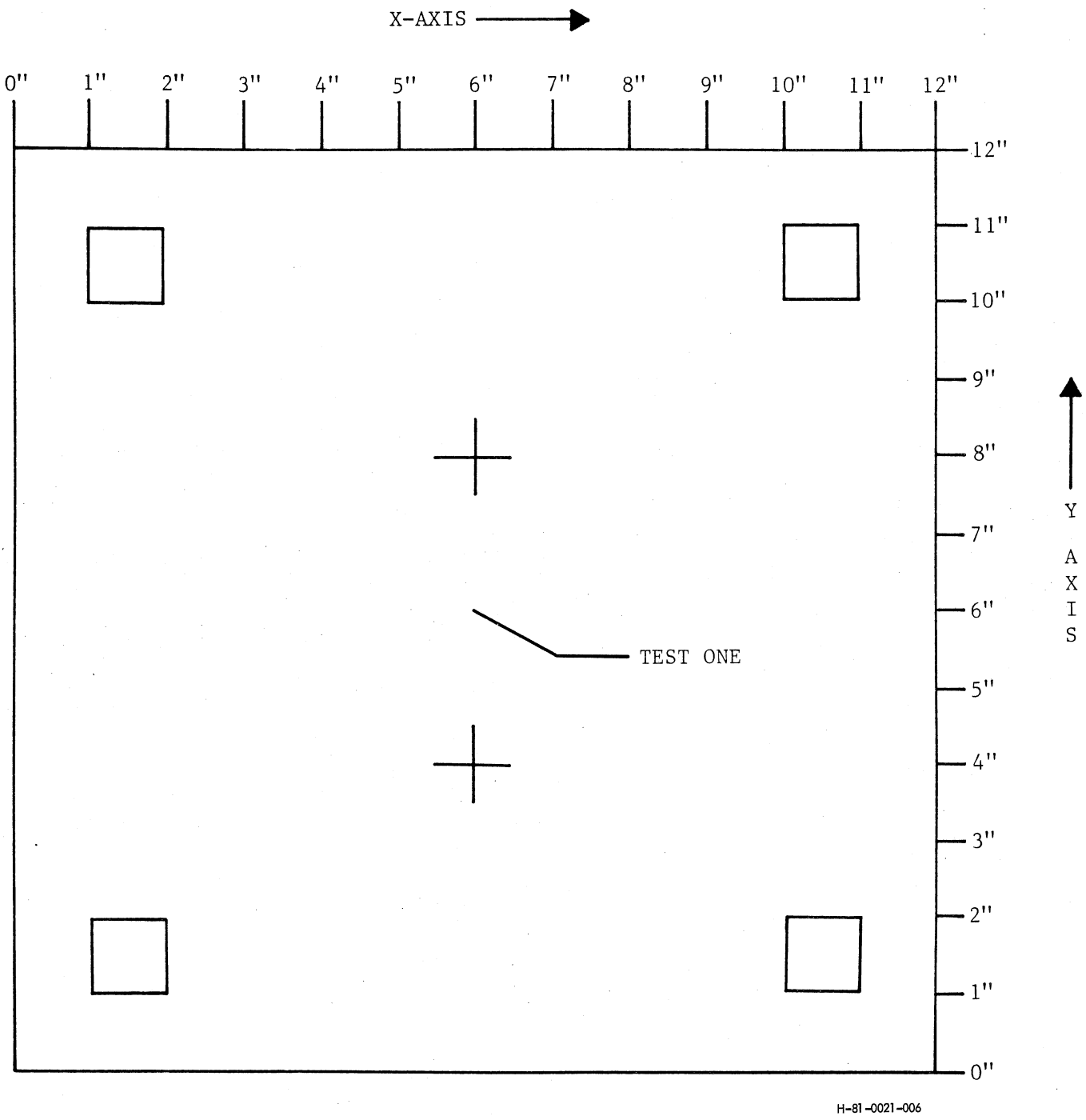


Figure 6-1

LINE NO.

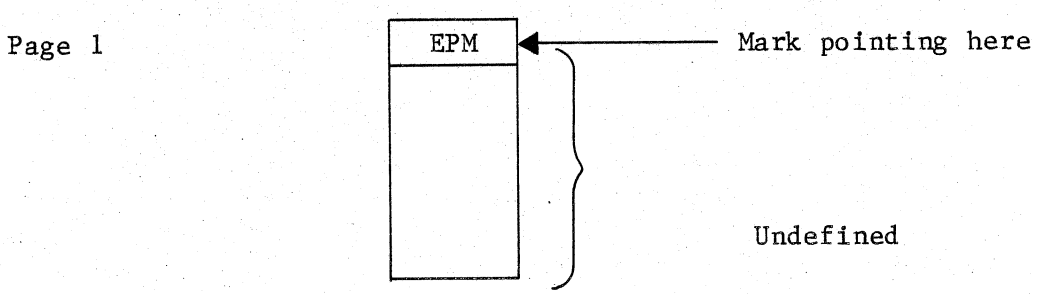
```
180          Call MOVE (-.5, 0., 1)
190          Call DRAW (1.0, 0., 1)
200          Call MOVE -.5, -.5, 1)
210          Call DRAW (0, 1.0, 1)
220          Call ADDREF (1)
230          Call MOVE (1.5, 10.5, 0)
240          Call PICTUR (2)
250          Call MOVE (10.5, 10.5, 0)
260          Call PICTUR (2)
270          Call MOVE (10.5, 1.5, 0)
280          Call PICTUR (2)
290          Call MOVE (1.5, 1.5, 0)
300          Call PICTUR (2)
310          Call MOVE (6.0, 8.0, 0)
320          Call PICTUR (3)
330          Call MOVE (6.0, 4.0, 0)
340          Call PICTUR (3)
```

The call to INIT (line 30) initializes the FSP program and a full screen box and an error code value of "00" are displayed on the monitor.

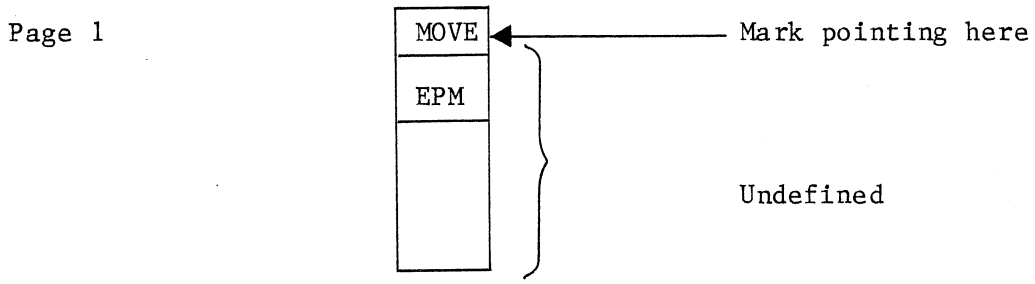
The call to LAYOUT (line 40) sections GRAPHIC 8 memory into three pages of 500 words each. The first word of each page (i.e., mark Ø) is set up to contain an end of page mark. The end of page mark (EPM) is equivalent to a return statement in a subroutine.

The call to SCALE (line 50) sets up FSP to map all values in the range of "0" to "12" into the equivalent monitor coordinate system. The user coordinate system defines the lower left corner of the monitor as 0".0" and the upper right hand corner as 12", 12". For the monitor coordinate system the lower left corner is always -512,-512 and the upper right corner is always +511, +511. This is always true regardless of which values are specified in the call to SCALE.

The call to ADDREF (line 60) opens up page 1 in the add mode. In the add mode, an end of page mark (EPM) is added after each refresh data word is stored in page 1. This call also sets up a GRAPHIC 8 memory address pointer to point to the first word (mark Ø) in page 1. After the call to ADDREF, page 1 looks as follows:

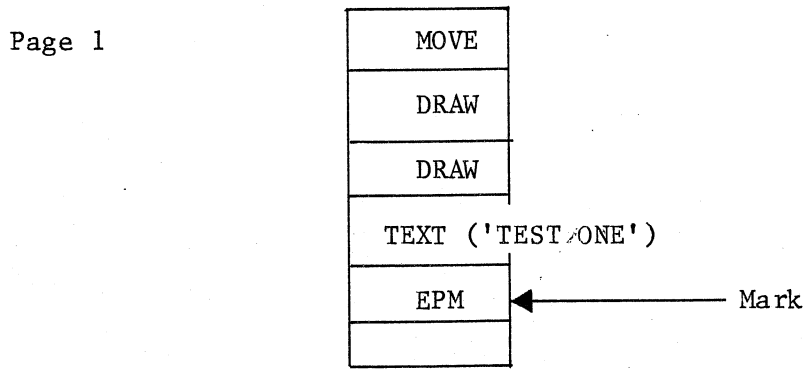


After the call to MOVE (line 70), page 1 looks as follows:



Note that after the call to MOVE, the mark value points to the new address in which the EPM is stored.

After the two calls to DRAW (lines 80 and 90) and the call to SETEXT (line 100), page 1 looks as follows:

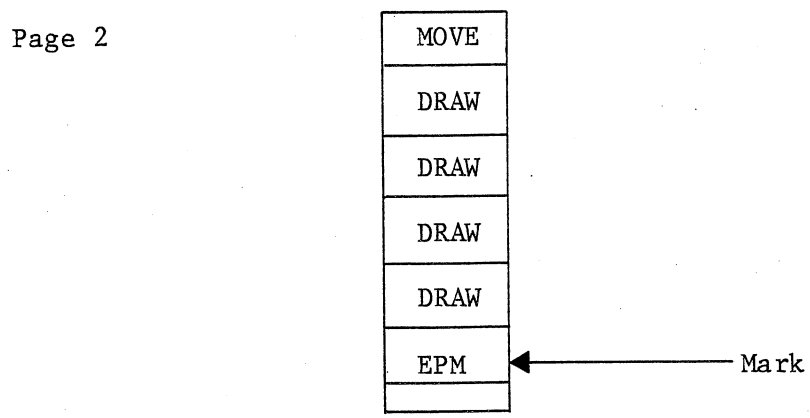


At this point the following is displayed on the monitor.

TEST ONE

The call to ADDREF (2) (line 110) opens up page 2 in the add mode. This call also sets up a GRAPHIC 8 memory address pointer to point to the first word (mark Ø) in page 2. It also saves the last mark value associated with page 1.

After the call to MOVE (line 120) and the 4 calls to DRAW (lines 130 to 160), page 2 looks as follows:

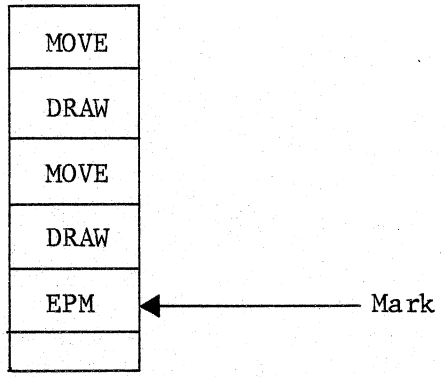


At this point nothing in page 2 is displayed on the monitor since a call to PICTUR has not been made. Page 1 is always displayed.

The call to ADDREF (3) (line 170) opens up page 3 in the add mode. This call also sets up a GRAPHIC 8 memory address pointer to point to the first word (mark Ø) in page 3. It also saves the last mark value associated with page 2.

After the calls to MOVE, DRAW, MOVE, DRAW (lines 180 to 210), page 3 looks as follows:

Page 3

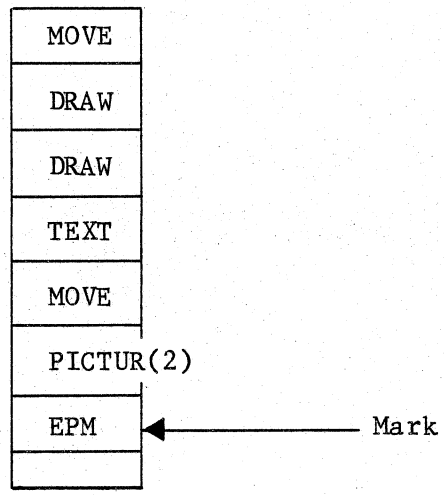


At this point nothing in page 3 is displayed on the monitor since a call to PICTUR has not been made.

The call to ADDREF (1) (line 220) re-opens page 1 in the add mode. This call also sets up a GRAPHIC 8 memory address pointer to point to the last word in page 1 that contains the EPM. It also saves the last mark value associated with page 3.

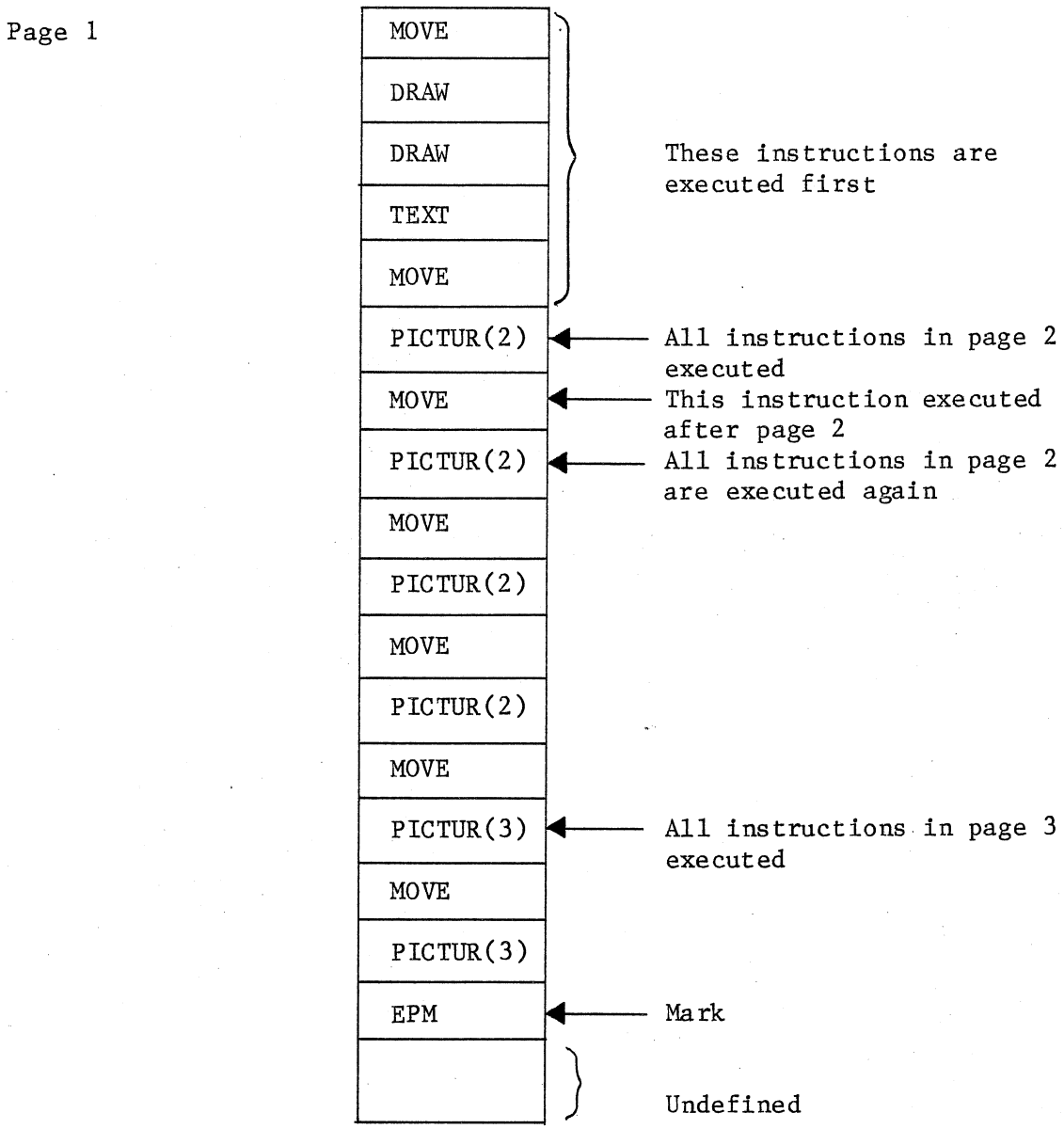
After the calls to MOVE and PICTUR (2) (lines 230 and 240), page 1 looks as follows:

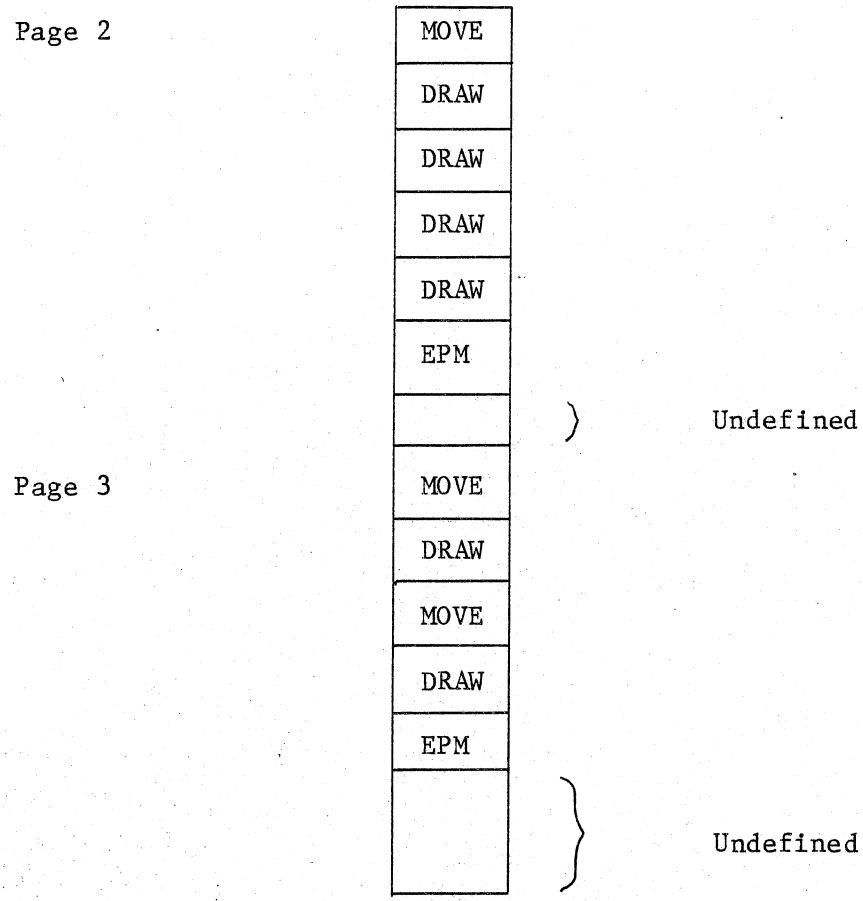
Page 1



At this point a box is displayed at the top left side of the monitor. The call to PICTUR (2) causes a subroutine call to be made to page 2. This causes the instructions in page 2 to be executed. When the EPM is executed in page 2, program control is returned back to page 1.

