H-79-0347

Graphic7™

COMPUTER GRAPHICS DISPLAY SYSTEM

FORTRAN SUPPORT PACKAGE [FSP] USER'S MANUAL

SA
SANDERS
ASSOCIATES, INC. 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.

Reprint - October 1980 - 100 Copies

Reprint-February 1981-100 Copies

TABLE OF CONTENTS

Section 1

Introduction

TABLE OF CONTENTS (Cont)

Section 9

Setup Routines

Section 10

Image Generation Routines

Section 11

Page Management Routines

Section 12

Status Routines

Section 13

Event Routine

Section 14

Alphanumeric Keyboard Routines

Section 15

Photopen Item Routines

Section 16

Photopen Scan Routines

TABLE OF CONTENTS (Cont)

Section 17

Trackball/Forcestick/Data Tablet Routines

Section 18

Miscellaneous Routines

Section 19

Packed Vector Mode

Section 20

Coordinate Converter Routines

TABLE OF CONTENTS (Cont)

Section 21

Image Control Routines

Section 22

FSP Input/Output

Section 23

Deliverable Items

Section 24

Installation Procedure

Section 25

Startup Procedure

APPENDICES

SECTION 1

INTRODUCTION

This Programmers Reference Manual for the Fortran Support Package (FSP) is provided by Sanders in support of its GRAPHICS 7 interactive display terminal.

1.1 SUBROUTINE CONCEPT

FSP is a collection of 62 Fortran-callable subroutines. The routines require little knowledge of the GRAPHIC 7 terminal, yet allow the user maximum utilization of its interactive capabilities.

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 7 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 Enhanced (GCP+), which is resident in read-only memory in the GRAPHIC 7.

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 7 terminal via the MSGOUT subroutine (provided by the customer). GCP+ in the GRAPHIC 7 processes the message to produce the desired results.

FSP also receives and interprets messages from GCP+ in response to a POLL request. These messages contain PHOTOPEN, 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 9 for a more detailed description of these routines.

Error detection is also available under program control. When the user calls to 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 18 for a detailed description of the GETERR routines.

Error codes are defined in Appendix C.

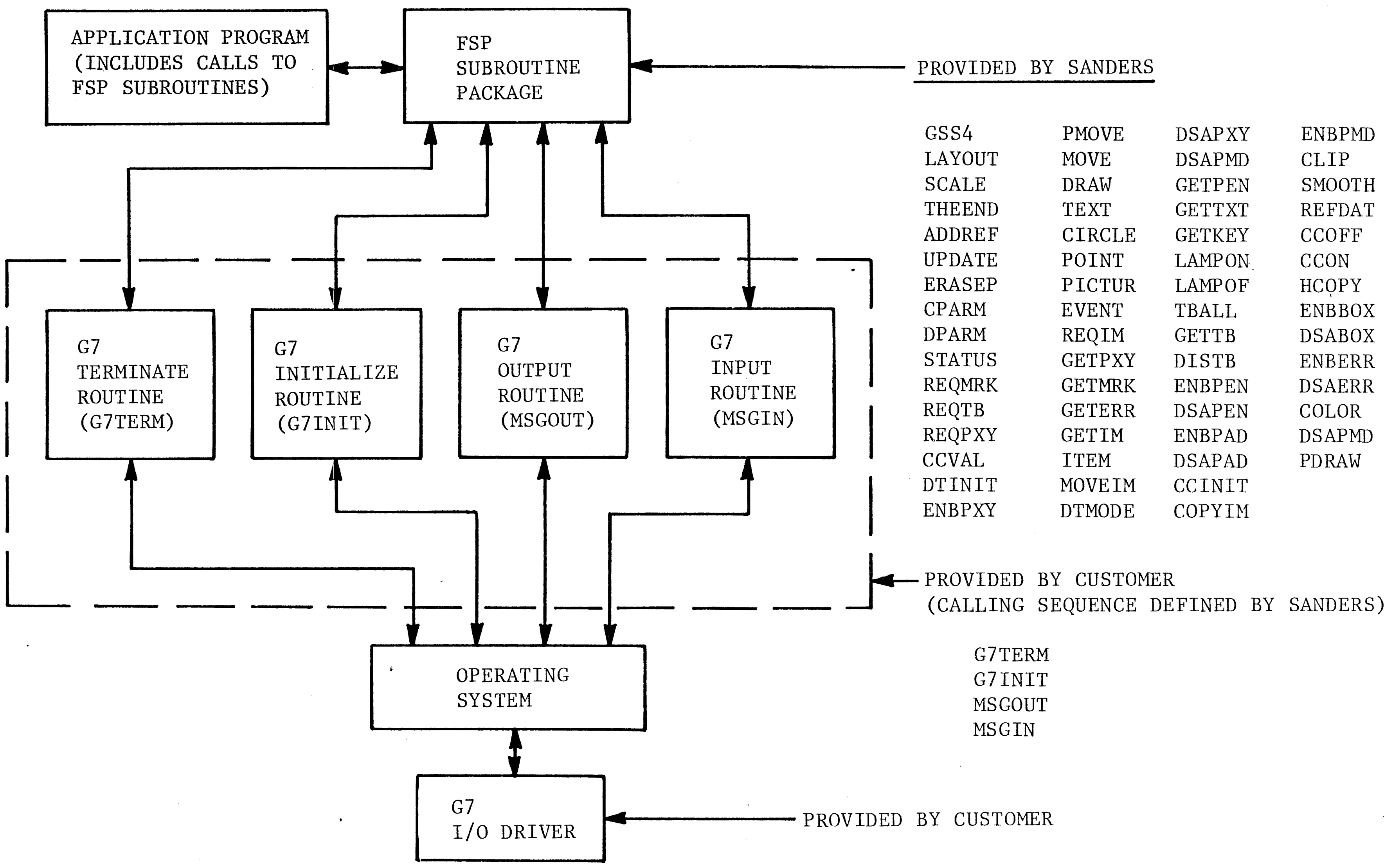

APPLICATION PROGRAM (INCLUDES CALLS TO FSP SUBROUTINES)
FSP SUBROUTINE PACKAGE
PROVIDED BY SANDERS
GSS4
LAYOUT
SCALE
THEEND
ADDREF
UPDATE
ERASEP
CPARM
DPARM
STATUS
REQMRK
REQTB
REQPXY
CCVAL
DTINIT
ENBPXY
PMOVE
MOVE
DRAW
TEXT
CIRCLE
POINT
PICTUR
EVENT
REQIM
GETPXY
GETMRK
GETERR
GETIM
ITEM
MOVEIM
DTMODE
DSAPXY
DSAPMD
GETPEN
GETTXT
GETKEY
LAMPON
LAMPOF
TBALL
GETTB
DISTB
ENBPEN
DSAPEN
ENBPAD
DSAPAD
CCINIT
COPYIM
ENBPMD
CLIP
SMOOTH
REFDAT
CCOFF
CCON
HCOPY
ENBBOX
DSABOX
ENBERR
DSAERR
COLOR
DSAPMD
PDRAW
G7 TERMINATE ROUTINE (G7TERM)
G7 INITIALIZE ROUTINE (G7INIT)
G7 OUTPUT ROUTINE (MSGOUT)
G7 INPUT ROUTINE (MSGIN)
PROVIDED BY CUSTOMER
(CALLING SEQUENCE DEFINED BY SANDERS)
G7TERM
G7INIT
MSGOUT
MSGIN
OPERATING SYSTEM
G7 I/O DRIVER
PROVIDED BY CUSTOMER

Figure 1-1

SECTION 2

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 7 terminal.

3. FSP is machine independent.

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

5. Windowing of user data including:

   a. Data scaling: The conversion of user coordinates to refresh coordinates and vice versa.

   b. Image scissoring: Truncating portions of a display that extend beyond the screen boundaries.

6. Modifying images presently displayed (selective updating).

7. Each copy of FSP in the host supports one GRAPHIC 7 controller with four CRT indicators, two keyboards, two trackballs or data tablets, two PHOTOPENs, conic generator, and 2D coordinate converter.

8. Operator interactions with application program:

   a. Alphanumeric keyboard

   b. Function keys

   c. Trackball, forcestick, or data tablet

   d. PHOTOPEN

9. Generation of all refresh instructions including image generation commands (MOVE, DRAW, CIRCLE, POINT, TEXT).

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. Local keyboard manipulations performed at the terminal.

    a. Characters typed directly into a refresh scratchpad.

    b. Scratchpad area can be edited from the keyboard.

13. Local PHOTOPEN operations performed at the terminal.

    a. PHOTOPEN finder - The position of the PHOTOPEN on the screen is determined by the GCP+, by flashing a grid pattern locating the PHOTOPEN position.

14. Mass transfer of existing refresh data to the terminal. This allows for off line generated refresh code to be passed directly to the GRAPHIC 7 terminal and inserted into the refresh memory without any additional processing.

15. All floating point arithmetic processing of FSP is done in the host computer. The GRAPHIC 7 GCP+ performs fixed point arithmetic.

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

    a. Initial or additional data.

    b. Editing data (selective updating).

17. Hard copy capability. The application program can request that the image on the screen be hard copied on the Sanders 570 Hard Copy Unit.

18. Displayed images can be rotated and translated on the CRT. Four subroutines exist for manipulating the 2D coordinate converter hardware option.

19. All position data transmitted between host and GRAPHIC 7 is in screen coordinates.

SECTION 3

FSP DISTRIBUTED PROCESSING

This section describes how graphics tasks are distributed between FSP in the host and GCP+ in the terminal.

3.1 FSP Processing

1. All floating point conversion.

   a) Scaling: conversion of user floating point coordinates to display coordinates.

   b) Windowing: zooming and offsetting.

2. Scissoring: the 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 7 terminal.

5. Receiving and converting all messages from the GRAHPIC 7 terminal to a manageable form for Fortran. This includes converting screen coordinates to floating point user coordinates.