After the remaining statements in the FSP program (lines 250 to 340) are executed, the GRAPHIC 8 memory looks as follows:





At this point the image shown in figure 6-1 is displayed on the monitor.

NOTE

The page 1 instructions are the only instructions that are executed, as they are sent down, without a CALL PICTUR. All other pages require a CALL PICTUR in order to be displayed.

The UPDATE and ERASEP subroutines are normally used in response to some operator action. For example, the function keys on a keyboard could be programmed to cause certain modifications to a display image. To illustrate the use of UPDATE and ERASEP, let's say that it is now desired to perform the following actions in response to function key responses from an operator.

FUNCTION KEY	ACTION
16	Remove box display from 4 corners. (Effectively delete or erase the instructions stored in page 2.)
17	Replace the 'TEST ONE' characters with 'TEST TWO'.

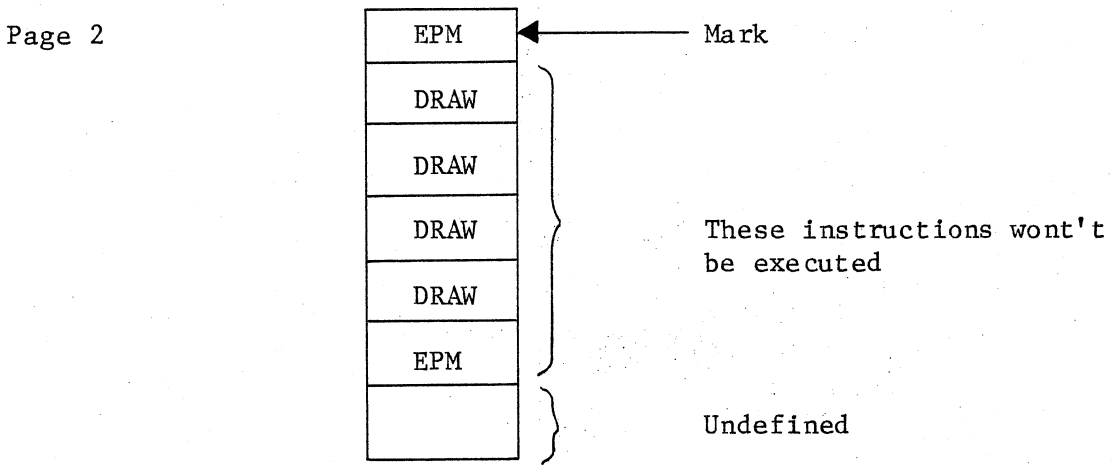
All operator inputs from the GRAPHIC 8 are returned via the EVENT subroutine. This subroutine is described in Section 7. To avoid confusion, let's say that the FSP program has been properly set up to detect function key responses.

When a function key 16 response is detected, the following FSP code could be used to erase page 2:

```
CALL UPDATE (2,0)
CALL ERASEP
```

The call to UPDATE sets up the GRAPHIC 8 address pointer to point to the first instruction in page 2. The call to ERASEP stores an EPM in page 2 which replaces the first instruction.

Based on the previous example, page 2 would look as follows:



At this point the boxes are no longer displayed at the four corners. In page 1 there are four CALL PICTUR (2) instructions; but every time page 2 is executed now, the first instruction executed in page 2 is an EPM so program control returns to page 1. (I.e., the four DRAWS and second EPM in page 2 will never get executed.)

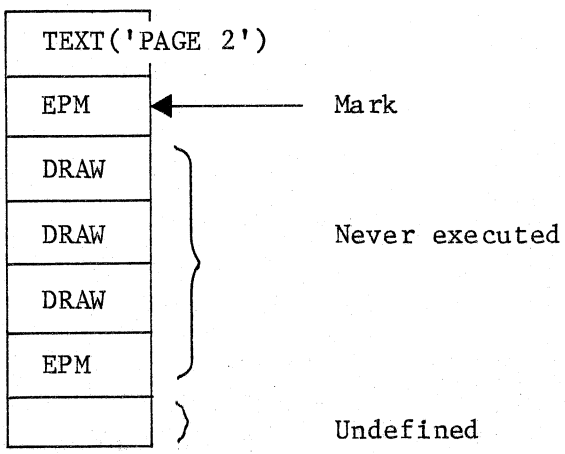
NOTE

When an ERASEP is executed, the page is also reopened. This has the same effect as executing another ADDREF(2). For example, if the FSP program were coded in the following way in response to a function key 16 response:

```
CALL UPDATE (2,0)
CALL ERASEP
CALL SETEXT ('PAGE 2',6)
```

then page 2 would look as follows after the TEXT instruction is executed:

Page 2



At this point the text will be displayed in place of the 4 small boxes (i.e., 'PAGE 2' will appear in the 4 corners of the image box).

In the previous example, if we want to replace the text 'TEST ONE' with 'TEST TWO', we must know where the SETEXT instruction is located in page 1. The following code would have to be added to the previous example to determine the location of the SETEXT instruction in page 1.

LINE NO.

```
95              CALL GETMRK (MARK)
```

The above code would be inserted between lines 90 and 100.

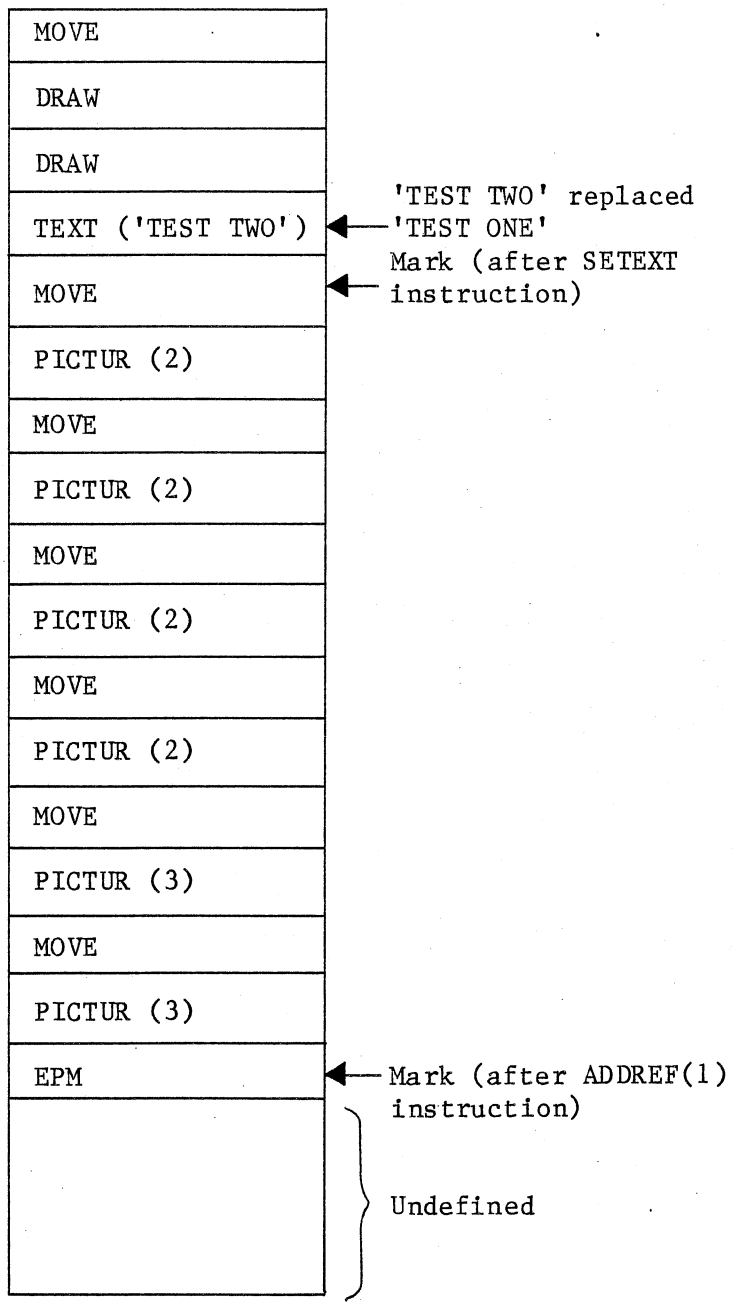
The CALL to GETMRK retrieves the current mark value from the event table. After the call to GETMRK, the variable MARK contains the mark value of the SETEXT instruction.

Now we are ready to process function key 17 type responses. For a function key 17 response, the following code would be added:

```
CALL UPDATE (1, MARK)

CALL SETEXT ('TEST TWO', 8)

CALL ADDREF (1)
```

The call to UPDATE sets up the address pointer to point to the address where the SETEXT instruction ('TEST ONE') is located. The call to SETEXT ('TEXT TWO') replaces the previous SETEXT instruction. At this point the CRT indicator would display 'TEST TWO'. When UPDATE is executed, edit mode is entered. In this mode, no EPM is inserted after the SETEXT instruction is added. After the SETEXT instruction is executed, page 1 looks as follows:

Page 1



Note that after the SETEXT instruction is replaced in GRAPHIC 8 memory, the mark is pointing to the MOVE following the SETEXT instruction. The call to ADDREF (1) takes us out of edit mode and into add mode. After the call to ADDREF(1), the mark is repositioned to the EPM. The call to ADDREF(1) is necessary so that all future FSP subroutine calls made for page 1 will get added to the end of page 1. If no call to ADDREF(1) is made, all future FSP subroutine calls made for page 1 would be added following the SETEXT instructions. In essence we would be destroying the refresh data in page 1.

When UPDATE is used, care must be used to ensure that the original refresh data is not destroyed. For example, when the 'TEST ONE' text was replaced, it was replaced with a TEXT string consisting of exactly 8 characters (i.e., the same number of characters as the original text string 'TEST ONE'). If a text string of more than 8 characters were inserted in place of the 'TEST ONE' text string, then these additional characters would be stored following the TEXT instruction. In this case the MOVE instruction would be destroyed. If the TEXT string were very large, the remaining instructions in page 1 could easily be over-written and destroyed. If the text string were less than 8 characters, then the text string would have to be space filled to a length of 8 characters.

The MOVEIM and COPYIM subroutines provide the user with a means of correcting this problem, but due to the complexity of these two routines it is advised that the user become familiar with writing FSP Programs before an attempt is made to use these routines.

6.1 OPEN PAGE FOR ADDING REFRESH DATA

NAME: ADDREF

FUNCTION: This routine opens the specified page and sets the mark to either the beginning of the page, if it is empty, or directly following the last data entered into the page.

CALLING FORMAT: CALL ADDREF (IPAGE)

DESCRIPTION OF PARAMETERS:

IPAGE = An integer variable containing the page number to be opened.

$1 \leq \text{IPAGE} \leq 255$

DETAILED DESCRIPTION:

This subroutine is used to set up a page for initial orders (if empty) or for addition of graphics orders to the page. This subroutine does not give the caller the ability to edit previous graphic orders as does the UPDATE subroutine (see next description).

6.2 OPEN PAGE FOR EDITING REFRESH DATA

NAME: UPDATE

FUNCTION: The requested page is opened for editing with the page mark set to the value supplied by the caller.

CALLING FORMAT: CALL UPDATE (IPAGE, MARK)

DESCRIPTION OF PARAMETERS:

IPAGE = An integer variable containing the page number to be opened.

$1 \leq$ IPAGE ≤ 255

MARK = An integer variable containing the position in the page where the mark is to be positioned.

DETAILED DESCRIPTION:

The current page and mark are set to IPAGE and MARK, and mode is changed to edit. Used to modify existing refresh. Note that in the edit mode it is possible to inadvertently insert refresh instructions, over and beyond the current page ending. This will cause an error to occur.

6.3 ERASE FROM PAGE MARK TO END OF PAGE

NAME: ERASEP

FUNCTION: This routine erases the currently open page from the current position of the mark to the end of the page. It does not change the mark.

CALLING FORMAT: CALL ERASEP

DETAILED DESCRIPTION:

Moves the end of page mark to current mark of the currently opened page.

6.4 GRAPHIC SUBROUTINE CALL

NAME: PICTUR

FUNCTION: Causes the contents of the specified page to be displayed at the current mark and position.

CALLING FORMAT: CALL PICTUR (IPAGE)

DESCRIPTION OF PARAMETERS:

IPAGE = An integer variable containing the page number to be displayed (that is, linked to).

DETAILED DESCRIPTION:

This routine causes the contents of page IPAGE to be called from the current mark and position. Note that the page calls should not be arranged so that a page can eventually call itself.

6.5 GET MARK REQUEST INFORMATION

NAME: GETMRK

CALLING FORMAT: CALL GETMRK (M)

DESCRIPTION OF PARAMETERS:

M = An integer variable returned to the caller containing the present page mark.

DETAILED DESCRIPTION:

This routine retrieves the current value of the mark and returns it to the calling program in the variable M.

6.6 MOVE A BLOCK OF GRAPHIC ORDERS

NAME: MOVEIM

FUNCTION: Allows the caller to move all data between a given position and the end of the current page to another mark position on that page. The current mark (end of page) will be changed. Current page mark can be obtained by calls to EVENT and GETMRK.

CALLING FORMAT: CALL MOVEIM (MARKFR, MARKTO)

DESCRIPTION OF PARAMETERS:

MARKFR = Integer variable indicating the mark location that data will be moved from.

MARKTO = Integer variable indicating the mark location that the data will be moved to.

The following condition must exist:

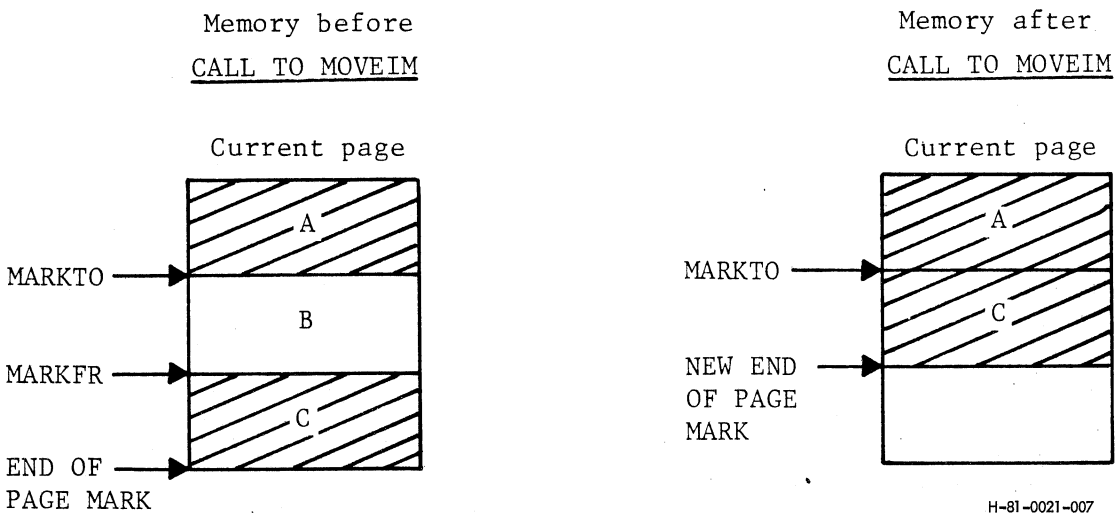
$$MARKTO < MARKFR \leq \text{END OF PAGE}$$

DETAILED DESCRIPTION:

This routine moves data between a specified mark location (MARKFR) and the current end of page to a given mark location (MARKTO). The current end of page will be changed to equal MARKTO plus the length of the data move (old end of page minus MARKFR).

This routine automatically sends the current mark (new end of page) back to the host. The user calls GETMRK to get the value. The user may use the value of the mark for subsequent updates.

In terms of GRAPHIC 8 memory, the current page is altered as shown below when a CALL MOVEIM (MARKFR, MARKTO) is executed.



The result of the MOVEIM operation is that section B has been deleted from refresh memory and additional refresh memory has been freed for re-use by the FSP programmer.

6.7 COPY A BLOCK OF GRAPHIC ORDERS

NAME: COPYIM

FUNCTION: Allows the user to copy the data between two given marks on the current page to the end of that page. The current mark (end of page) changes.

CALLING FORMAT: CALL COPYIM (MARKA, MARKB)

DESCRIPTION OF PARAMETERS:

MARKA = An integer variable specified by the caller which gives the starting mark location of the data being copied.

MARKB = An integer variable specified by the caller which gives the last mark location of the data being copied.

The following condition must exist:

MARKA < MARKB < END OF PAGE

Space must be available at the end of the page.

DETAILED DESCRIPTION:

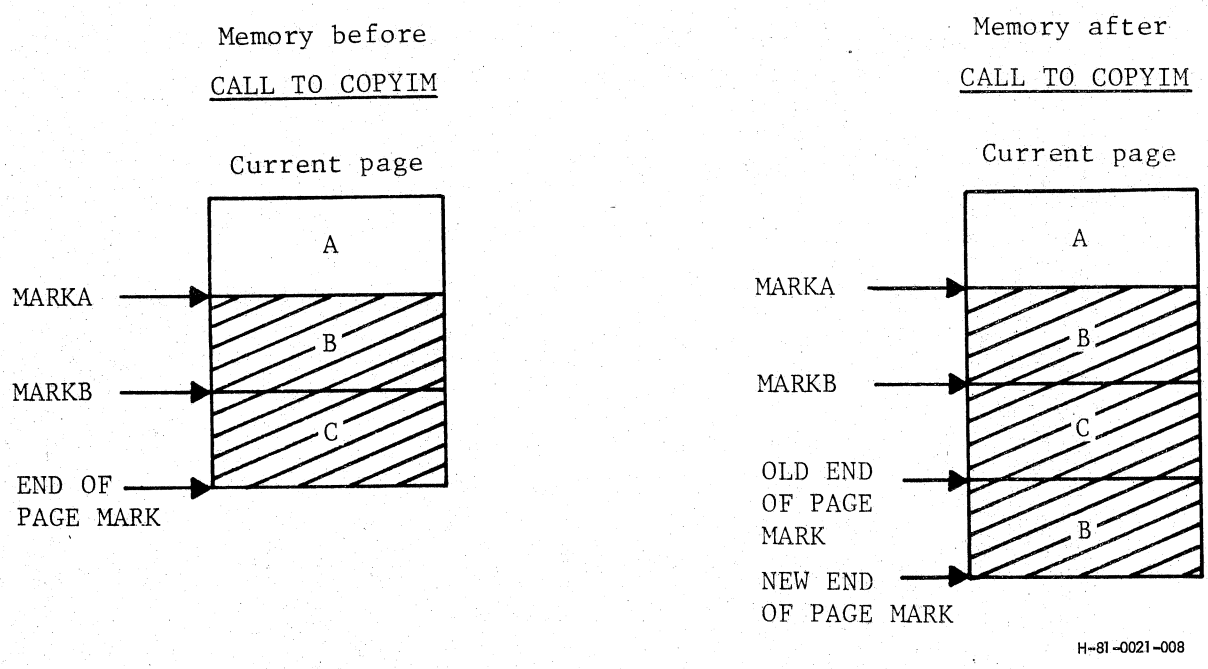
This routine copies data between two given marks on the current page to the end of the current page. The current end of page is modified to equal the end of page prior to the CALL COPYIM plus the length of the data copied.

This routine determines if room is available at the end of the page for the data to be copied. If not enough space is available, error 64 is issued and no data is copied.

This routine allows the user to expand an existing image by copying or appending the image to the end of the current page where additional FSP functions may be performed.

This routine automatically sends the current mark (new end of page) back to the host. The user calls GETMRK to ge the new mark value.

In terms of GRAPHIC 8 memory, the current page is altered as shown below when a CALL COPYIM (MARKA, MARKB) is executed.



The result of the COPYIM operation is that a copy of the refresh code in section B is appended to the end of the current page. After the COPYIM operation, the mark pointer changes to reflect the new end of page mark.

SECTION 7

EVENT ROUTINE

An asynchronous "event" will occur when one of the following operator actions takes place:

- One of the 16 function keys on the keyboard is pressed.
- One of the 16 matrix keys on the keyboard is pressed.
- When an alphanumeric key depression causes the "text input" buffer to become full.
- A CR (carriage return) key is pressed on the keyboard.
- The data tablet pen is pressed and released while in the "automatic" mode.

An asynchronous event will also occur with the "error" event which is generated when any of the error conditions described in Appendix C are detected.

The final asynchronous event is the "illegal response from terminal" event. This event occurs when FSP in the host receives a message from the GRAPHIC 8 which is incomprehensible.

A synchronous (predictable) "event" will occur as a result of the following FSP calls:

REQTB - Request current coordinate of a PED
REQIM - Request refresh data

The FSP programmer becomes aware that an "event" has occurred only by calling the EVENT subroutine described in paragraph 7.1. The EVENT subroutine returns an event code value to the caller which indicates either that no event has occurred or that one of the above asynchronous or synchronous events has occurred.

7.1 POLL TERMINAL FOR EVENT OR REQUEST RESPONSE

NAME: EVENT

FUNCTION: Wait for one of the previously described sync, or a sync events to occur. An event code is returned to the caller indicating the type of event (if any).

CALLING FORMAT: CALL EVENT (IEVNT)

DESCRIPTION OF PARAMETERS:

IEVNT = An integer variable returned to the caller indicating the event type.

1 = No events
2 = PED motion, call GETTB
3 = A line of text is available, call GETTXT
4 = A function key was pressed, call GETKEY
5 =
6 = } Used only by Graphic 7
7 =
8 = Error (XY overflow, halt, index out of range, etc.), call GETERR.
9 = Refresh dump available, call GETIM
10 =
11 = } Used only by Graphic 7 PED status
12 = Illegal response from the terminal

SECTION 8

PERIPHERAL DEVICE ROUTINES

FSP allows and supports up to four keyboards and four positional entry devices (PEDs). The keyboards combine a 32 function keyboard and an alphanumeric keyboard into one physical unit. PEDs which are devices used to enter X and Y coordinate information include trackballs, joysticks and data tablets.

This section is divided into two groups of subroutines:

1. Keyboard Subroutines

2. PED Subroutines

8.1 KEYBOARD ROUTINES

This section describes the routines available to the user to handle alphanumeric and function keyboard data. These routines are named below:

- ENBPAD - Enable alphanumeric scratchpad
- DSAPAD - Disable alphanumeric scratchpad
- GETTXT - Retrieve alphanumeric text information
- GETKEY - Retrieve function key information

Each keyboard has an 86 character buffer or pad associated with it which receives alphanumeric key characters as they are typed. The pad is displayable as a single line of text at a user specified X, Y coordinate.

When scratchpad is enabled a "text" event is created (event code = 3) when one of the following operator actions take place:

- An alphanumeric key depression causes the associated pad to become full.
- A "return" key is pressed and at least one character is already in the pad.

Once a "text" event is created, the pad is cleared (reset to blanks).

A "key" event (event code = 4) is created when the operator presses any of the 16 function keys or any of the 16 matrix keys.

8.1.1 ENABLE ALPHANUMERIC SCRATCHPAD

NAME: ENBPAD

FUNCTION: Specifies parameters for the receipt and display of scratchpad information.

CALLING FORMAT: CALL ENBPAD (IKEY, IDISP, X, Y, IMAX)

DESCRIPTION OF PARAMETERS:

IKEY = (1 through 4) keyboard number specified by the caller.

IDISP = (0 through 15) monitors on which the caller wishes alphanumeric information displayed. See ENBBOX routine for associated values.

X = Real variable specified by the caller indicating the X position in user coordinates of first character.

Y = Real variable specified by the caller indicating the Y position in user coordinates of first character.

IMAX = (1 through 86) user specified number of characters allowed in pad (default is 1).

DETAILED DESCRIPTION:

This routine allows the user to establish monitors, location, keyboard, and maximum number of characters for each scratchpad area. Note that the keyboard is always enabled. ENBPAD only displays the pad and establishes its parameters. The user may use the keyboard without enabling the pad. If the user does not call ENBPAD, the default IMAX of 1 is used, which means that every alphanumeric key depression causes a text event (event code = 3). (See GETTXT (section 8.1.3) for further information.)

8.1.2 DISABLE ALPHANUMERIC SCRATCHPAD

NAME: DSAPAD

FUNCTION: Turns off the display of the alphanumeric scratchpad area.

CALLING FORMAT: CALL DSAPAD (IKEY)

DESCRIPTION OF PARAMETERS:

IKEY = The keyboard number 1 through 4, specified by the caller.

DETAILED DESCRIPTION:

Turns off the display only for the selected keyboard. Does not change any other parameters. The user may still use the keyboard and events are still generated as explained in GETTXT. DSAPAD simply causes the keyboard information not to appear on the displays.

8.1.3 GET TEXT EVENT INFORMATION

NAME: GETTXT

FUNCTION: Transfers the text buffer characters obtained by the EVENT subroutine to the caller.

CALLING FORMAT: CALL GETTXT (IARRAY, ISIZE, NCHAR, KBD)

DESCRIPTION OF PARAMETERS:

IARRAY = A user defined integer array into which the currently completed text input buffer is transferred. Each array element contains one 7-bit ASCII character in the rightmost 7 bits of the element. The eighth bit is always set to a 1.

ISIZE = An integer variable supplied by the caller containing the maximum number of characters to be placed in the array, i.e., ISIZE is the size of the array. If an input buffer string is longer than the array size, those characters which don't fit are lost.

NCHAR = An integer variable returned to the caller, containing the number of characters (elements) placed in the array. The carriage return character used to terminate the input buffer is not included in the array or character count.

KBD = An integer variable returned to the caller indicating the keyboard to which the input buffer is associated.

KBD = 1 = keyboard #1
KBD = 2 = keyboard #2
KBD = 3 = keyboard #3
KBD = 4 = keyboard #4

DETAILED DESCRIPTION:

This subroutine should normally be called only after a call to EVENT has indicated that a "text" event has occurred.

Associated with each of the four alphanumeric keyboards in the terminal is an 86-character input buffer (there are four such buffers). As a key is pressed on the keyboard, its corresponding character is added to the next character position in its corresponding input buffer. If a typing error is made while entering characters into the scratchpad, the RUBOUT key can be used to make corrections. RUBOUT deletes the last character typed. Successively pressing RUBOUT can delete the entire line.

The scratchpad characters are sent to the host and the input buffer is cleared (reset to blanks) when one of the following events occurs:

1. A carriage return key is typed. If the scratchpad is empty and a carriage return is entered, no event flag is sent back to the host.

2. The buffer is full.

See ENBPAD for further information.

8.1.4 GET FUNCTION KEY EVENT INFORMATION

NAME: GETKEY

FUNCTION: Returns data pertaining to the keyboard and function key pressed.

CALLING FORMAT: CALL GETKEY (KBD, KEY)

DESCRIPTION OF PARAMETERS:

KBD = An integer variable returned to the caller indicating which of the four keyboards caused the event.

KEY = An integer variable returned to the caller indicating which of the 32 function keys was pressed.

DETAILED DESCRIPTION:

This routine retrieves the key number after a function key event has been received. Function keys are always enabled.

FUNCTION KEYS:

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MATRIX KEYS:

7	8	9	15
4	5	6	14
1	2	3	13
10	0	11	12

8.2 TRACKBALL/FORCESTICK/DATA TABLET ROUTINES

This section describes how to program the trackball or forcestick or data tablet.

The following routines are described in this section:

ENBPED - Attach user symbol to PED

DSAPED - Detach user symbol from PED

ENBCUR - Enable hardware or software cursor

DSACUR - Disable hardware or software cursor

REQTB - Request PED X, Y

GETTB - Get PED request information

FSP, when initialized (INIT call), sets the X, Y position of the PED to the center of the screen.

A PED Event is created when one of the following operator actions or program actions take place:

- REQTB subroutine is called (synchronous event)
- The data tablet pen is pressed and released while in the automatic mode (asynchronous event)

The information available as a result of a PED event is as follows:

- PED causing the event (1, 2, 3, or 4)
- X, Y coordinate of the PED in user coordinates

The subroutine GETTB is used to obtain the above information once the event has occurred.

An additional PED feature is provided by the ENBPED and ENBCUR subroutines which lock a visual cursor to the movement of the PED; i.e., as the PED is moved, so is the visual cursor.

The following operator and/or program actions are required to use a PED:

1. Enable PED and link PED movement to visual cursor (ENBPED, ENBCUR).
2. Move PED to desired X, Y coordinate (operator action). Visual cursor follows PED movement.
3. Request and get PED X, Y position (REQTB, EVENT, GETTB) or, if PED is a data tablet in the automatic mode.
4. Get last PED position (EVENT, GETTB).

8.2.1 ENABLE PED

NAME: ENBPED

FUNCTION: This routine allows the user to attach a user defined symbol to a specified PED and define the PED as a trackball/forcestick or a data tablet.

CALLING FORMAT: CALL ENBPED (IDVNM, IDVTP, IPAGE, IMARK)

IDVNM = Integer Variable between 1 and 4 corresponding to Device Number.

1 - PED1	3 - PED3
2 - PED2	4 - PED4

IDVTP = Integer Variable defining the PED device being used.

0 - Trackball/Forcestick
1 - Data Tablet
2 - Data Tablet (Return position on switch release)

IPAGE = Integer variable containing the number of the page that the refresh instructions for the user defined cursor resides.

IMARK = Integer variable containing the mark value of the refresh instructions for the user defined cursor.

DETAILED DESCRIPTION:

ENBPED allows the user to attach a PED to previously stored refresh instruction on a specified page at a specified mark. Any future updating of the PED position will be directed to this symbol rather than any one of the default symbols.

8.2.2 DISABLE PED

NAME: DSAPED

FUNCTION: DSAPED is used to disable a PED device from a user defined symbol.

CALLING FORMAT: CALL DSAPED (IDUNM)

PARAMETER DESCRIPTION:

IDUNM = Integer Variable containing the device number (1-4) of the PED device to be disabled.

1 - PED1	3 - PED3
2 - PED2	4 - PED4

DETAILED DESCRIPTION:

DSAPED will disable a PED device from a user defined symbol.

8.2.3 ENABLE CURSOR

NAME: ENBCUR

FUNCTION: This routine will enable either the hardware or software default cursor, attach the specified PED to the cursor, display the cursor on the specified monitor (or monitors) and define the PED as a Trackball/Forcestick or Data Tablet.

CALLING FORMAT: CALL ENBCUR (IDVNM, IDISPL, IDVTP, ICUR)

DESCRIPTION OF PARAMETERS:

IDVNM = Integer variable between 1 and 4 corresponding to Device Number

1 - PED1	3 - PED3
2 - PED2	4 - PED4

IDISPL = Integer variable between 1 and 15 corresponding to the monitor (or monitors) that the cursor will be displayed and updated on. This argument is ignored if hardware cursor is being enabled (see status routine for acceptable values).

IDVTP = Integer variable defining the PED device being used.

0 - Trackball/Forcestick
1 - Data Tablet
2 - Data Tablet (Return position on switch release)

ICUR = Integer variable defining the default cursor to be used.

0 - Default Software cursor
1 - Default Hardware cursor

DETAILED DESCRIPTION:

ENBCUR will enable a cursor to be displayed on the specified monitor (or monitors) and attach a PED device to the cursor. This routine must be called if the user wishes to use one of the default cursors. If the hardware cursor is used, a cross hair display will be placed on the specified monitors. If the software cursor is used, the symbol will be *n where n is the number of the cursor.

When using a Trackball/Forcestick the user will always be required to use the GETXY Routine to obtain the current X-Y location of the symbol. This X-Y location will correspond to the position of the asterisk and not the numeral.

When using a DATA TABLET the user has a choice between operating mode. If a device type of 1 is chosen then the operation of the data tablet will be the same as the Trackball/Forcestick; but, when the device type of 2 is used then the current X-Y location of the specified symbol will be sent back to the host each time the switch on the pen is released.

8.2.4 DISABLE CURSOR

NAME: DSACUR

FUNCTION: This routine will disable a PED device from one of the default cursors.

CALLING SEQUENCE: CALL DSACUR (IDVNM)

DESCRIPTION OF PARAMETERS:

IDVNM = Integer variable corresponding to the device numbers.

1 - PED1	3 - PED3
2 - PED2	4 - PED4

DETAILED DESCRIPTION:

DSACUR will be called after an ENBCUR has been called and will disable the PED symbol that is attached to the specified PED device, by removing the symbol instructions from the refresh.

8.2.5 REQUEST PED X,Y

NAME: REQTB

FUNCTION: Requests the current X, Y coordinate of the specified PED.

CALLING FORMAT: CALL REQTB (NUMBER)

DESCRIPTION OF PARAMETERS:

NUMBER = Integer variable supplied by the caller indicating the PED selected.

1 - PED1	3 - PED3
2 - PED2	4 - PED4

DETAILED DESCRIPTION:

This routine causes a PED event to occur which has an event code of 2. The PED event contains the current X,Y position of the PED as indicated by the current cursor position on the screen. This call should not be used if the PED is a data tablet in the automatic mode.

NOTE

This call only causes a PED event to occur. The actual X, Y position of the PED is obtained by a combination of the EVENT and GETTB subroutines.

8.2.6 GET PED REQUEST INFORMATION

NAME: GETTB

FUNCTION: Retrieves the current X, Y coordinate of the PED.

CALLING FORMAT: CALL GETTB (NUMBER, X, Y)

NUMBER = An integer variable returned to the caller identifying the PED which caused the event.

X, Y = Real variables containing the current location of the PED in user coordinates.

DETAILED DESCRIPTION:

This routine is called after the EVENT subroutine has returned an event code of 2 (PED event).

Example 1:
```
C  Read current PED #1 position
       CALL REQTB (1)
10     CALL EVENT (IEVNT)
       IF (IEVNT .NE. 2) GO TO 10
       CALL GETTB (NUMBER, X, Y)
       IF (NUMBER .NE. 1) GO TO 10
```

Example 2:
```
C  Read current PED #2 position where PED #2
C  is a data tablet in the automatic mode
20     CALL EVENT (IEVNT)
       IF (IEVNT .NE. 2) TO GO 20
       CALL GETTB (NUMBER, X, Y)
       IF (NUMBER .NE. 2) GO TO 20
```

8.2.7 PED PROGRAMMING EXAMPLES

Example 1:
```
C Initialize forcestick as PED #1 using
C default software cursor on display #1

      CALL INIT
      CALL LAYOUT
      CALL ENBCUR (1, 8, 0, 0)

C AT this point in the program an "*1" appears
C on display #1 in the center of the screen and
C follows the operators actions on the
C forcestick.  No PED events are generated,
C however, until the program issues a REQTB call
```

Example 2:

```
C Initializes data tablet as PED #2, in automatic
C mode using default hardware cursor on display #1

                CALL INIT
                CALL LAYOUT
                CALL ENBCUR (2, 8, 2, 1)

C At this point the hardware cursor (cross hair)
C will be displayed on monitor #1 and will
C be controlled by PED #2 (Data Tablet)

C At this point in the program the operator
C may move the cross hairs only by moving the Data
C Tablet pen with the switch pressed.  Once the
C switch is released, a PED event is created
C which is detected by the EVENT call and
C processed by the GETTB call - i.e., no REQTB is
C required in this mode.  Multiple PED events are
C possible in the automatic mode simply by pointing
C to a position on the tablet and pressing
C and releasing the switch.
```

Example 3:

```
C Initialize trackball as PED #2 using
C a user defined cursor
C
C Create a single page 50 words in length

                CALL INIT (1,0,1)
                CALL LAYOUT (1,50)

C Define the cursor to be the letter "A"

                CALL MOVE (512.,512.,0)
                CALL SETEXT ('A',1)

C Note:  The mark associated with the above absolute
C        MOVE call is = 0 since nothing
C        else has been placed in the page.
C        Also the "A" is displayed at the center
C        of the screen but not under trackball control.

C Link trackball to user defined cursor

                CALL ENBPED (2,0,1,0)

C At this point in the program the "A" is linked
C to the trackball and is moveable.
```

SECTION 9

PACKED VECTOR MODE

The following subroutines are described in this section:

ENBPMD - Enable packed vector mode

PMOVE - Packed vector move

PDRAW - Packed vector draw

DSAPMD - Disable packed vector mode

Packed vector mode is primarily intended for serial interface users. Using packed vector mode can result in a 4:1 speed increase when inserting absolute move and absolute draws into refresh.