6. Controls refresh file management, LAYOUT.

3.2 GCP+ Processing

1. Receives messages from the host computer.

2. Processes messages from the host computer.

3. Handles PED manipulations and symbol.

4. Finds the PHOTOPEN position on a blank screen.

5. Displays alphanumeric keyboard inputs on the screen in a predefined scratch pad area.

6. Handles editing of text displayed in the scratch pad.

7. Formats all messages to the host computer.

8. Services all display interrupts.

9. Services all display peripheral devices.

10. Performs validation test and diagnostics.

SECTION 4

FSP SUBROUTINE LIBRARY

The FSP subroutines can be categorized as follows:

A. Setup Routines

1. GSS4
2. LAYOUT
3. SCALE
4. ENBBOX
5. DSABOX
6. ENBERR
7. DSAERR
8. THEEND

B. Image Generation Routines

1. MOVE
2. DRAW
3. TEXT
4. POINT
5. CIRCLE
6. REFDAT

C. Page Management Routines

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

D. Status Routines

1. CPARM
2. DPARM
3. STATUS
4. LAMPON
5. LAMPOF
6. COLOR

E. Event Routine

1. EVENT

F. Alphanumeric/Function Keyboard Routines

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

G. Photopen Item Routines

1. ENBPEN
2. DSAPEN
3. ITEM
4. GETPEN

H. Photopen Scan Routines

1. ENBPXY
2. DSAPXY
3. REQPXY
4. GETPXY

I. Trackball/Forcestick/Data Tablet Routines

1. TBALL
2. DISTB
3. DTINIT
4. DTMODE
5. REQTB
6. GETTB

J. Miscellaneous Routines

1. HCOPY
2. REQIM
3. GETIM
4. GETERR

K. Packed Vector Routines

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

L. Coordinate Converter Routines

1. CCINIT
2. CCVAL
3. CCON
4. CCOFF

M. Image Control Routines

1. CLIP
2. SMOOTH

SECTION 5

HARDWARE CONFIGURATIONS SUPPORTED

FSP supports either one or two display stations. A display station may have the following equipment:

- Monitor
- Slave monitor
- PHOTOPEN
- Trackball or forcestick or data tablet
- Alphanumeric/function keyboard
- Hardcopy

The basic FSP supports the following hardware in the terminal controller:

- Memory configurations up to 128K
- Character generator
- Vector/position generator
- Conic generator
- 2D coordinate converter

SECTION 6

PAGING CONCEPT

A GRAPHIC 7 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:

TOTAL MEMORY	USER REFRESH SPACE
8K	5K*
16K	11K*
32K	23K
64K	55K
80K	71K
96K	87K
128K	119K

*These memory configurations are exceptions to the above paragraph. For 8K systems, 2K of memory is set aside to support options. For 16K systems, 4K of memory is set aside to support options. On systems where no present or future option support is needed, modifications can be made to FSP to increase user refresh space to 7K or 15K in 8K or 16K memory systems.

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 7.
- A page may not cross bank boundaries.
- Page 1 exists entirely in memory bank 0.
- Page 1 is always refreshed and can be thought of as the "mainline" refresh program.

- Pages 2 and above are not always refreshed and may be thought of as refresh subroutines.

"Mark" Definition

- A mark is a relative pointer into a page.
- Each page has a corresponding mark pointer associated with it.
- Mark values range from 0 to 32K-4
  e.g.
  A mark value of 4 refers to the 5th memory location relative to the start of a page.
- The length of a page is defined in terms of 16-bit words.

The LAYOUT call (see paragraph 9.2) allows the caller to define Graphic pages (divide memory into sections).

The page and mark combination allows any memory location to be addressed by the FSP routines.

SECTION 7

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 9.3 for a detailed description of subroutine SCALE.

SECTION 8

USE OF LABELLED COMMON

FSP uses labelled common. The user should 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		279
COORD		9
PVMD		9
LAYOT		516
MAST		5
PERIPH		6
PER2		2
PEN		1
LMEM		11
	Total	838

SECTION 9

SETUP ROUTINES

The following subroutines are described in this section:

GSS4	-	Initialize the terminal to FSP mode.
LAYOUT	-	Define FSP memory layout in the GRAPHIC 7 terminal.
SCALE	-	Define user coordinate system.
ENDBOX	-	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 7. The GRAPHIC 7 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 GSS4. The GRAPHIC 7 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 and determines what image generation routines are called. 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 7 that it is no longer communicating with a FSP host application and to place it in teletypewriter emulation mode.

9.1 INITIALIZE THE TERMINAL TO FSP MODE

NAME: GSS4

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

CALLING FORMAT: CALL GSS4 (IUNIT, IDUM, IFACE)

DESCRIPTION OF PARAMETERS:

IUNIT = Integer variable containing the logical unit # assigned to the GRAPHIC 7 I/O driver. This value is supplied as an argument to subsequent G7INIT, G7TERM, MSGOUT, and MSGIN subroutine calls (see Section 22 for a more detailed description of these subroutines).

IDUM = Dummy argument (for expansion)

IFACE = Integer variable containing the type of hardware interface between the host and the GRAPHIC 7 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 GSS4 routine causes the customer-supplied G7INIT routine described in paragraph 22.1 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.

9.2 DEFINE FSP MEMORY LAYOUT IN THE GRAPHIC 7 TERMINAL

NAME: LAYOUT

FUNCTION: Partitions the memory in the GRAPHIC 7 into pages. This routine must be the second FSP routine called (GSS4 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.