The packed vector mode feature is most useful when large amounts of X, Y move and draw data are being created over the serial interface. Packed vector mode can also be used on parallel interface systems but it is strongly recommended that packed vector messages not be used on parallel systems. No FSP internal ASCII code conversions are required for parallel transmissions and the use of packed vector messages on parallel systems will result in a decrease in speed due to the FORTRAN overhead involved in processing packed moves and draws.

Calls to PMOVE and PDRAW data create packed vector data in an output buffer. When the buffer is filled, the data in the buffer is sent to GCP automatically. An important function of DSAPMD is to ensure that no residual data is lost by sending the contents of the output buffer to GCP.

NOTE

Once packed vector mode is enabled by calling ENBPMD, the only calls allowed to FSP are to PMOVE, PDRAW, and DSAPMD.

Sample user program segment:

```
C
C        REFRESH CODE FOR DRAWING BARRED BOX
C
      X = XCOORD
      Y = YCOORD
      XRIGHT = XCOORD + 128.
C
C     ENABLE PACKED VECTOR MODE
C
      CALL ENBPMD
C
      DO 40 M = 1, 64
      Y = Y + 2.
      CALL PMOVE (X, Y)
      CALL PDRAW (XRIGHT, Y)
   40 CONTINUE
C
C     DISABLE PACKED VECTOR MODE
C
      CALL DSAPMD
C
```

9.1 ENABLE PACKED VECTOR MODE

NAME: ENBPMD

FUNCTION: This routine enables the packed vector mode.

CALLING FORMAT: CALL ENBPMD

DESCRIPTION OF PARAMETERS: None

DETAILED DESCRIPTION:

This routine enables the packed vector mode, allowing the user to send graphic absolute move and draw commands in packed mode. ENBPMD must be called before the PMOVE, PDRAW, and DSAPMD routines may be called. Once packed vector mode is enabled, any number of calls to PMOVE and PDRAW and one call to DSAPMD are allowed. No other FSP subroutines may be called while in packed vector mode. A call to DSAPMD is required to disable packed vector mode.

9.2 PACKED VECTOR MOVE

NAME: PMOVE

FUNCTION: Allows the caller to move to a desired absolute X, Y position in user coordinates while in packed vector mode.

CALLING SEQUENCE: CALL PMOVE (X, Y)

DESCRIPTION OF PARAMETERS:

X, Y = Real variables specified by the caller indicating the absolute X, Y user coordinate is to be moved. (Note: X and Y must be in the range specified in the user's call to SCALE.)

DETAILED DESCRIPTION:

PMOVE operates similarly to the FSP subroutine MOVE in absolute mode 0 (paragraph 5.1). The current position is moved to the X, Y coordinate specified by the caller. The X, Y coordinate then becomes the current position.

PMOVE converts the absolute X, Y user coordinate into a display coordinate, formats (packs) the X, Y data for transfer to GCP, and causes an absolute move graphic order to be inserted at the mark position of the currently opened refresh page. PMOVE may be called only when packed vector mode is enabled (see ENBPMD).

9.3 PACKED VECTOR DRAW

NAME: PDRAW

FUNCTION: Allows the user to draw a line (vector) from the current position to the absolute X, Y position in user coordinates while in packed vector mode.

CALLING FORMAT: CALL PDRAW (X, Y)

DESCRIPTION OF PARAMETERS:

X, Y = Real variables specified by the caller indicating the absolute X, Y position in user coordinates while in packed vector mode.

DETAILED DESCRIPTION:

PDRAW operates similarly to the FSP subroutine DRAW in absolute mode 0 (paragraph 5.2). A line is drawn from the current position to the X, Y coordinate specified by the caller. The X, Y coordinate then becomes the current position.

PDRAW converts the absolute X, Y user coordinate into a display coordinate, formats (packs) the X, Y data for transfer to GCP, and causes an absolute draw graphic order to be inserted at the mark position of the currently opened page. PDRAW may be called only when packed vector mode is enabled (see ENBPMD).

9.4 DISABLE PACKED VECTOR MODE

NAME: DSAPMD

FUNCTION: Disables packed vector mode by changing the FSP operating mode from packed vector mode back to standard FSP call mode.

CALLING FORMAT: CALL DSAPMD

DESCRIPTION OF PARAMETERS: None

DETAILED DESCRIPTION:

Once packed vector mode has been enabled by a call to ENBPMD, a call to DSAPMD must be made before calls to any other non-packed vector mode routines.

This routine also ensures that residual packed vector data will be sent (see introduction to this section).

SECTION 10

TWO DIMENSIONAL SCALE, ROTATE AND TRANSLATE ROUTINES

This section describes the routines available to the user for applications involving the 2D Coordinate Converter.

- CC2DBL — Create 2D Converter Block
- MOVE2D (X, Y, MODE) — Create 2D Move Graphic Order
- DRAW2D (X, Y, MODE) — Create 2D Draw Graphic Order
- T2D2D (IGRAPH, IPAG2D) — Transform Graphic Page to 2D Page
- MTRX2D (ARRAY) — Update 2D Composite Matrix in CC2DBL
- V2DBOX (LV, RV, BV, TV) — Update Viewbox in CC2DBL

10.1 INITIALIZE 2D VIEWBOX AND 2D MATRIX

NAME: CC2DBL

FUNCTION: Generates coordinate converter instructions to initialize the viewbox boundaries and the matrix parameters.

CALLING FORMAT: CALL CC2DBL

DESCRIPTION OF PARAMETERS: NONE

DETAILED DESCRIPTION:

This routine generates two Coordinate Converter instructions and places them at the beginning of the currently opened page. The first instruction generated (LBOX) initializes the viewbox boundaries, which are used in clipping. The viewbox parameters are initialized as follows:

Parameter	Value Set to by CC2DBL
Viewbox left (Minimum X)	-512.
Viewbox bottom (Minimum Y)	-512.
Viewbox right (Maximum X)	511.
Viewbox top (Maximum Y)	511.

The second instruction generated (LMTX) initializes the Matrix parameters which are used in the transformation process. All Matrix elements except the scale factors are initialized to Ø. The scale factors, Matrix elements (1,1) and (2,2) are set to 256. This is equivalent to 1/64 in the fractional two's complement notation (see section 7.2) and is the default scale factor. The combination of the LBOX and LMTX instructions at the beginning of a page is referred to as the CCBLK of the page.

CC2DBL Format

WORD #	COMMAND	FORMAT	DESCRIPTION
0	LBOX	LBOX	LOAD VIEWBOX PARAMETERS
1		LV	LEFT BOUNDARY
2		BV	BOTTOM BOUNDARY
3		RV	RIGHT BOUNDARY
4		TV	TOP BOUNDARY
5	LMTX	LMTX	LOAD MATRIX PARAMETERS
6		M11	MATRIX ELEMENTS
7		M12	
8		M21	
9		M22	
10		M31	
11		M32	

10.2 CREATE 2D MOVE GRAPHIC ORDER

NAME: MOVE2D

FUNCTION: Generates either an absolute or relative 2D move graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL MOVE2D (X, Y, MODE)

DESCRIPTION OF PARAMETERS:

X, Y = Real variables supplied by the user which specify the 2D coordinate of the desired position. The coordinate is specified in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic instruction to be generated. When MODE=0 an absolute MOVE is implied and when MODE=1 a relative MOVE is implied. When MODE=0, the coordinate (X, Y) specifies an absolute 2D coordinate. When MODE=1, the coordinate (X, Y) specifies an offset to be moved from the current position.

DETAILED DESCRIPTON:

This routine converts the coordinate values specified to absolute screen coordinates and generates either an absolute or relative 2D move graphic order. This graphic order is placed at the mark position of the currently opened page. Note that relative moves are restricted to 1/2 of the screen size.

10.3 CREATE 2D DRAW GRAPHIC ORDER

NAME: DRAW2D

FUNCTION: Generates either an absolute or relative 2D draw graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL DRAW3D (X, Y MODE)

DESCRIPTION OF PARAMETERS:

X,Y = Real variables supplied by the user which specify the 2D coordinate of the end point of a line to be drawn. The coordinate is in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic instruction to be generated. When MODE=0, an absolute DRAW is implied and when MODE=1, a relative DRAW is implied. When MODE=0, the coordinate (X, Y) specifies the absolute coordinate of the end point of a line to be drawn. When MODE=1, the coordinate specifies the offsets to be used in drawing a line from the current position to a new position.

DETAILED DESCRIPTION:

This routine converts the coordinate values specified to absolute screen coordinates and generates either an absolute or relative 2D draw graphic order. This graphic order is placed at the mark position of the currently opened page. Note that relative draws are restricted to 1/2 of the screen size.

10.4 TRANSFORM 2D to 2D

NAME: T2D2D

FUNCTION: Transform a page of graphic orders into a page which consists entirely of 2D graphic orders.

CALLING FORMAT: CALL T2D2D (IGRAPH, IPAG2D)

DESCRIPTION OF PARAMETERS:

IGRAPH = An integer variable supplied by the user which specifies the number of the graphic page which is to be transformed. IGRAPH must be in the range:

$1 < IGRAPH < 256, IGRAPH \neq IPAG2D$

IPAG2D = An integer variable supplied by the user which specifies the number of the 2D page in which the transformed graphic orders are to be placed. IPAG2D must be in the range:

$1 < IPAG2D < 256, IPAG2D \neq IGRAPH$

DETAILED DESCRIPTION:

This routine sets up the Coordinate Converter to perform a graphic page to 2D transformation on the page of graphic instructions specified by IGRAPH. The Coordinate Converter is started and a block of transformed graphic instructions is output to the page specified by IPAG2D. Note that unpredictable results will occur if the 2D output file is not large enough to accommodate the transformed graphic instructions. Refer to Appendix D of the Graphic 7 Fortran Support Package (FSP) User's Manual to determine the necessary output file size.

10.5 COMPUTE AND REPLACE MATRIX PARAMETERS

NAME: MTRX2D

FUNCTION: Compute matrix parameters and generate coordinate converter instruction to update the matrix parameters.

CALLING FORMAT: CALL MTRX2D (ARRAY)

DESCRIPTION OF PARAMETERS:

ARRAY = A seven element real array supplied by the user which specifies the scaling, translation, and rotation factors necessary for computing the matrix parameters. The array parameters and their valid ranges are specified below:

ARRAY Element	Definition	Range
1	X-Pre Translation	$-(XU-XL) \leq XPRE \leq +(XU-XL)$
2	Y-Pre Translation	$-(YU-YL) \leq YPRE \leq +(YU-YL)$
3	X Scale Factor	$1/256 \leq XSC < 128$
4	Y Scale Factor	$1/256 \leq YSC < 128$
5	Z Rotation (X-Y Plane)	any real number (in radians)
6	X-Post Translation	$-(XU-XL) \leq XPOST \leq +(XU-XL)$
7	Y-Post Translation	$-(YU-YL) \leq YPOST \leq +(YU-YL)$

The variables used above are defined in the SCALE subroutine description.

e.g. CALL SCALE (XL, YL, XU, YU)

DETAILED DESCRIPTION:

This routine uses the array parameters passed to compute new matrix parameters. These parameters are computed as indicated in Appendix D and are entered into the CC2DBL of the currently opened page, replacing the previous matrix parameters. This routine must be called each time the user wishes to change translation, scaling, or rotation factors.

10.6 UPDATE VIEWBOX

NAME: V2DBOX

FUNCTION: Update viewbox boundaries in the currently opened page.

CALLING FORMAT: CALL VIEWBX (LV, RV, BV, TV)

DESCRIPTION OF PARAMETERS:

LV = Real variable supplied by the user which specifies the viewbox left boundary (minimum X). The valid range for LV is: XL < LV < RV.

RV = Real variable supplied by the user which specifies the viewbox right boundary (maximum X). The valid range for RV is: LV < RV < XU.

BV = Real variable supplied by the user which specifies the viewbox bottom boundary (minimum Y). The valid range for BV is: YL < BV < TV.

TV = Real variable supplied by the user which specifies the viewbox top boundary (maximum Y). The valid range for TV is: BV < TV < YU.

The parameters LV, RV, BV and TV are all specified in the user coordinate system.

DETAILED DESCRIPTION:

This routine updates the viewbox boundaries in the currently opened page. The CC2DBL viewbox parameters, generated by a previous call to the CC2DBL routine, are updated. This routine must be called in order to change the viewbox.

SECTION 11

THREE DIMENSIONAL SCALE, ROTATE AND TRANSLATE ROUTINES

This section describes the routines available to the user for applications involving the 2D/3D Coordinate Converter.

- INIT3D - Initialize 3D System
- SCAL3D - Define coordinate system
- CCBLK - Create 3D coordinate converter block
- MOVE3D - Create 3D move graphic order
- DRAW3D - Create 3D draw graphic order
- T3D2D - Transform 3D page into 2D page
- MTRX3D - Update composite matrix in CCBLK
- VIEWPT - Update view point in CCBLK
- VIEWBX - Update viewbox parameters in CCBLK

The remaining pages describe each subroutine in detail.

11.1 INITIALIZE 3D

NAME: INIT3D

FUNCTION: Initialize FSP variables for 3D Coordinate Converter support.

CALLING FORMAT: CALL INIT3D

DESCRIPTION OF PARAMETERS: None

DETAILED DESCRIPTION:

This routine sets default values for the Z-axis user coordinates. The default lower boundary is 0 and the default higher boundary is 32767.

11.2 DEFINE Z COORDINATE SYSTEM

NAME: SCAL3D

FUNCTION: Set the user coordinate values for the Z-axis user coordinates (third coordinate).

CALLING FORMAT: CALL SCAL3D (ZL, ZU)

DESCRIPTION OF PARAMETERS:

ZL = Real variable supplied by the user which specifies the value to be assigned to the lower boundary of the Z-axis in the user coordinate system. Note that ZL is coincident with the screen surface.

ZU = Real variable supplied by the user which specifies the value to be assigned to the upper boundary of the Z-axis in the user coordinate system.

DETAILED DESCRIPTION:

This routine sets the Z-axis user coordinates (third coordinate) to the values passed (ZL and ZU). This allows the caller to define the Z-axis near and far coordinates in real numbers. The 3D move and draw subroutines convert a coordinate in real numbers to an integer display coordinate. This conversion process is based upon the values of ZL and ZU. Without a call to SCAL3D, the Z-axis of the user coordinate system is equal to the default Z-axis coordinates, i.e., ZL=0 and ZU=32767. Note that the Z-axis is defined within a left-handed coordinate system. ZL is the Z-axis point that corresponds to the screen and ZU is the Z-axis point that is the furthest from the screen extending into the screen. The value of ZU must be greater than the value of ZL or unpredictable results will occur.

11.3 INITIALIZE VIEWBOX, VIEWPOINT, AND MATRIX

NAME: CCBLK

FUNCTION: Generate Coordinate Converter instructions to initialize the viewbox boundaries, the viewpoint and the matrix parameters.

CALLING FORMAT: CALL CCBLK

DESCRIPTION OF PARAMETERS: None

DETAILED DESCRIPTION:

This routine generates two Coordinate Converter instructions and places them at the beginning of the currently opened page. The first instruction generated (LBOX) initializes the viewbox boundaries, which are used in clipping, and the viewpoint which is used in generating perspective. The viewbox and viewpoint parameters are initialized as follows:

Parameter	Value Set to by CCBLK
Viewbox left (Minimum X)	-512.
Viewbox bottom (Minimum Y)	-512.
Viewbox near (Minimum Z)	0.
Viewbox right (Maximum X)	511.
Viewbox top (Maximum Y)	511.
Viewbox far (Maximum 2)	32767.
X Viewpoint	0.
Y Viewpoint	0.
Z Viewpoint	-32767.

The second instruction generated (LMTX) initializes the matrix parameters which are used in the coordinate transformation process. All matrix elements except the scale factors are initialized to 0. The scale factors, matrix elements (1,1), (2,2), and (3,3) are set to 256. This is equivalent to 1/64 in the fractional two's complement notation and is the default scale factor. The combination of the LBOX and LMTX instructions at the beginning of a page is referred to as the CCBLK of the page.

CCBLK FORMAT

WORD #	COMMAND	FORMAT	DESCRIPTION
0	LBOX	LBOX	Load viewbox parameters
1		LV	Left Boundary
2		BV	Bottom Boundary
3		NV	Near Boundary
4		RV	Right Boundary
5		TV	Top Boundary
6		FV	Far Boundary
7		Xa	X - Eye Point
8		Ya	Y - Eye Point
9		Za	Z - Eye Point
10	LMTX	LMTX	Load matrix parameters
11		M11	Matrix Elements
12		M12	
13		M13	
14		M21	
15		M22	
16		M23	
17		M31	
18		M32	
19		M33	
20		M41	
21		M42	
22		M43	

11.4 CREATE 3D MOVE GRAPHIC ORDER

NAME: MOVE3D

FUNCTION: Generates either an absolute or relative 3D move graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL MOVE3D (X,Y,Z,MODE)

DESCRIPTION OF PARAMETERS:

X,Y,Z = Real variables supplied by the user which specify the 3D coordinate of the desired position. The coordinate is specified in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic instruction to be generated. When MODE=0 an absolute MOVE is implied and when MODE=1 a relative MOVE is implied. When MODE=0, the coordinate (X,Y,Z) specifies an absolute 3D coordinate. When MODE=1, the coordinate (X,Y,Z) specifies an offset to be moved from the current position.

DETAILED DESCRIPTION:

This routine converts the coordinate values specified to absolute screen coordinates and generates either an absolute or relative 3D move graphic order. This graphic order is placed at the mark position of the currently opened page. Note that relative moves are restricted to 1/2 of the screen size.

11.5 CREATE 3D DRAW GRAPHIC ORDER

NAME: DRAW3D

FUNCTION: Generates either an absolute or relative 3D draw graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL DRAW3D (X,Y,Z,MODE)

DESCRIPTION OF PARAMETERS:

X,Y,Z = Real variables supplied by the user which specify the 3D coordinate of the end point of a line to be drawn. The coordinate is in the user coordinate system.

MODE = An integer variable supplied by the caller which identifies the type of graphic instruction to be generated. When MODE=0, an absolute DRAW is implied and when MODE=1, a relative DRAW is implied. When MODE=0, the coordinate (X,Y,Z) specifies the absolute coordinate of the end point of a line to be drawn. When MODE=1, the coordinate specifies the offsets to be used in drawing a line from the current position to a new position.

DETAILED DESCRIPTION:

This routine converts the coordinate values specified to absolute screen coordinates and generates either an absolute or relative 3D draw graphic order. This graphic order is placed at the mark position of the currently opened page. Note that relative draws are restricted to 1/2 of the screen size.

11.6 TRANSFORM 3D to 2D

NAME: T3D2D

FUNCTION: Transform a page of graphic orders into a page which consists entirely of 2D graphic orders.

CALLING FORMAT: CALL T3D2D (IPAG3D,IPAG2D)

DESCRIPTION OF PARAMETERS:

IPAG3D = An integer variable supplied by the user which specifies the number of the 3D page which is to be transformed. IPAG3D must be in the range:

$1 < IPAG3D < 256$, $IPAGE3D \neq IPAG2D$

IPAG2D = An integer variable supplied by the user which specifies the number of the 2D page in which the transformed graphic orders are to be placed. IPAG2D must be in the range:

$1 < IPAG2D < 256$, $IPAG2D \neq IPAG3D$

DETAILED DESCRIPTION:

This routine sets up the coordinate converter to perform a 3D to 2D transformation on the page of graphic instructions specified by IPAG3D. The coordinate converter is started and a block of transformed graphic instructions is output to the page specified by IPAG2D. Note that unpredictable results will occur if the 2D output file is not large enough to accommodate the transformed graphic instructions. Refer to Appendix D to determine the necessary output file size.

11.7 COMPUTE AND REPLACE MATRIX PARAMETERS

NAME: MTRX3D

FUNCTION: Compute matrix parameters and generate coordinate converter instruction to update the matrix parameters.

CALLING FORMAT: CALL MTRX3D (ARRAY)

DESCRIPTION OF PARAMETERS:

ARRAY = A 12 element real array supplied by the user which specifies the scaling, translation, and rotation factors necessary for computing the matrix parameters. The array parameters and their valid ranges are specified below:

ARRAY Element	Definition	Range
1	X-Pre Translation	$-(XU-XL) \leq XPRE \leq +(XU-XL)$
2	Y-Pre Translation	$-(YU-YL) \leq YPRE \leq +(YU-YL)$
3	Z-Pre Translation	$-(ZU-ZL) \leq ZPRE \leq +(ZU-ZL)$
4	X Scale Factor	$1/256 < XSC < 128$
5	Y Scale Factor	$1/256 \leq YSC < 128$
6	Z Scale Factor	$1/256 \leq ZSC < 128$
7	X Rotation (Y-Z Plane)	any real number (in radians)
8	Y Rotation (X-Z Plane)	any real number (in radians)
9	Z Rotation (X-Y Plane)	any real number (in radians)
10	X-Post Translation	$-(XU-XL) \leq XPOST \leq +(XU-XL)$
11	Y-Post Translation	$-(YU-YL) \leq YPOST \leq +(YU-YL)$
12	Z-Post Translation	$-(ZU-ZL) \leq ZPOST \leq +(ZU-ZL)$

The variables used above are defined in the SCALE and SCAL3D subroutine descriptions.

e.g. CALL SCALE (XL,YL,XU,YU)

CALL SCAL3D (ZL,ZU)

DETAILED DESCRIPTION:

This routine uses the array parameters passed to compute new matrix parameters. These parameters are computed as indicated in Appendix A and are entered into the CCBLK of the currently opened page, replacing the previous matrix parameters. This routine must be called each time the user wishes to change translation, scaling, or rotation factors.

11.8 UPDATE VIEW POINT IN CCBLK

NAME: VIEWPT

FUNCTION: Updates view point parameters in the CCBLK of the currently opened 3D page.

CALLING FORMAT: CALL VIEWPT (X,Y,Z)

DESCRIPTION OF PARAMETERS:

(X,Y,Z) = Real variables supplied by the user which specify the viewing point which is to be used in generating perspective. These parameters are specified in user coordinates. The valid ranges for these parameters are:

$$|X| \leq \frac{XU-XL}{2}$$

$$|Y| \leq \frac{YU-YL}{2}$$

$$0 < Z \leq ZU-ZL$$

DETAILED DESCRIPTION:

This routine updates the view point in the currently opened page. The CCBLK view point parameters, generated by a previous call to the CCBLK routine, are updated. This routine must be called in order to change the view point. Note that the view point parameter range limits specified above are constrained by the host computer word size.

11.9 UPDATE VIEWBOX

NAME: VIEWBX

FUNCTION: Update viewbox boundaries in the currently opened page.

CALLING FORMAT: CALL VIEWBX (LV,RV,BV,TV,NV,FV)

DESCRIPTION OF PARAMETERS:

LV = Real variable supplied by the user which specifies the viewbox left boundary (minimum X). The valid range for LV is: $XL < LV < RV$.

RV = Real variable supplied by the user which specifies the viewbox right boundary (maximum X). The valid range for RV is: $LV < RV < XU$.

BV = Real variable supplied by the user which specifies the viewbox bottom boundary (minimum Y). The valid range for BV is: $YL < BV < TV$.

TV = Real variable supplied by the user which specifies the viewbox top boundary (maximum Y). The valid range for TV is: $BV < TV < YU$.

NV = Real variable supplied by the user which specifies the viewbox near boundary (minimum Z). The valid range for NV is: $ZL \leq NV < FV$.

FV = Real variable supplied by the user which specifies the viewbox far boundary (maximum Z). The valid range for FV is: $NV < FV \leq ZU$.

The parameters LV, RV, BV, TV, NV and FV are all specified in the user coordinate system.

DETAILED DESCRIPTION:

This routine updates the viewbox boundaries in the currently opened page. The CCBLK viewbox parameters, generated by a previous call to the CCBLK routine, are updated. This routine must be called in order to change the viewbox.

SECTION 12

IMAGE CONTROL ROUTINES

The routines in this section have been designed specifically to control the presentation of the image displayed.

Software Control

- CLIP - Remove data outside the viewing window
- SMOOTH - Remove unnecessary points in a curve
- SPLIT - Modify screen sectioning parameters

12.1 REMOVE OFF-SCREEN DATA

NAME: CLIP

FUNCTION: Eliminates lines and graphic data that lie outside a user specified window.

CALLING FORMAT: CALL CLIP (IOP, X1, Y1, X2, Y2, XLC, YLC, XUC, UYC, XPOSB, YPOSB)

DESCRIPTION OF PARAMETERS:

If IOP = 1,

then

X1, Y1 = are supplied by the caller to 'CLIP' as candidates for a positioning command (MOVE). CLIP checks these values and determines if they lie within the window described by (XLC, YLC) and (XUC, YUC). IOP is returned as follows:

8 - (X1, Y1) lie outside the window. No action should be taken by the caller on these coordinates.

9 - (X1, Y1) lie within the window. The caller may call 'MOVE' to position (X1, Y1).

NOTE

X2, Y2 are not used.

If IOP = 2,

then

X2, Y2 = are supplied by the caller as the end point of a vector to be drawn having as its start point the current position. CLIP determines how much, if any, of the vector will be visible within the specified window. IOP is returned as follows:

7 - the start point of the vector is within window. Call DRAW using X2, Y2.

8 - the vector lies entirely outside the window and therefore should not be drawn. (No action should be taken by the caller.)

9 - the start point of the vector lies outside the window. The user should call MOVE using X1, Y1 to move to within the window boundaries. A call to DRAW using X2, Y2 will then display the visible portion of the vector.

(XLC, YLC, XUC, YUC) = window limits in user coordinates.

XLC = X lower left corner
YLC = Y lower left corner

XUC = X upper right corner
YUC = Y upper right corner

(XPOSB, YPOSB) = current position. These are used to keep track of the current position, and should be the same two variables on each call to CLIP. These are neither set nor used by the user.

NOTE

This is a software function and all point removal is done prior to hardware windowing functions.

12.2 SMOOTH DISPLAYED LINES

NAME: SMOOTH

FUNCTION: This subroutine straightens lines by removing unnecessary points thus saving room in the display buffer.

CALLING FORMAT: CALL SMOOTH (IOP, X, Y, ISAVE, XSAVE, YSAVE, MSAVE, EPS)

DESCRIPTION OF PARAMETERS:

1. IOP = 1: Line break. This call initializes the subroutine. X and Y are not used.

2. IOP = 2: New point. X and Y represent a new point in the current line.

3. IOP = 3: Line end. This call indicates that the last point has been appended to the current line. X and Y are not used. This call forces out a point. The next call can be with IOP = 2 to continue the line or with IOP = 1 to start a new line.

On return, IOP, X and Y are set as follows:

1. IOP = 4: No action required.

2. IOP = 5: Call MOVE to move to X, Y.

3. IOP = 6: Call DRAW to draw a line from the present position to X, Y.

4. IOP = 7: Parameter error - IOP not 1-3.

The "line break" call always returns IOP = 4 and the "line end" call never returns IOP = 5.

The parameter ISAVE is used internally to remember how many points are being buffered, and should be the same variable on each call to SMOOTH. XSAVE and YSAVE are real arrays of length MSAVE which are used to buffer data points. Each call to SMOOTH should pass the same two arrays. EPS is the amount of excursion from a straight line that will cause the line to be broken. To avoid any change in the screen image due to smoothing, EPS should be set to the screen resolution, which is 0.0009765625 (1/1024), of full screen width for the GRAPHIC 8. In user coordinates, screen width is the difference between the first and third (or second and fourth) arguments to SCALE. Smaller values of EPS may use more display buffer at the expense of making the picture less precise. IF EPS is set to 0.0, SMOOTH produces output identical to its input with the exception of eliminating adjacent coincidental points.

12.3 DEFINE PIXEL MEMORY MAPPING PARAMETERS

NAME: SPLIT

FUNCTION: Allows the caller to map pixel memory up to three horizontal sections on the display monitor.

CALLING FORMAT: CALL SPLIT (IMON, INUM, ARRAY)

DESCRIPTION OF PARAMETERS:

IMON = Integer variable supplied by the caller specifying which display monitor to select.

1 = display monitor #1
.
.
.
4 = display mnitor #4

INUM = Integer variable supplied by the caller specifying the number of sections into which the display monitor is to be divided.

ARRAY = Real array supplied by the caller containing the X and Y values, in user coordinates, for the start positions and the number of lines for each area of pixel memory that is to be mapped to the display monitor. ARRAY will be either three, six, or nine elements long defined in the following manner:

ARRAY (1) - area #1 start X coordinate
(2) - area #1 start Y coordinate
(3) - number of lines to build section #1
(4) - area #2 start X coordinate
(5) - area #2 start Y coordinate
(6) - number of lines to build section #2
(7) - area #3 start X coordinate
(8) - area #3 start Y coordinate
(9) - number of lines to build section #3

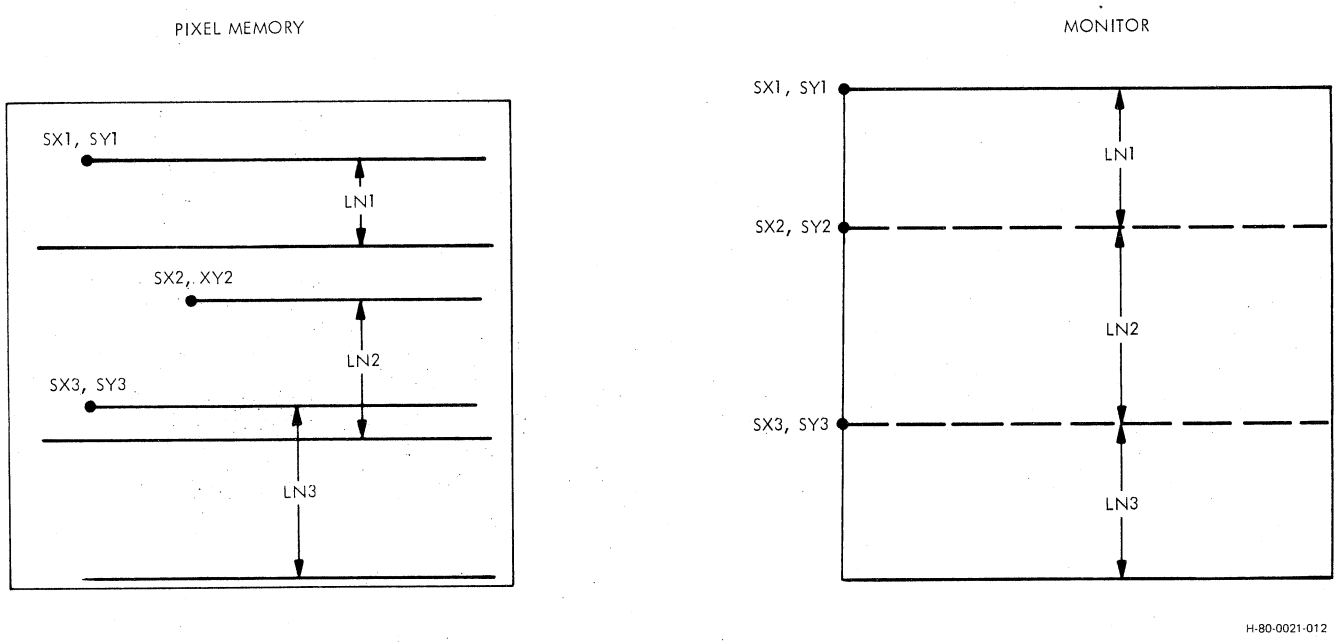
DETAILED DESCRIPTION:

This routine will generate the necessary graphic orders and place them at the mark position in the currently open page.

Proper use of this call requires an understanding of the relationships between the components of the GRAPHIC 8 controller. The following information is important to the user of SPLIT.

- Each pair of mapping memories is dedicated to one display monitor.
- The image being displayed will be affected immediately upon execution of these graphic orders. Therefore, the user of SPLIT must be carefully coordinated with calls to image generation routines.
- There is a timing overhead involved in updating the proper control registers needed for SPLIT. Therefore, it should be called as a graphic subroutine and executed only once.

The following diagram is set up to show the relationship between pixel memory areas and display monitor sections. (See example under COLORI section.)


PIXEL MEMORY
SX1, SY1
LN1
SX2, XY2
LN2
SX3, SY3
LN3
MONITOR
SX1, SY1
LN1
SX2, SY2
LN2
SX3, SY3
LN3
H-80-0021-012

SECTION 13

DATA TRANSFER ROUTINES

This section describes how to transmit a block of predefined refresh data request and subsequently receive a block of refresh data, store pixel data in read/write memory, transfer pixel data from read/write memory to pixel memory, and obtain error codes currently displayed on enabled display monitors.

The following routines are included in this section.

- REFDAT - Transfer a block of refresh data
- REQIM - Request refresh data
- GETIM - Receive refresh data
- MOVDAT - Store/write pixel data
- GETERR - Receive error information

13.1 TRANSFER A BLOCK OF PREDEFINED GRAPHIC ORDERS

NAME: REFDAT

FUNCTION: Allows the caller to transfer and display a block of predefined graphic orders (MOVE, DRAW, TEXT, POINT, CIRCLE)

CALLING FORMAT: CALL REFDAT (IARRAY, N)

DESCRIPTION OF PARAMETERS:

IARRAY = An integer array containing graphic orders right adjusted in the right-most 16 bits of each element.

N = An integer variable containing the number of elements in the array.

$$1 \leq N \leq 20$$

DETAILED DESCRIPTION:

This routine takes the lower 16 bits (right-most) of the first N elements found in the array IARRAY and places them at the mark position of the currently opened page.

The contents of the array IARRAY must be predefined graphic orders. The image generated by the transferred contents of the array IARRAY are displayed when the currently opened page is displayed.

13.2 REQUEST REFRESH IMAGE

NAME: REQIM

FUNCTION: Initiates a request for a block of up to 20 words to be transferred from an opened page back to the host.

CALLING FORMAT: CALL REQIM (NINST)

DESCRIPTION OF PARAMETERS:

NINST = Integer variable specifying the number of refresh data to be transferred starting at present mark.

$$1 \leq \text{NINST} \leq 20$$

DETAILED DESCRIPTION:

This routine causes an image event to occur which has an event code of 9. The image event contains up to 20 words of refresh code, starting at the current mark position supplied. The page is assumed to be the currently opened page.

13.3 GET REFRESH IMAGE

NAME: GETIM

FUNCTION: Retrieves from the event tables an array of data which is the refresh image code.