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

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 23720 words; the maximum size of all other pages is 32763 words.

AUTOMATIC ALLOCATION

NPAGES = 0 (Supplied by caller and indicates automatic allocation requested.)

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:

• 8K,16K,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)
• 80K, 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

CONFIGURATION

NPAGES = -1 (Supplied by caller and indicates configuration request; no pages allocated.)

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 7 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 11.5). 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 GSS4, followed by a call LAYOUT.

Example

```
C
C    ALLOCATE 20,200 WORDS OF THE
C       GRAPHIC 7 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    CALL LAYOUT FOR 7 PAGES
C
     CALL LAYOUT (7, LNGARY)
C
C
C
```

9.3 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 integer 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.

9.4 TURN BORDER DISPLAY ON

NAME: ENBBOX

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

CALLING FORMAT: CALL ENBBOX (IND)

DESCRIPTION OF PARAMETERS:

IND - An integer variable indicating which of the four possible indicators the border is to be presented on.

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 DESCRIPTION

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

9.5 TURN BORDER DISPLAY OFF

NAME: DSABOX

FUNCTION: Allows the caller to remove the rectangular border from selected indicators

CALLING FORMAT: CALL DSABOX (IND)

DESCRIPTION OF PARAMETERS:

IND = An integer variable indicating which of the four possible border indicators to remove. See ENBBOX for the associated indicators values.

DETAILED DESCRIPTION:

Removes outline around the displayable area on selected indicators.

9.6 TURN ERROR DISPLAY ON

NAME: ENBERR

FUNCTION: Allows the caller to turn on the error display area

CALLING FORMAT: CALL ENBERR (IND)

DESCRIPTION OF PARAMETERS:

IND = An integer variable indicating which of the four possible error displays to present. See ENBBOX for the associated indicators values.

DETAILED DESCRIPTION:

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

9.7 TURN ERROR DISPLAY OFF

NAME: DSAERR

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

CALLING FORMAT: CALL DSAERR (IND)

DESCRIPTION OF PARAMETERS:

IND = An integer variable indicating which of the four possible error displays to remove. See ENBBOX for a list of the values for IND and the associated indicators.

DETAILED DESCRIPTION:

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

9.8 TERMINATE FSP MODE

NAME: THEEND

FUNCTION: Causes the GRAPHIC 7 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 7 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 a GSS4 call to put the terminal back into the FSP mode of operation.

SECTION 10

IMAGE GENERATION ROUTINES

The subroutines described in this section permit the application programmer to describe objects in user coordinates. The actual appearance of the objects on the GRAPHIC 7 display is determined by the following:

- The graphic orders created by the calls to the image generation routines described in this section.

  MOVE -- Move beam to the position specified

  DRAW -- Draw a line

  TEXT -- Display text characters

  POINT -- Display a point

  CIRCLE -- Draw a circle

  REFDAT -- Transfer a block of predefined graphic orders.

- The current value of display parameters which have been previously set by calls to the status subroutines:

  CPARM -- Character size, spacing, and orientation.

  DPARM -- Drawing and refresh rates

  STATUS -- Blinking, intensity, line type, and display CRT usage

  COLOR -- Color selection when applicable (red, yellow, green and orange)

- The current pages and areas of pages being displayed which have been defined by previous calls to page management routines.

  PICTUR -- Select pages to be displayed

  ERASEP -- Select area within page which is to be erased.

Examples in Appendix E illustrate various display images generated by calls to routines in this section and the effects that display parameter settings and page management activities have on these images.

10.1 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, 2, or 3)

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

Relative mode (MODE = 1)

Relative X, Y coordinate (deltas) to be moved from the current beam 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 = 3

3 = X, Y supplied is absolute and absolute graphic order is to be generated (similar to MODE = 0). It is expected, however, that subsequent calls to MOVE or DRAW will have MODE = 2.

DETAILED DESCRIPTION:

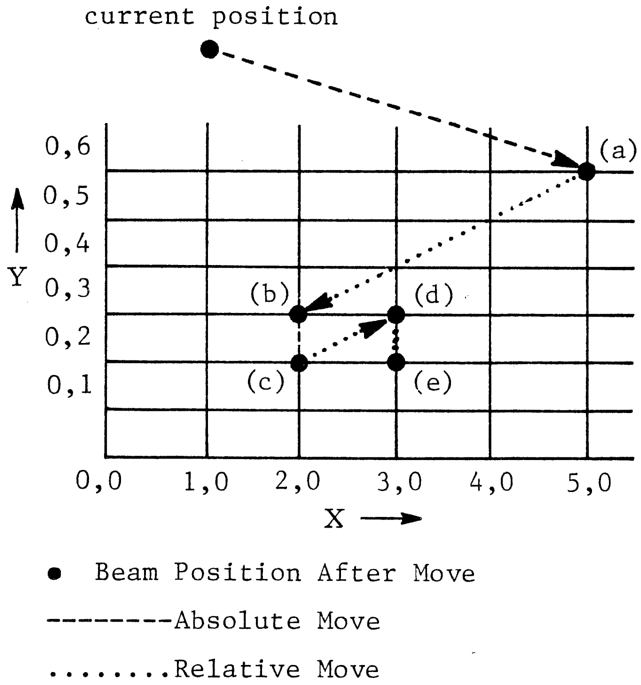
Mode = 2 and MODE = 3 are provided to allow a user whose data base contains only absolute X, Y coordinates to produce relative graphic orders without calculating the deltas.

```
Example:  C  The following call produces an absolute graphic order
          C  which moves the beam to (1,1)
                           CALL MOVE (1.,1.,3)
          C  The following call produces a relative graphic order
          C  which draws the beam 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 the beam 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). The TBALL routine described in paragraph 17.1 can be used to link to the absolute move and to locally move the entity around under control of the PED.

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
```



10.2 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 beam 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

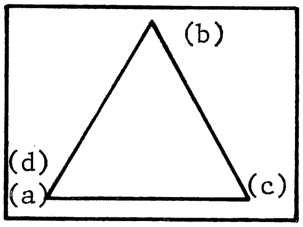
DETAILED DESCRIPTION:

The behavior of this routine is almost identical to the MOVE subroutine except that DRAW graphic orders are produced rather than MOVE graphic orders. Note, however, that MODE = 3 is not allowed for the DRAW routine.

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 and (4) the drawing rate. For color displays, a previous call to COLOR determines the color of the line.

Example

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


      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 (-4.0,0.,1)(d)
```

10.3 DISPLAY TEXT CHARACTERS

NAME: TEXT

FUNCTIONS: 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.

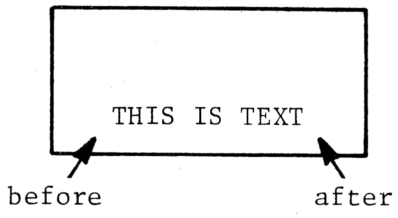
$$1 \leq N \leq 86$$

IARRAY = An integer array supplied by the caller in which each element of the array contains an 8 bit ASCII character code right adjusted in the element (see Appendix B for character codes).

DETAILED DESCRIPTION:

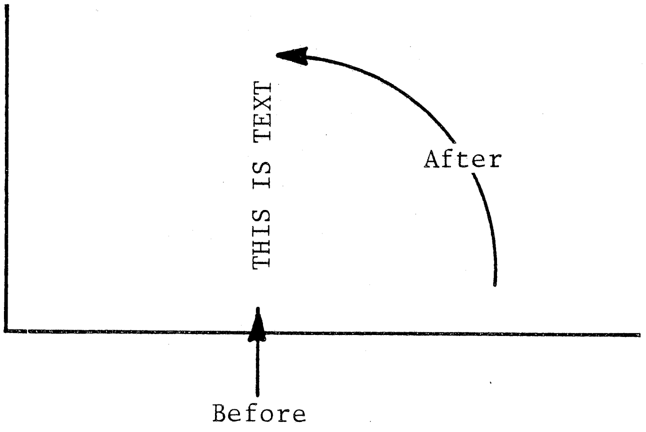
If N is odd, a null character is stored as the last text character. When the currently opened page is displayed, the GRAPHIC 7 displays text starting at the current position of the beam in either a horizontal or vertical direction with character size and spacing determined by a previous user call to CPARM. The text intensity, blink or no blink, and color (color displays only) has also been determined by calls to other FSP routines. The current beam position after the text is displayed is located at the location of the last text character drawn (blanks included)

```
C
C     NON-ROTATED TEXT
C

      CALL MOVE (1.,1.,0)
      CALL TEXT (12, IARRAY)
```



Beam Position

```
C
C     ROTATED TEXT
C

      CALL MOVE (1.,1.,0)
      CALL TEXT (12,IARRAY)
```



Beam Position

10.4 DISPLAY A POINT

NAME: POINT

FUNCTION: Generates a "point" graphic order and places it at the mark position of the currently opened page.

CALLING FORMAT: CALL POINT

DETAILED DESCRIPTION:

This call does not change the position of the beam but simply causes a point to appear at the current position of the beam.

Example:

```
C  Plot 3 horizontal points starting
C  at (512,512) along the positive X
C  axis with the spacing between points = 5
   CALL MOVE (512.,512.,0)
   CALL POINT
   CALL MOVE (5.,0.,1)
   CALL POINT
   CALL MOVE (5.,0.,1)
   CALL POINT
```

10.5 DRAW A CIRCLE

NAME: CIRCLE

FUNCTION: Allows the caller to display specified quadrants of a circle centered around the current position of the beam.

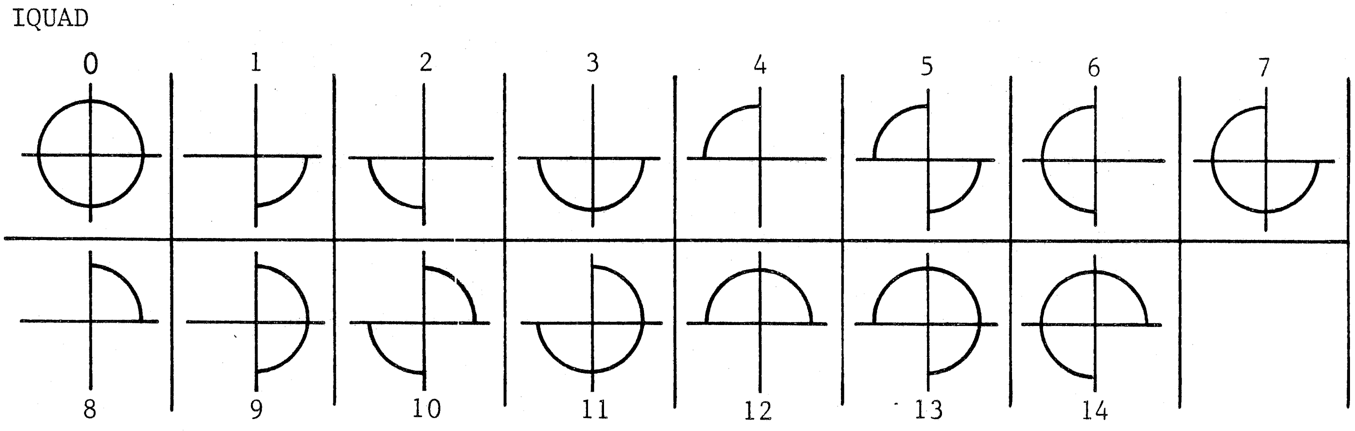
CALLING FORMAT: CALL CIRCLE (RADIUS, IQUAD)

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



DETAILED DESCRIPTION:

This routine places a "draw circle" or "draw quadrant(s)" graphic order at the mark position of the currently opened page. When the currently opened page is displayed, the GRAPHIC 7 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.

10.6 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.

SECTION 11

PAGE MANAGEMENT ROUTINES

The page management routines are used to select the memory address in the GRAPHIC 7 that will be used to store the next graphic instruction. The GRAPHIC 7 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, POINT, CIRCLE, REFDAT), FSP generates the equivalent graphic controller instructions which are sent to the GRAPHIC 7 and stored in the GRAPHIC 7 memory.

The page management routines consist of the following subroutines:

ADDREF - Open page for adding refresh data.

UPDATE - Open page for editing refresh data.

ERASEP - Erase from page mark to end of page.

PICTUR - Graphic subroutine call.

REQMRK - Request the present page mark.

GETMRK - Get mark request information.

MOVEIM - Move a block of 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 CRT indicator.

When the application program calls the LAYOUT subroutine, the GRAPHIC 7 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 7 memory would look as follows:

ADDRESS	GRAPHIC 7 MEMORY
0	Memory space used by GCP+
3000	Start of Page 1
5000	Start of Page 2
6000	Start of Page 3
6500	End of memory for use by FSP

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 6500 are unassigned and would be unavailable for storage of refresh data.

When refresh data is to be added to GRAPHIC 7 memory, the address is selected by adding the start address of the current page to the current mark. The following table indicates the GRAPHIC 7 memory address that would be selected, based on the current page and current mark.

CURRENT PAGE	CURRENT MARK	START ADDRESS OF PAGE	GRAPHIC 7 MEMORY ADDRESS
1	0	3000	3000
1	5	3000	3005
2	0	5000	5000
2	876	5000	5876
3	0	6000	6000
3	499	6000	6499
3	500	6000	*

*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 Ø to 499.

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 11-1.

NOTE

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

An FSP program which generates the display image in figure 11-1 is given below:

```
LINE NO.