CALLING FORMAT: CALL GETIM (IARRAY, ISIZE, NINST, IPAGE)

DESCRIPTION OF PARAMETERS:

IARRAY = An integer array supplied by the caller into which the refresh data is transferred.

ISIZE = An integer variable supplied by the caller containing the maximum number of words of refresh data to be placed in the array; i.e., ISIZE is the size of the array. Any data in excess of the limit is discarded.

NINST = An integer variable returned to the caller specifying the number of words of refresh data transferred.

IPAGE = An integer variable returned to the caller identifying the page number from which the data was transferred.

DETAILED DESCRIPTION:

The REQIM routine initiates this event; the EVENT subroutine detects the event and this routine retrieves the data.

Example:

```
C     Read 10 words from page 3 starting at
C     mark 5.
      DIMENSION IARRAY (10)
         .
         .
         .
      CALL UPDATE (3,5)
      CALL REQIM (10)
10    CALL EVENT (IEVNT)
      IF (IEVNT.NE.9) GO TO 10
      CALL GETIM (IARRAY, 10, NINST, IPAGE)
```

13.4 MOVE PIXEL DATA

NAME: MOVDAT

FUNCTION: This subroutine allows the caller to store the pixel data from a screen image into read/write memory and to retrieve pixel data in read/write memory to present it to the display monitor.

CALLING FORMAT: CALL MOVDAT (IPATH, IDATPG, XI, YI, XF, YF, IMODE)

DESCRIPTION OF PARAMETERS:

IPATH = An integer variable supplied by the caller specifying the direction of data transfer.
- 2, Read/write memory to pixel memory
- 1, Pixel memory to read/write memory

IDATPG = An integer variable supplied by the caller containing the page number where the pixel data resides or will reside.

XI, YI = Absolute or relative initial X, Y coordinates of the lower left corner of the rectangular pixel array. The coordinate is in the user coordinate system.

XF, YF = Absolute or relative final X, Y coordinates of the upper right corner of the rectangular pixel array. The coordinate is in the user coordinate system.

IMODE = Integer variable supplied by the caller describing the rectangular pixel array in pixel memory
- 1, Horizontal scan with absolute X, Y address
- 2, Vertical scan with absolute X, Y address
- 3, Horizontal scan with X, Y address relative to the current X, Y position
- 4, Vertical scan with the X, Y address relative to the current X, Y position

DETAIL DESCRIPTION:

A Move Pixel Data graphic order is constructed at the current mark of the currently opened page.

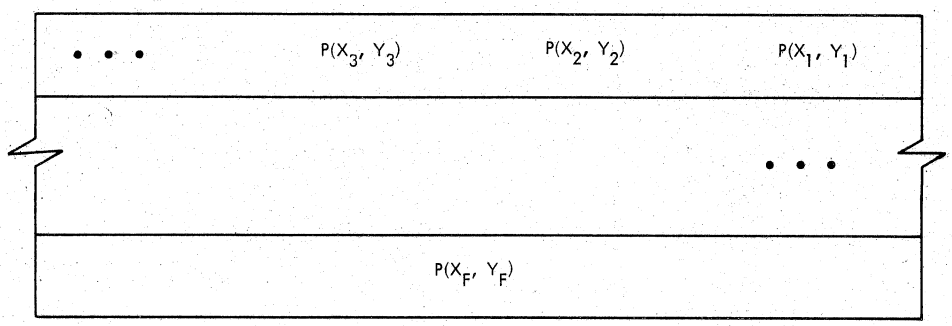
PROGRAMMING CONSIDERATIONS:

When the user wishes to write to pixel memory, the pixel data is first formatted and written to the designated data page IDATPG. The target display(s) are selected prior to the CALL MOVDAT using the CALL STATUS.

```
C open page
     CALL ADDREF (page)

C select the display
     CALL STATUS (IBL, INDX, IVT, IDISP)

C setup move pixel data graphic order
     CALL MOVDAT (1, IDATPG, XI, YI, XF, YF, IMODE)

C send formatted pixel data to data page
     CALL ADDREF (IDATPG)
     CALL REFDAT (IARRAY, IDATPG)

C link to refresh
     CALL PICTUR (1)
     CALL PICTUR (page)
```

Pixel data (value of pixel gray or color level) is formatted by the program (usually prior to the CALL MOVDAT) as 4 or 8 bit bytes packed right justified into each entry of the array IARRAY for a total of 16 bits per array entry. The size of the byte (4 or 8) corresponds to the bits per pixel of the system including blink if present.

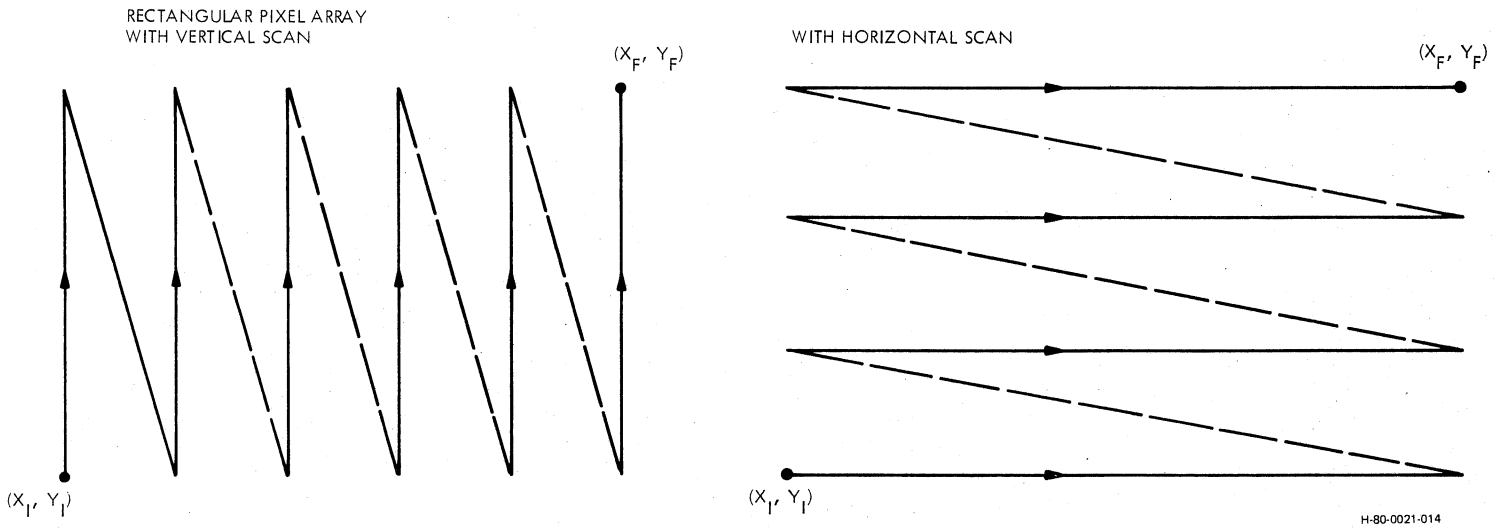
Format of packed Pixel Data Array for 4 bits per pixel system P (Xn, Yn) represents the gray value for the nth pixel in the rectangular pixel array on the screen.



H-80-0021-013

When the user wishes to retrieve data from pixel memory, only one display should be selected. Since the data being retrieved is from the pixel memory currently being written by graphic orders, the page in which the Move Pixel data graphic order is executed should be the last page called in the sequence of linked pages in page 1.

Once this page is called and the data is transferred to the data page, the program can remove the page from linked pages. This will ensure that the data page is not being updated when the program wishes to operate on the retrieved pixel data.



13.5 TRANSMIT OUTPUT BUFFER

NAME: DATEND

FUNCTION: This subroutine allows the caller to force the transmission of the data in the output buffer regardless of the conditions set by blocking mode.

CALLING FORMAT: CALL DATEND

DETAIL DESCRIPTION:

All data in the output buffer is transmitted to the GRAPHIC 8 terminal immediately.

13.6 GET ERROR INFORMATION

NAME: GETERR

FUNCTION: Retrieves information concerning an error event

CALLING FORMAT: CALL GETERR (IARRAY)

DESCRIPTION OF PARAMETERS:

IARRAY = A 4-word integer array supplied by the caller into which the 4-word error information is placed.

```
IARRAY(1) = Error code
IARRAY(2) = 0
IARRAY(3) = 0
IARRAY(4) = 0
```

DETAILED DESCRIPTION:

Detection of any of the error conditions described in Appendix C causes an error event (event code 8). An error event is detected by the EVENT routine and this routine actually retrieves the error data. The error code is returned as two 8-bit ASCII characters right adjusted in IARRAY (1). These same two characters are displayed in certain cases in the upper left corner of the display.

Example:

```
      CALL EVENT (IEVNT)
      IF (IEVNT.NE.8) GO TO 10
      CALL GETERR (IARRAY)
10    CONTINUE
```

SECTION 14

FSP INPUT/OUTPUT

Four subroutines are used by FSP for performing I/O to the GRAPHIC 8 terminal:

- G8INIT - Initialize host/GRAPHIC 8 I/O driver
- G8TERM - Terminate host/GRAPHIC 8 I/O driver
- MSGOUT - Output message to GRAPHIC 8 terminal
- MSGIN - Input message from GRAPHIC 8 terminal

The calling sequences for these subroutines are defined by Sanders, but the actual routines themselves are supplied by the customer, i.e., Sanders does not provide the host software necessary to perform the actual I/O (see figure 1), unless special arrangements have been made.

FSP OUTPUT

Most of the FSP subroutines, when called, perform the following functions:

1. Create a message (header plus data) and place it in an output block.
2. Call MSGOUT to transmit the message to the GRAPHIC 8 terminal for execution.

FSP INPUT

The graphic control program enhanced (GCP) in the GRAPHIC 8 sends data to the host when polled by the host, i.e., keyboard, and PED events are sent to the host only on request. A poll request and response sequence works as follows:

CALL MSGOUT - Outputs a POLL request to GRAPHIC 8

CALL MSGIN - Read POLL response from GRAPHIC 8

FSP then analyzes the POLL response message and updates internal tables accordingly.

14.1 INITIALIZE HOST/GRAPHIC 8 I/O DRIVER

NAME: G8INIT

FUNCTION: To initialize the host/GRAPHIC 8 I/O driver.

CALLING FORMAT: CALL G8INIT (IUNIT)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 8 in the call to the INIT subroutine.

DETAILED DESCRIPTION:

The calling sequence for this routine is defined by Sanders, but the actual routine itself must be supplied by the customer. This routine is not called directly by the application program, but rather is called internally by FSP as a result of the call to INIT subroutine.

For parallel hosts, when the application program makes a call to the FSP subroutine, several functions are performed to initialize the GRAPHIC 8 terminal to FSP mode. One function is to logically connect the application program with the GRAPHIC 8 within the architecture of the host operating system. The actual operations needed to perform the connection are host-dependent. Internal to the INIT subroutine a call is made to the G8INIT subroutine to provide the customer with a mechanism for performing the I/O driver initialization process. In terms of the GRAPHIC 8, the key function performed by G8INIT is to initialize (under program control) the terminal controller by pulsing the INIT control line to the GRAPHIC 8 parallel interface card. The INIT pulse resets the terminal controller to the system mode.

NOTE

The INIT control line is exposed to the host end.

For serial hosts, the G8INIT subroutine is not needed. To eliminate compilation errors, the customer should write a dummy G8INIT subroutine that only has a return statement.

For example:
```
SUBROUTINE G8INIT (IUNIT)
RETURN
END
```

For parallel hosts, the design of the G8INIT subroutine is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Host operating system
- GRAPHIC 8 I/O driver (provided by customer)

Any unrecoverable errors detected by this subroutine, I/O driver or operating system, should cause termination of the job with appropriate diagnostic messages.

14.2 TERMINATE HOST/GRAPHIC 8 I/O DRIVER

NAME: G8TERM

FUNCTION: To terminate the host/GRAPHIC 8 I/O driver

CALLING FORMAT: CALL G8TERM (IUNIT)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 8 in the call to the INIT subroutine.

DETAILED DESCRIPTION:

The calling sequence for this routine is defined by Sanders, but the actual routine itself must be supplied by the customer. This routine is not called directly by the application program, but rather is called internally by FSP as a result of the call to the INIT subroutine.

For parallel hosts, when the applications program makes a call to the THEEND subroutine, several functions are performed to terminate the FSP mode of operation and to return the GRAPHIC 8 terminal to the TTY emulator mode. One function is to logically terminate the connection between the application program and the GRAPHIC 8. The actual operations needed to terminate the connection are host-dependent. Internal to the THEEND subroutine a call is made to the G8TERM subroutine to provide the customer with a mechanism for performing the I/O driver termination process.

For serial hosts, the G8TERM subroutine is not needed. To eliminate compilation errors, the customer should write a dummy G8TERM subroutine that only has a return statement.

For parallel hosts, the design of the G8TERM subroutine is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Host operating system
- GRAPHIC 8 I/O driver (provided by customer)

Any unrecoverable errors detected by this subroutine, I/O driver or operating system, should cause termination of the job with appropriate diagnostic messages.

14.3 OUTPUT MESSAGE TO GRAPHIC 8 TERMINAL

NAME: MSGOUT

FUNCTION: Outputs a message to the GRAPHIC 8 terminal.

CALLING FORMAT: CALL MSGOUT (IUNIT, IBUF, IELEMC)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 8 in the call to the INIT subroutine.

IBUF = An integer array, each entry of which contains two 8-bit bytes (one element), right adjusted.

IELEMC = An integer variable containing the number of elements in the array IBUF to the output. Control is returned to the caller only after all elements have been successfully transmitted.

DETAILED DESCRIPTION:

The calling sequence for this routine is defined by Sanders, but the actual routine itself must be supplied by the customer. This routine is not called directly by the application program, but rather is called internally as a result of calls to other FSP routines.

This routine invokes the I/O driver and requests output via the appropriate operating system call.

The actual details of this subroutine depend on the customer's design and implementation. The design is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Serial or parallel interface
- Host operating system
- GRAPHIC 8 I/O driver (provided by customer)

NOTE

In the serial mode, a "carriage return" must be appended to the data provided by the caller, i.e., MSGOUT sends to the GRAPHIC 8 each character supplied by the caller and then MSGOUT sends a carriage return. The 8 bit code for a carriage return may be either of the following:

00001101
or
10001101

Any unrecoverable errors detected by this routine, I/O driver or operating system should cause termination of the job with appropriate diagnostic messages.

14.4 INPUT MESSAGE FROM GRAPHIC 8 TERMINAL

NAME: MSGIN

FUNCTION: Inputs a message from the GRAPHIC 8 terminal.

CALLING FORMAT: CALL MSGIN (IUNIT, IBUF, IELEMC)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 8 in the call to the INIT subroutine.

IBUF = An integer array into which data received from the GRAPHIC 8 will be placed. Data will be packed two 8-bit bytes (one element) per array entry.

IELEMC = An integer variable containing the number of elements in the array IBUF to be filled. Control is returned to the caller only after all elements have been successfully received.

DETAILED DESCRIPTION:

The calling sequence for this routine is defined by Sanders, but the actual routine itself must be supplied by the customer. This routine is not called directly by the application program, but rather is called internally as a result of calls to other FSP routines.

This routine invokes the I/O driver and requests input via the appropriate operating system call.

The actual details of this subroutine depend on the customer's design and implementation. The design is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Serial or parallel interface
- Host operating system
- GRAPHIC 8 I/O driver (provided by customer)

NOTE

In the serial mode, the GRAPHIC 8 terminates each message sent to the host with a carriage return. This carriage return should be stripped by MSGIN and not supplied to the caller as part of his message.

Any unrecoverable errors detected by this subroutine, I/O driver or operating system, should cause termination of the job with appropriate diagnostic messages.

SECTION 15

DELIVERABLE ITEMS

The following FSP items are provided to the customer at installation time:

A. FSP Source Code

The FSP subroutines are provided as interpreted source code in card deck form (029 keypunch).

B. FSP Programmer's Reference Manual

C. FSP Sample Program

A sample FSP demonstration program written in FORTRAN is provided as interpreted source code in card deck form (029 keypunch).

D. Listings of the FSP Routines and the Sample Program

SECTION 16

INSTALLATION PROCEDURE

The following steps must be taken by the customer before the FSP sample program may be run:

A. The FSP subroutines (provided by Sanders) must be made part of the operating system subroutine library.

B. A GRAPHIC 8 I/O driver must be written and made part of the operating system.

C. The MSGOUT, MSGIN, G8INIT, and G8TERM subroutines must be written and made part of the operating system subroutine library.

D. A SETEXT subroutine must be written before the sample FSP program can be used.

E. The FSP sample program must be compiled and link-edited to create a load module.

NOTE

The following installation-dependent source statements in the sample program must be changed before compilation.

```
CALL INIT (IUNIT,Ø,IFACE)
```

IUNIT must be set to the logical unit number assigned to the GRAPHIC 8 device driver.

IFACE must be set to 1 for parallel interface, or set to 2 for serial interface.

After the FSP sample program has been successfully run, the customer should consider making the following improvements:

1. The INSERT, EXTRAC, and SHIFT subroutines should be rewritten in assembly language to improve overall system speed. For parallel users, when these routines are rewritten, the customer can expect to see improvements in the order of 50%. (I.e., if it takes 30 seconds to display an image on the CRT indicator using the FORTRAN versions of INSERT, EXTRAC, and SHIFT, then the same image should take about 15 seconds to display when using assembly language versions of these subroutines.)

For serial users, the speed improvements will only be reflected at the higher baud rates. No speed improvement will probably be seen when operating below 2400 baud. At 9600 baud, a speed improvement in the order of 15% to 30% can probably be achieved.

2. The CKPOLL subroutine can be modified to minimize the use of system resources when running FSP programs.

   When the CKPOLL subroutine is delivered, it is configured to operate in a polling mode. The GRAPHIC 8 is also configured to ignore command header errors. The polling mode configuration (set up by Sanders) works in the following manner:

   a. FSP user calls EVENT

   b. EVENT sends a POLL message to the GRAPHIC 8 via MSGOUT

   c. The GRAPHIC 8 receives the POLL message and does the following:

      (1) Sends out the next message in the O/P buffer.