   10        Call GSS4 (3, 0, 2)
   20        Call LAYOUT (3, LPAGES)
   30        Call SCALE (0.0, 0.0, 12.0, 12.0)
   40        Call ADDREF (1)
   50        Call MOVE (6.0, 6.0,0)
   60        Call DRAW (7.0, 5.5, 0)
   70        Call DRAW (8.0, 5.5, 0)
   80        Call TEXT (8, ITEXT)
   90        Call ADDREF (2)
  100        Call MOVE (-.5, -.5, 1)
  110        Call DRAW (1.0, 0., 1)
  120        Call DRAW (0., 1.0, 1)
  130        Call DRAW (-1.0, 0., 1)
  140        Call DRAW (0., -1.0, 1)
  150        Call ADDREF (3)
  160        Call MOVE (-.5, 0., 1)
  170        Call DRAW (1.0, 0., 1)
  180        Call MOVE (-.5, -.5, 1)
  190        Call DRAW (0, 1.0, 1)
  200        Call ADDREF (1)
  210        Call MOVE (1.5, 10.5, 0)
  220        Call PICTUR (2)
  230        Call MOVE (10.5, 10.5,0)
  240        Call PICTUR (2)
  250        Call MOVE (10.5, 1.5, 0)
  260        Call PICTUR (2)
  270        Call MOVE (1.5, 1.5, 0)
  280        Call PICTUR (2)
  290        Call MOVE (6.0, 8.0, 0)
  300        Call PICTUR (3)
  310        Call MOVE (6.0, 4.0, 0)
  320        Call PICTUR (3)
```

The call to GSS4 (line 10) initializes the FSP program and a full screen box and an error code value of "00" are displayed on the CRT indicator.

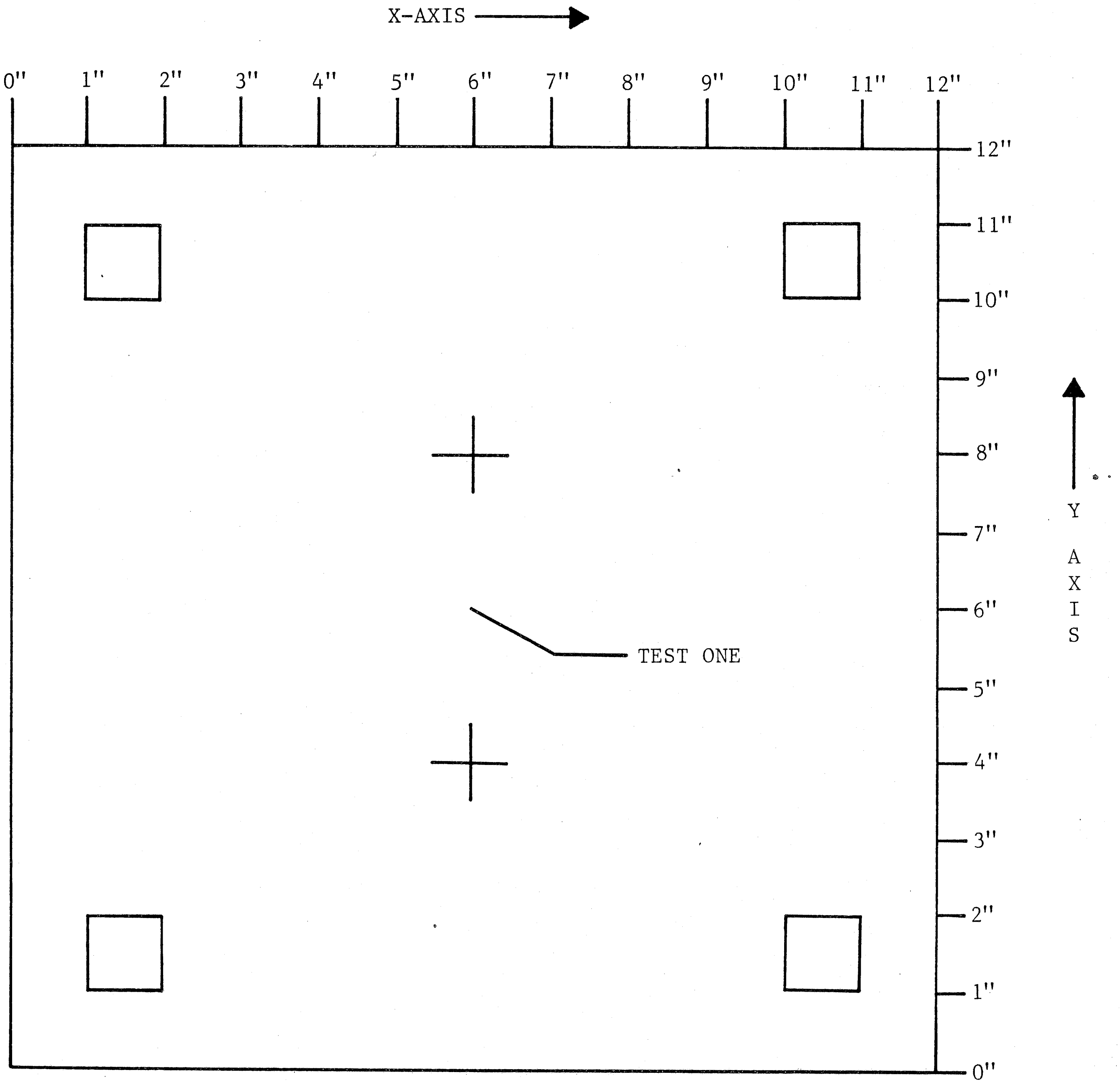
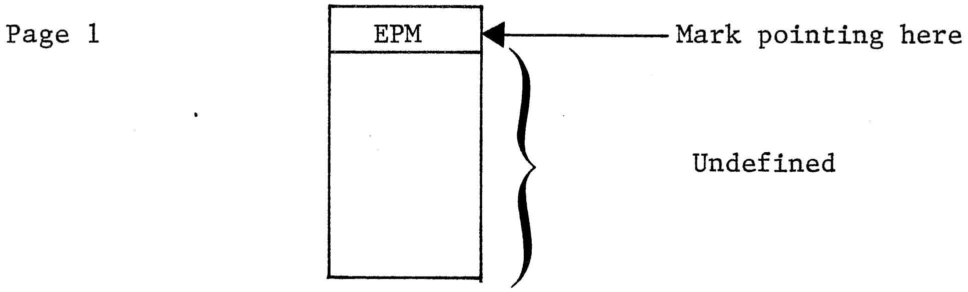


FIGURE 11-1

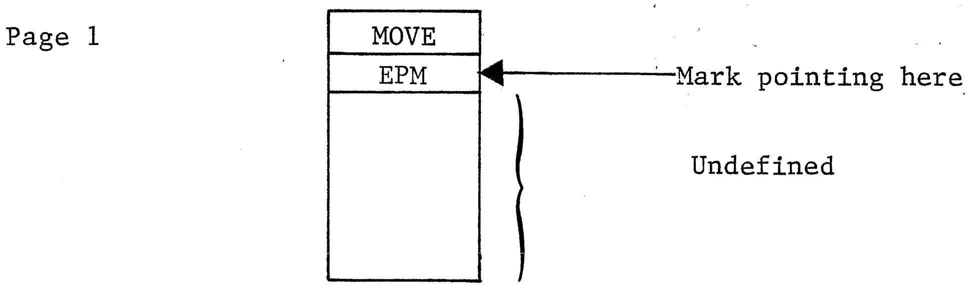
The call to LAYOUT (line 20) sections GRAPHIC 7 memory into three pages. The first word of each page (i.e., mark Ø) is set up to contain an end of page mark. The end of page mark is equivalent to a return statement in a subroutine.