      (2) If the O/P buffer is empty, the GRAPHIC 8 returns a dummy message to indicate that no message is ready. (Normally messages get stored in the O/P buffer in response to some operator inputs.)

For this configuration, the host computer is looping in a constant event loop. (I.e., for every POLL message sent, the GRAPHIC 8 returns a message.)

To minimize the number of host to GRAPHIC 8 messages, the CKPOLL subroutine can be modified so that the GRAPHIC 8 only sends a message back to the host computer when a new message is stored in the O/P buffer.

For parallel users, this type of poll mode can be selected by changing the IPOLL variable to 1. For this mode, error detection can also be enabled by setting IPOLL to 9. (I.e., when IPOLL = 9, error detection is enabled and messages are sent from the GRAPHIC 8 to the host only when a new message is stored in the O/P buffer.)

For serial users, this type of poll mode can be selected by changing the IPOLL variable to 1. For this mode, error detection can also be enabled by setting IPOLL to 9. For half-duplex serial transmissions, no problems should be encountered with an IPOLL value of 9. For full-duplex serial transmissions, echoing types of problems can be encountered. (I.e., when the host computer receives a message from the GRAPHIC 8, it echos it back to the GRAPHIC 8, which results in an endless loop of command header errors.) If error detection is enabled for full-duplex, then the user must write the MSGIN software in a way that ensures that no echoing of messages back to the GRAPHIC 8 occurs.

The CKPOLL subroutine can also be configured to operate in a special type of polling mode. In this mode, the sending of GRAPHIC 8 to host messages is controlled by a user designated special character. This mode works as follows:

1. FSP user calls EVENT.

2. EVENT sends a POLL message to the GRAPHIC 8 via MSGOUT.

3. EVENT sends a special character to the GRAPHIC 8 to indicate that the host is set up to read in the next message from the GRAPHIC 8.

4. When the GRAPHIC 8 receives the POLL message, it starts looking for the special character. When it detects the special character, it sends the next message back to the host.

The special character type of polling mode is only applicable to serial users. This mode is used in cases where the host operating system can't get set up in time to receive incoming messages from the GRAPHIC 8.

If the special character type of polling mode is used, then the ISPCHR variable should be changed to the customer-selected value.

NOTE

The CKPOLL subroutine can also be set up to operate in a non-polling mode. In this mode the GRAPHIC 8 sends messages back to the host computer anytime there is a message in the O/P buffer. Please refer to the IM initialize I/O message formats message for additional information on running FSP programs in a non-polling environment. The IM message is described in the GRAPHIC 8 GCP Programmers Reference Manual.

Normally when the THEEND subroutine is called, the GRAPHIC 8 is returned to the full-duplex teletypewriter emulator. If half-duplex is being used, the THEEND subroutine can be modified to return the user to the half-duplex teletypewriter emulator as follows:

Change IOUTB(2) = 30884 to IOUTB(2) = 30880

SECTION 17

STARTUP PROCEDURE

The following paragraphs assume the following:

- All steps of the installation procedure have been performed.
- The GRAPHIC 8 terminal is hardware-wise connected to the host.
- All power is on and the brightness and contrast knobs on the display indicator are set properly.

1. Press LOCAL button on front panel of the GRAPHIC 8 and observe built-in test pattern. Validate (using this pattern and its associated built-in diagnostics) that the terminal is in working order.

2. Press the "RETURN" key on the keyboard (causes pattern to disappear and "BO M" to appear.

3. Press "Y" key followed by "RETURN" key to enter teletypewriter emulation mode. At this point the GRAPHIC 8 performs as a teletypewriter emulator until such time that a FORTRAN/FSP program (e.g., FSP sample program) is executed in the host. The call to GSS4, when executed, causes GCP to enter the SYSTEM mode. GCP remains in the SYSTEM mode until a call to THEEND is made, at which time the teletypewriter emulator is re-entered.

NOTE

Refer to the GRAPHIC 8 GCP Programmer's Reference Manual (H-80-0444) for more information on LOCAL mode features.

APPENDIX A

ALPHABETICAL SUMMARY OF SUBROUTINES

The following FORTRAN callable FSP subroutines are available to the application program in the host computer.

FSP SUBROUTINES

PAGE	SUBROUTINE	DESCRIPTION
6-12	ADDREF (IPAGE)	Open page for adding refresh data
10-1	CC2DBL	Create 2D Converter Block
11-2	CCBLK	Build 3D Parameter Block
5-7	CIRCLE (RADIUS, IQUAD)	Draw a circle
12-1	CLIP (IOP, X1, Y1, X2, Y2, XLC, YLC, XUC, YUC, XPOSB, YPOSB)	Remove off-screen data
4-4	COLORI (IDISPL, INUM, ISNDX, ARRAY1, ARRAY2, ARRAY3, IMODEL)	Color control using either RGB or HLS model
6-15	COPYIM (MARKA, MARKB)	Copy a block of graphic orders
13-5	DATEND	Send all contents of output buffer to GRAPHIC 8
5-3	DRAW (X, Y, MODE)	Draw a vector
10-3	DRAW2D (X, Y, MODE)	Draw a 2 dimensional vector
11-5	DRAW3D (XD, YD, ZD, MODE)	Draw a 3 dimensional vector
3-6	DSABOX (IND)	Turn border display off
8-8	DSACUR (IDUNM)	Turn crosshair cursor off
3-7	DSAERR (IND)	Turn error display off
8-2	DSAPAD (IKEY)	Disable alphanumeric scratch pad
8-6	DSAPED (IDUNM)	Disable a PED Device

PAGE	SUBROUTINE	DESCRIPTION
9-3	DSAPMD	Disable packed vector mode
5-8	ELIPSE (XSEMI, YSEMI, IQUAD)	Draw an ellipse
3-5	ENBBOX (IND)	Turn border display on
8-7	ENBCUR (IDUNM, IDISPL, IDVTP, ICUR)	Turn on one of the default cursors
3-6	ENBERR (IND)	Turn error display on
8-2	ENBPAD (IKEY, IND, X, Y IMAX)	Enable alphanumeric scratch pad
8-6	ENBPED (IDUNM, IDVTP, IPAGE, KMARK)	Enable a PED device
9-2	ENBPMD	Enable packed vector
6-13	ERASEP	Erase from page mark to end of page
7-1	EVENT (IEVNT)	Poll terminal for event or request
5-10	FILL (INUM, ARRAY, IMODE)	Fill a convex polygon
14-1	G8INIT (IUNIT)	Initialize host/GRAPHIC 8 I/O driver
14-3	G8TERM (IUNIT)	Terminate host/GRAPHIC 8 I/O driver
13-6	GETERR (IARRAY)	Get error information
13-2	GETIM (IARRAY, ISIZE NINST, IPAGE)	Get refresh image
8-4	GETKEY (KBD, KEY)	Get function key event information
6-14	GETMRK (M)	Get mark request information
8-9	GETTB (NUMBER, X, Y)	Get red request information
8-3	GETTXT (IARRAY, ISIZE, NCHAR, KBD)	Get text event information
4-6	GRAYI (IDISP, INUM, ISNDX, ARRAY)	Define the look-up table with selectable Gray levels
5-9	HTPLOT (TAB, INUM, YARRAY, JMODE)	Plot a series of Y coordinates equally spaced

FSP SUBROUTINES (Cont)

PAGE	SUBROUTINE	DESCRIPTION
3-1	INIT (IUN, IOPTN, IFACE)	Initialize GRAPHIC 8 to FSP mode
11-1	INIT3D	Initialize 3D Coordinate Converter
4-4	LAMPOF (KBD, LAMP)	Turn keyboard lamp off
4-3	LAMPON (KBD, LAMP)	Turn keyboard lamp on
3-2	LAYOUT (NPAGES, LNGARY)	Define graphic page layout
4-2	LMARGN	Set left (or lower) margin
5-1	MOVE (X, Y, MODE)	Move to the 3D position specified
10-2	MOVE2D (X, Y, MODE)	Move to the 2D position specified
13-3	MOVDAT (IPATM, IDATPG, XI, YI, XF, YF, IMODE)	Store and retrieve Pixel data
11-5	MOVE3D (XD, YD, ZD, MODE)	Move to the 3D position specified
6-14	MOVEIM (MARKFR, MARKTO)	Move a block of graphic orders
10-4	MTRX2D (ARRAY)	Replace 2D composite matrix
11-6	MTRX3D (ARRAY)	Replace 3D composite matrix
14-5	MSGIN (IUNIT, IBUF, IELEMC)	Input message from GRAPHIC 8 Terminal
14-3	MSGOUT (IUNIT, IBUF, IELEMC)	Output message to GRAPHIC 8 Terminal
5-6	NEWLIN	Return to left (or lower) margin and increment one line
9-3	PDRAW (X, Y)	Packed vector draw
6-13	PICTUR (IPAGE)	Graphic subroutine call
9-2	PMOVE (X,Y)	Packed vector move
13-1	REFDAT (IARRAY, N)	Transfer a block of graphic orders
13-2	REQIM (NINST)	Request refresh image
8-8	REQTB (NUMBER)	Request PED X, Y
3-5	SCALE (XL, YL, XU, YU)	Define 2 dimensional coordinates
11-2	SCAL3D (ZL, ZU)	Define 3 dimensional coordinates

FSP SUBROUTINES (Cont)

PAGE	SUBROUTINE	DESCRIPTION
4-7	SCOLOR (ICOLOR)	Select Color
4-8	SGRAY (IGRAY)	Select Gray Level
12-2	SMOOTH (IOP, X, Y, ISAVE, XSAVE, YSAVE, MSAVE, EPS)	Smooth displayed lines
12-3	SPLIT (IMONIT, INUM, ARRAY)	Split screen on specified monitor
4-2	STATUS (IBL, INT, IVT, IND)	Set display status
10-3	T2D2D (IGRAPH, IPAG2D)	Transform graphic page to 2D page
11-6	T3D2D (IPAGE3D, IPAG2D)	Convert 3D page to 2D page
5-4	TEXT (N, IARRAY)	Display text characters
3-7	THEEND	Terminate FSP mode
4-1	TPARM (ICSIZE, CHSPAC, RLSPAC, ICROT)	Set text parameters
6-12	UPDATE (IPAGE, MARK)	Open page for editing refresh data
10-5	V2DBOX (LV, RV, BV, TV)	Update 2D viewbox parameters in CCBLK
11-8	VIEWBX (VWLEFT, VWRGHT, VWBOTM, VWTOP, VWNEAR, VWFAR)	Update 3D viewbox parameters in CCBLK
11-7	VIEWPT (X, Y, Z)	Change viewpoint position
5-10	VTPLOT (TAB, INUM, XARRAY, IMODE)	Plot a series of X coordinates equally spaced
5-9	XYPLOT (INUM, XARRAY, YARRAY, IMODE)	Plot a series of points at the X-Y positions specified

APPENDIX B

ASCII CODES

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
000	00	NUL	@	NULL, CTRL SHIFT P, TAPE LEADER
001	01	SOH	A	START OF HEADER, SOM
002	02	STX	B	START OF TEXT, EOA
003	03	ETX	C	END OF TEXT, EOM
004	04	EOT	D	END OF TRANSMISSION, END
005	05	ENQ	E	ENQUIRY, WRU, WHO ARE YOU
006	06	ACK	F	ACKNOWLEDGE, RU, ARE YOU
007	07	BEL	G	BELL
010	08	BS	H	BACKSPACE, FE0
011	09	HT	I	HORIZONTAL TAB, TAB
012	0A	LF	J	LINE FEED, NEW LINE, NL
013	0B	VT	K	VERTICAL TAB, VTAB
014	0C	FF	L	FORM FEED, FORM, PAGE
015	0D	CR	M	CARRIAGE RETURN, EOL
016	0E	SO	N	SHIFT OUT, RED SHIFT
017	0F	SI	O	SHIFT IN, BLACK SHIFT
020	10	DLE	P	DATA LINK ESCAPE, DC0
021	11	DC1	Q	XON, READER ON
022	12	DC2	R	TAPE, PUNCH ON
023	13	DC3	S	XOFF, READER OFF
024	14	DC4	T	TAPE, PUNCH OFF
025	15	NAK	U	NEGATIVE ACKNOWLEDGE, ERR

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
026	16	SYN	V	SYNCHRONOUS IDLE, SYNC
027	17	ETB	W	END OF TEXT BUFFER, LEM
030	18	CAN	X	CANCEL, CANCL
031	19	EM	Y	END OF MEDIUM
032	1A	SUB	Z	SUBSTITUTE
033	1B	ESC		ESCAPE, PREFIX
034	1C	FS		FILE SEPARATOR
035	1D	GS		GROUP SEPARATOR
036	1E	RS		RECORD SEPARATOR
037	1F	US		UNIT SEPARATOR
040	20	SP		SPACE, BLANK
041	21	!		
042	22	"		
043	23	#		
044	24	$		
045	25	%		
046	26	&		
047	27	'		APOSTROPHE
050	28	(		
051	29	)		
052	2A	*		
053	2B	+		
054	2C	,		COMMA
055	2D	-		MINUS
056	2E	.		

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
057	2F	/		
060	30	0		NUMBER ZERO
061	31	1		NUMBER ONE
062	32	2		
063	33	3		
064	34	4		
065	35	5		
066	36	6		
067	37	7		
070	38	8		
071	39	9		
072	3A	:		
073	3B	;		
074	3C	<		LESS THAN
075	3D	=		
076	3E	>		GREATER THAN
077	3F	?		
100	40	@		SHIFT P
101	41	A		
102	42	B		
103	43	C		
104	44	D		
105	45	E		
106	46	F		
107	47	G		

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
110	48	H		
111	49	I		LETTER I
112	4A	J		
113	4B	K		
114	4C	L		
115	4D	M		
116	4E	N		
117	4F	O		LETTER O
120	50	P		
121	51	Q		
122	52	R		
123	53	S		
124	54	T		
125	55	U		
126	56	V		
127	57	W		
130	58	X		
131	59	Y		
132	5A	Z		
133	5B			SHIFT K
134	5C			SHIFT L
135	5D			SHIFT M
136	5E			↑ SHIFT N
137	5F	_		← SHIFT O, UNDERSCORE

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
140	60	;		ACCENT GRAVE
141	61	a		
142	62	b		
143	63	c		
144	64	d		
145	65	e		
146	66	f		
147	67	g		
150	68	h		
151	69	i		
152	6A	j		
153	6B	k		
154	6C	l		
155	6D	m		
156	6E	n		
157	6F	o		
160	70	p		
161	71	q		
162	72	r		
163	73	s		
164	74	t		
165	75	u		
166	76	v		
167	77	w		

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
170	78	x		
171	79	v		
172	7A	z		
173	7B	{		
174	7C	|		VERTICAL SLASH
175	7D	}		ALT MODE
176	7E	~		(ALT MODE)
177	7F	DEL		DELETE, RUBOUT

APPENDIX C

ERROR CODES

When an error occurs in an FSP program, the error code will appear in the upper left corner of the screen. The code displayed will be "00" until an error is encountered, at which time the error code will change to reflect the error condition. The user is then required to correct the error and rerun the user program. FSP will continue to execute the user program after an error has occurred, but unpredictable results may be observed.

The following list of error codes should aid the user in correcting any error that occurs.

CODE	DESCRIPTION	ROUTINE CALLED WHEN ERROR OCCURRED
02	GCP output buffer full PED coordinates not sent to the host.	REQTB
03	GCP output buffer full - GCP error message cannot be sent to host.	----------
04	GCP input buffer full - messages cannot be sent to GCP.	----------
05	GCP output buffer full - keyboard information cannot be sent to host.	Pressing A/N or Function on Keyboard
08	GCP output buffer full - return image cannot be sent to the host.	REQIM
12	Current address not on current page.	ERASEP
19	IFAC Value out of range.	Internal routines
21	PED is not enabled.	REQTB, DTMODE*
22	Not in pack vector mode.	PDRAW, PMOVE, DSAPMD
30	Page number too large.	ADDREF, UPDATE, PICTUR
31	Mark not on specified page.	UPDATE, TBALL*
35	Illegal lamp number.	LAMPON, LAMPOF
37	Number of words to be transferred exceeds max of 20.	REQIM
38	PED number out of range.	DTINIT*, DTMODE*
40	Current refresh page is full.	MOVE, DRAW
41	Current refresh page is full.	TEXT, NEWLIN
42	Current refresh page is full.	CIRCLE ELIPSE
43	Current refresh page is full.	POINT*, VTPLOT, HTPLOT, XYPLOT
44	Current refresh page is full.	PICTUR
45	Current refresh page is full.	CPARM*
46	Current refresh page is full.	TPARM, LMARGN
47	Current refresh page is full.	STATUS

CODE	DESCRIPTION	ROUTINE CALLED WHEN ERROR OCCURRED
49	Current refresh page is full.	REFDAT
50	Current refresh page is full.	PDRAW, PMOVE
51	Color out of range or refresh page is full.	SCOLOR
52	Mode is out of range.	MOVE, DRAW, XYPLOT
58	Device number is out of range.	ENBCUR
62	Start mark is greater than end mark.	COPYIM, MOVIN
63	End mark exceeds end of page.	COPYIM, MOVIM
64	Exceeded page boundary.	COPYIM
70	Current refresh page is full.	MOV3D, DRAW3D
71	Current refresh page is full.	CCBLK
72	Cannot input from and output to same page.	T3D2D
77	Illegal monitor number.	SPLIT
81	Page 1 will not fit.	LAYOUT
82	Number of pages is not within range.	LAYOUT
83	Available refresh exceeded.	LAYOUT
91	Current refresh page is full.	FILL
92	Current refresh page is full.	COLORI, GRAYI

*Graphic 7 Only Routines

APPENDIX D

CONVERSION OF OLD FSP PROGRAMS

Programs which have already been written to run with FSP on a GRAPHIC 7 system can be made to run with FSP on the GRAPHIC 8 with a minimal number of changes. The necessary changes are described below.

1. Remove all calls to the following routines:

   a. ENBPEN
   b. DSAPEN
   c. ITEM
   d. GETPEN
   e. ENBPXY
   f. DSAPXY
   g. REQPXY
   h. GETPXY
   i. DPARM
   j. HCOPY
   k. CCINIT
   l. CCVAL
   m. CCON
   n. CCOFF
   o. COLOR

2. The following event types will not be returned from the EVENT routine:

   5 - photopen detect
   6 - photopen X, Y found
   10 - hardcopy complete

3. These routines exist only for backwards compatibility with the GRAPHIC 7.

   a. GSS4 - INITIALIZE
   b. POINT - DRAW A POINT PLOT
   c. REQMRK - REQUEST MARK POSITION
   d. CPARM - SET UP CHARACTER PARAMETERS
   e. COLOR - CHANGE COLOR
   f. DTINIT - INITIALIZE PED AS A DATA TABLET
   g. DTMODE - CHANGE DATA TABLET MODE
   h. TBALL - INITIALIZE PED AS A TRACKBALL
   i. DISTB - DISABLE PED DEVICE
   j. ENBPEN - ENABLE PHOTOPEN
   k. DSAPEN - DISABLE PHOTOPEN
   l. ITEM - DEFINE A GRAPHIC ITEM
   m. GETPEN - GET PHOTOPEN EVENT INFORMATION
   n. ENBXY - ENTER MULTIPLE PHOTOPEN SCAN MODE
   o. DSAXY - LEAVE MULTIPLE PHOTOPEN SCAN MODE

p. REQPXY - REQUEST SINGLE PHOTOPEN SCAN
q. GETPXY - GET PHOTOPEN SCAN EVENT DATA
r. HCOPY - INITIATE HARD COPY
s. CCINIT - INITIALIZE COORDINATE CONVERTER
t. CCVAL - ACTIVATE THE COORDINATE CONVERTER WITH SPECIFIC VALUES
u. CCON - TURN ON THE COORDINATE CONVERTER
v. CCOFF - TURN OFF THE COORDINATE CONVERTER

```
FSPT1    FORTRAN  V.5A(621) /KI      11-FEB-81      13:31                    PAGE 1

 C   EXAMPLE 1 FOR FSP8
 C
 C   USING AN ELIPSE AS A PED SYMBOL TEST ELLIPSE AND PED ROUTINES

       DIMENSION  FPTS(8)
       DIMENSION  LPCS(4)
       DATA FPTS  /10.,10.,10.,-10.,-10.,-10.,-10.,10./
       DATA ICSIZE,CHSPAC,RLSPAC,ICROT   /1,20.,40.,0/
 C     DATA LPGS   /256.,100.,100.,100./
 C   INITIALIZE
 C
       IPAGES = 4
       CALL INIT(5,0,2)
       CALL LAYOUT(IPAGES,LPGS)
       CALL SCALE(0.0,0.0,1000.0,1000.0)
       CALL ENBBOX(15)
       CALL ENBERR(15)
 C
 C   DISPLAY TITLE & SET UP CHARACTER PARAMETERS
 C
       CALL MOVE(150.0,800.0,0)
       CALL  LMARGN
       CALL TPARM(3,40.0,60.,ICROT)
       CALL SETEXT('      FSP8',11)
       CALL NEWLIN
 C     CALL SETEXT('PED & ELLIPSE TEST',10)
 C   SET UP PED SYMBOLS IN PAGE 2
 C
       CALL ADDREF(2)
       CALL MOVE(500.,500.,0)
       CALL ELIPSE(20.,40.,0)
       CALL ELIPSE(40.,20.,0)
       CALL FILL(4,FPTS,1)
 C
 C   NOW DISPLAY IT
 C
       CALL ADDREF(1)
       CALL PICTUR(2)
       CALL ENDPED(1,0,2,0)
 C
 C   LOOK FOR PED OR KEYBOARD EVENTS
 C
 10    CALL EVENT(1)
       GO TO (10,10,10,30,10,10,10,10,10,10,10,10,10,10,10,10)I
       GO TO 90
```

```
C
C   KEYBOARD PROCESSOR
C
30      CALL GETKEY(1,IKEY)
        IF ( IKEY  .LE.  15)  GO TO 40
        IKEY = IKEY    15
        GO TO (10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,90)IKEY
        GO TO 90
C
C
C
40      TO TO 10
C
C   END THE PROGRAM
C
90      CALL THEEND
        STOP
        END
```

APPENDIX E

PROGRAMMING EXAMPLES

EXAMPLE 1

Example 1 illustrates the use of INIT, LAYOUT, SCALE, THEEND.

NOTE

The CALL to INIT expects a parallel interface to be used.

```
C
C       EXAMPLE PROGRAM 1
C
        DIMENSION LPAGES(5)
        DATA LPAGES/1000, 100, 100, 10, 10/

C       INITIALIZE AND USE LOGICAL UNIT NUMBER 5.  TRANSMISSION
C       WILL BE OVER THE PARALLEL INTERFACE
C
        CALL INIT(3,0,1)
C
C       SPECIFY 5 PAGES OF USER DATA, EACH LENGTH BEING AS
C       DESCRIBED IN THE "LPAGES" DATA STATEMENT ABOVE.
C
        CALL LAYOUT(5,LPAGES)
C
C       SPECIFY ORIGIN AT THE LOWER-LEFT CORNER AND A LENGTH
C       AND WIDTH OF 1024.
C
        CALL SCALE(0.0,0.0,1023.0,1023.0)
C
C       (THE BODY OF THE DISPLAY PROGRAM GOES HERE)
C
C
C       WE ARE DONE, SHUT DOWN THE DISPLAY.
C
        CALL THEEND
C
C       EXIT THE PROGRAM
C
        CALL EXIT
        END
```

EXAMPLE 2

This sample program, when executed, displays figure E-1 on the screen. It illustrates user calls to the following subroutines.

INIT

LAYOUT

SCALE

ADDREF

MOVE

DRAW

XYPLOT

PICTUR

SETEXT

EVENT

GETKEY

TPARM

LMARGN

NEW LIN

In using these routines, it illustrates page linking and also the programming interaction with peripheral devices.

NOTE

After picture is displayed, to return to the TTY emulator, hit function key F15.

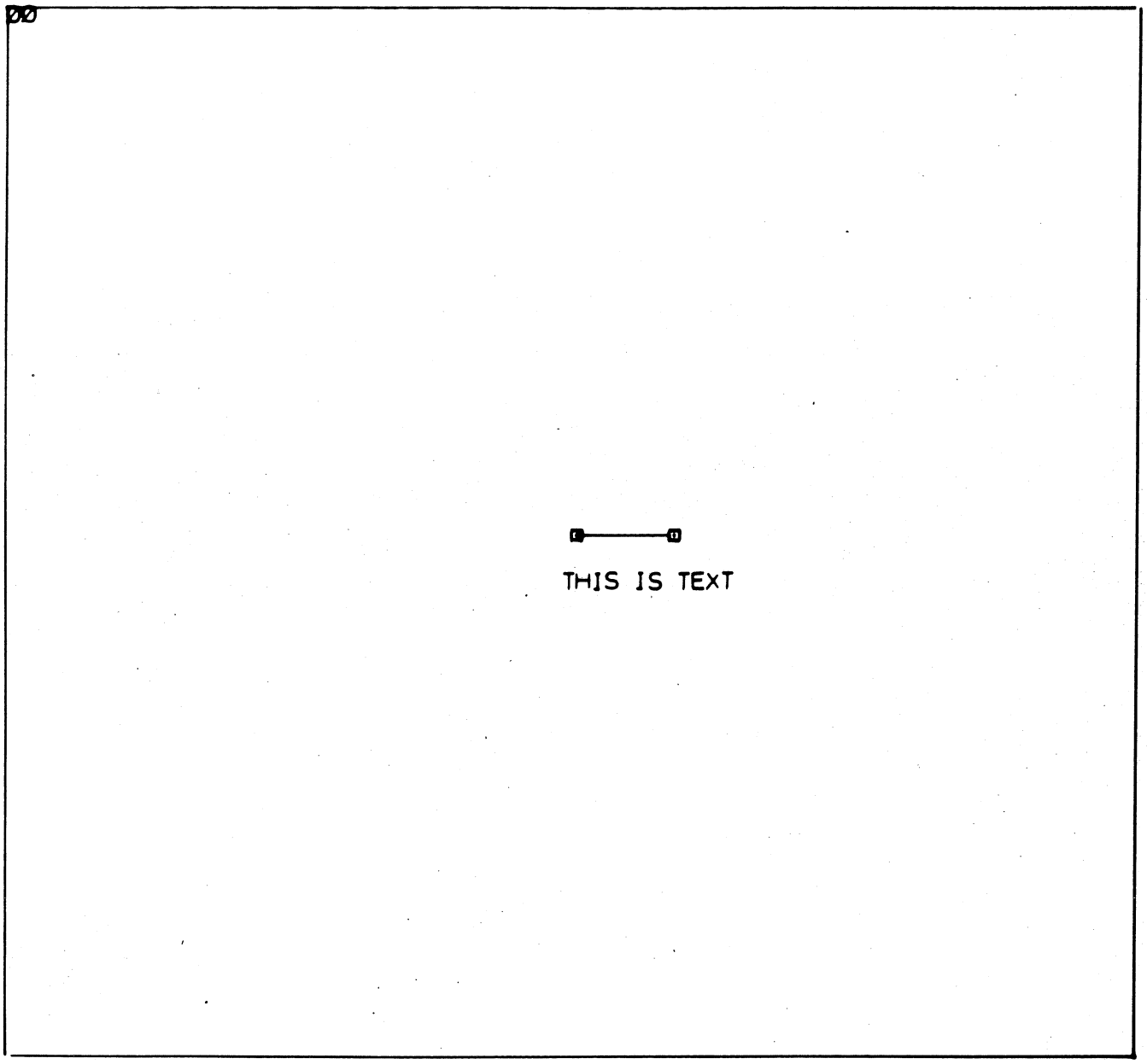


Figure E-1

```
00100  C       *** EXAMPLE 2 ***
00200  C  THE FOLLOWING EXAMPLE ILLUSTRATES THE USE OF ADDREF,MOVE,DRAW,SETEXT,
00300  C  XYPLOT,NEWLIN,LMARGN,PICTUR,EVENT,GETKEY,TPARM
00400  C  NOTE: TEXT IS DISPLAYED VIA A SUBROUTINE(SETEXT) UNIQUE TO THE HOST
00500  C  COMPUTER
00600  C       DIMENSION LPAGES(5),ITEX(6),XARRAY(10),YARRAY(10)
00700  C       DATA LPAGES /1000,100,100,10,10/
00800  C
00900  C  INITIALIZE AND USE LOGICAL UNIT NUMBER 5". TRANSMISSION
01000  C  WILL BE OVER THE SERIAL INTERFACE
01100  C
01200  C       CALL INIT(5,0,2)
01300  C
01400  C  SPECIFY 5 PAGES OF USER DATA, EACH LENGTH
01500  C  BEING AS DESCRIBED IN THE 'LPAGES' DATA STATEMENT ABOVE.
01600  C
01700  C       CALL LAYOUTS(5,LAPGES)
01800  C
01900  C  SPECIFY ORIGIN AT THE LOWER LEFT CORNER AND A LENGTH
02000  C  AND WIDTH OF 500
02100  C
02200  C       CALL SCALE(0,0,0,0,500.,500.)
02300  C
02400  C
02500  C  OPEN PAGE 1 FOR DISPLAYING REFRESH
02600  C
02700  C       CALL ADDREF(1)
02800  C
02900  C  SET CURRENT POSITION TO CENTER SCREEN AND SET CHARACTER PARAMETERS
03000  C  CHARSIZE=1,CHAR SPACE=10.,LINE SPACE=15.,ROTATION=0
03100  C
03200  C       CALL MOVE(250.,250.,0)
03300  C       CALL CPARM(1,10.,15.,0)
03400  C  DRAW A SHALL BOX IN PAGE 2, 10 UNITS BY 10 UNITS.
03500  C  NOTE THAT THESE MOVES AND DRAWS ARE ALL RELATIVE
03600  C
03700  C       CALL ADDREF(2)
03800
03900
04000
04100           CALL MOVE(5.,5.,1)
04200
04300
04400
04500           CALL DRAW(0.,-10.,1)
04600
04700
04800
04900
05000           CALL DRAW(-10.,0.,1)
05100
05200
05300
05400           CALL DRAW(0.,10.,1)
```

```
05500
05600
05700
05800          CALL DRAW(10.,0.,1)
05900  C
06000  C  RETURN TO CENTER
06100  C
06200          CALL MOVE(-5.,-5.,1)
06300  C  PUT A POINT IN CENTER OF BOX USING XYPLOT WITH RELATIVE COORDINATES
06400          XARRAY(1)=0
06500          YARRAY(1)=0
06600          CALL XYPLOT(1,XARRAY,YARRAY,1)
06700  C
06800  C  NOW GO BACK TO BUILDING PAGE 1
06900  C
07000          CALL ADDREF(1)
07100  C
07200  C  INSERT A SUBROUTINE CALL TO THE 10x10 BOX.  THIS WILL
07300  C  SHOW IT IN THE CENTER OF THE SCREEN.
07400  C
07500          CALL PICTUR(2)
07600  C
07700  C  DRAW A LINE FROM THE CENTER OF THIS BOX TO THE RIGHT 80 UNITS
07800  C
07900          CALL DRAW(330,250,0)
08000  C
08100  C  NOW DRAW ANOTHER 10x10 BOX HERE.
08200  C
08300          CALL PICTUR(2)
08400  C
08500  C       PLACE SOME TEXT JUST BELOW THE BOXES.
08600  C
08700  C       CALL MOVE(250.,230.,0)
08800          CALL LMARGN
08900          CALL SETEXT('THIS IS TEXT',12)
09000          CALL NEWLIN
09100  C       CALL SETEXT('FOR GRAPHIC 8 FSP',17)
09200  C  PICTURE SHOULD BE FINISHED NOW
09300  C
09400     USE THE EVENT ROUTINE TO SEARCH FOR FUNCTION KEY F15 STRIKE TO EXIT
09500     IF I IS NOT A4 THEN THE INTERRUPT WAS NOT FROM THE FUNCTION KEYBOARD ...
09600     IGNORE IT.
09700  C
09800  100     CALL EVENT(1)
09900
10000
10100
10200
10300
10400
10500          IF(1.NE.4)GO TO 100
10600
10700
10800
```

```
10900           CALL GETKEY(KBD,KEY)
11000
11100
11200
11300
11400           IF(KEY.NE.31)GO TO 100
11500   C
11600   C  WILL COME HERE IF KEY 31 WAS HIT, SO NOW WE ARE DONE,
11700   C  SHUT DOWN THE DISPLAY AND RETURN TO THE TTY EMULATOR,
11800           CALL THEEND
11900           END
```

APPENDIX F

PRODUCT PERFORMANCE REPORT

Occasionally, problems may be encountered in the use of products delivered to our customers. These problems or errors should be identified and communicated to Sanders Associates, Information Products Division by means of a Product Performance Report (PPR).

Product Performance Reports should be submitted to Sanders Associates. An appropriate specialist will review your PPR and attempt to resolve the problem or offer a temporary circumvention.

Every PPR is acknowledged upon receipt and answered in writing.

In preparing a PPR, the following guidelines should be followed for accurate and timely service to your problem.

1. Give as complete a description as possible of the problem encountered. Often a detail that may seem irrelevant will give a clue to solving the problem.

2. If possible, isolate the problem to a small example or procedure. This will make it easier for the specialist to duplicate the problem.

3. Include whatever documentation is possible, i.e., program listings, computer output or sample input. Annotations in a listing pointing to the error are very helpful.

PRODUCT PERFORMANCE REPORT

Page ___ of ___

Submit To:

CDMO
INFORMATION PRODUCTS DIVISION
SANDERS ASSOCIATES, INC.
DANIEL WEBSTER HIGHWAY, SOUTH
NASHUA, NEW HAMPSHIRE 03061

PPR #:

(assigned by the PPR center)

Product Identification and Version (or document)		Operating System & Version	Date
		Report Type	Priority
Name: Company: Address: Zip:		☐ Logic error ☐ Documentation ☐ Suggestion ☐ Inquiry ☐ Software ☐ Firmware ☐ Hardware Is the problem reproducible? ☐ Yes ☐ No	☐ Low ☐ Standard ☐ High
Phone:			
CPU:	Host-G8 interface:	Attached documents:	Distribution media:

Description:

PPR Center use only

Date received:	Date resolved:	
To specialist:		

THE INTENT AND PURPOSE OF THIS PUBLICATION IS TO PROVIDE ACCURATE AND MEANINGFUL INFORMATION TO SUPPORT EQUIPMENT MANUFACTURED BY SANDERS ASSOCIATES, INC. YOUR COMMENTS AND SUGGESTIONS ARE REQUESTED.

PLEASE USE THE FORM ON THE REVERSE SIDE TO REPORT ANY PROBLEMS YOU HAVE HAD WITH THIS PUBLICATION OR THE EQUIPMENT IT DESCRIBES.

FOLD FOLD

FIRST CLASS
PERMIT NO. 568
NASHUA, N.H.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by

Sanders Associates, Inc.
Information Products Division
Daniel Webster Highway, South
P.O. Box 868
Nashua, NH 03061

ATTN: DEPARTMENT 1-2894 (NHQ 1-447)

FOLD FOLD

Information Products Division
Federal Systems Group

SANDERS

Name: _____

Company: _____

Address: _____

_____

Telephone: [ ] _____

Date: _____

Sanders Equipment _____

Part Number _____

Software/Firmware System _____

Version _____

Host computer _____

Host operating system _____ Version _____

Host-GRAPHIC 8 interface _____

My problem is: hardware ☐ software ☐

firmware ☐ manual ☐

Description of problem (or suggestion for improvement):

Related tech manual number _____