The call to SCALE (line 30) sets up FSP to map all values in the range of "0" to "12" into the equivalent CRT coordinate system. The user coordinate system defines the lower left corner of the CRT as 0", 0" and the upper right hand corner as 12", 12". For the CRT 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 40) 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 7 memory address pointer to point to the first word (mark Ø) in page 1. After the call to ADDREF, page 1 looks as follows:



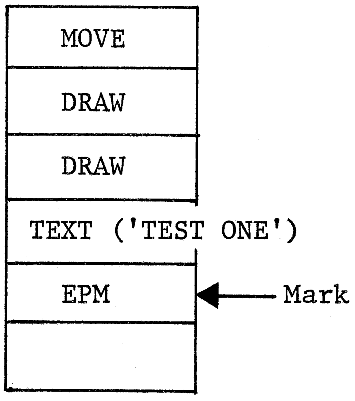
After the call to MOVE (line 50), page 1 looks as follows:



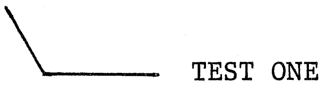
Note that after the call to MOVE, the mark value points to the new address in which the EPM is stored.

After the two calls to DRAW (lines 60 and 70) and the call to TEXT (line 80), page 1 looks as follows:

Page 1



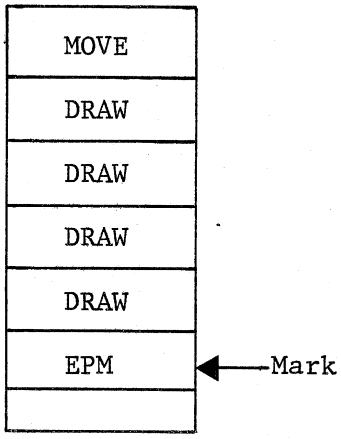
At this point the following is displayed on the CRT indicator.



The call to ADDREF (2) (line 90) opens up page 2 in the add mode. This call also sets up a GRAPHIC 7 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 100) and the 4 calls to DRAW (lines 110 to 140), page 2 looks as follows:

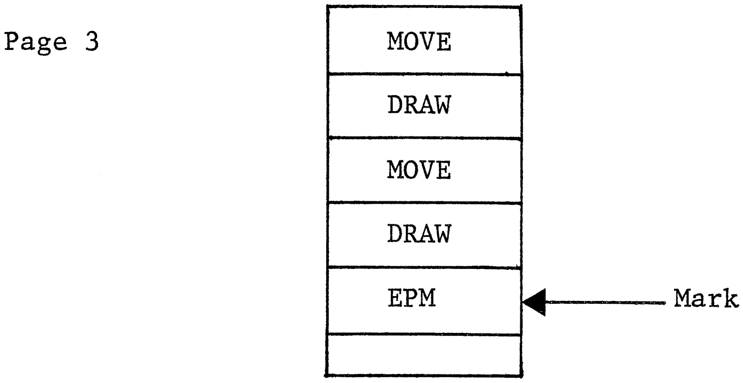
Page 2



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

The call to ADDREF (3) (line 150) opens up page 3 in the add mode. This call also sets up a GRAPHIC 7 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.

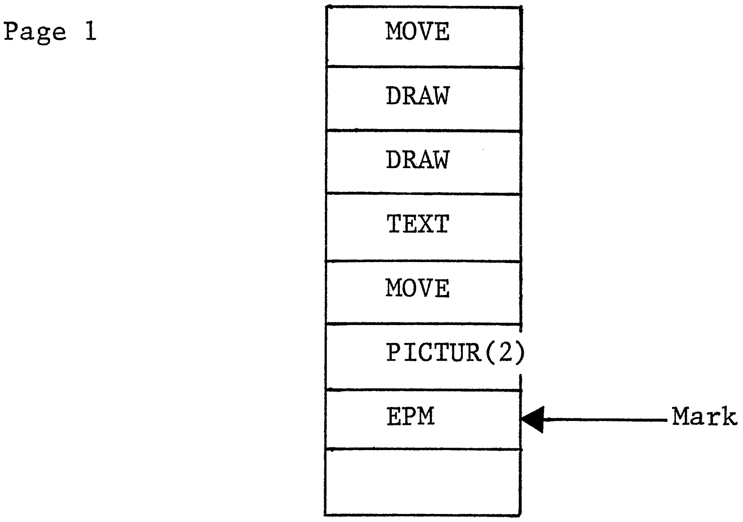
After the calls to MOVE, DRAW, MOVE, DRAW (lines 160 to 190), page 3 looks as follows:



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

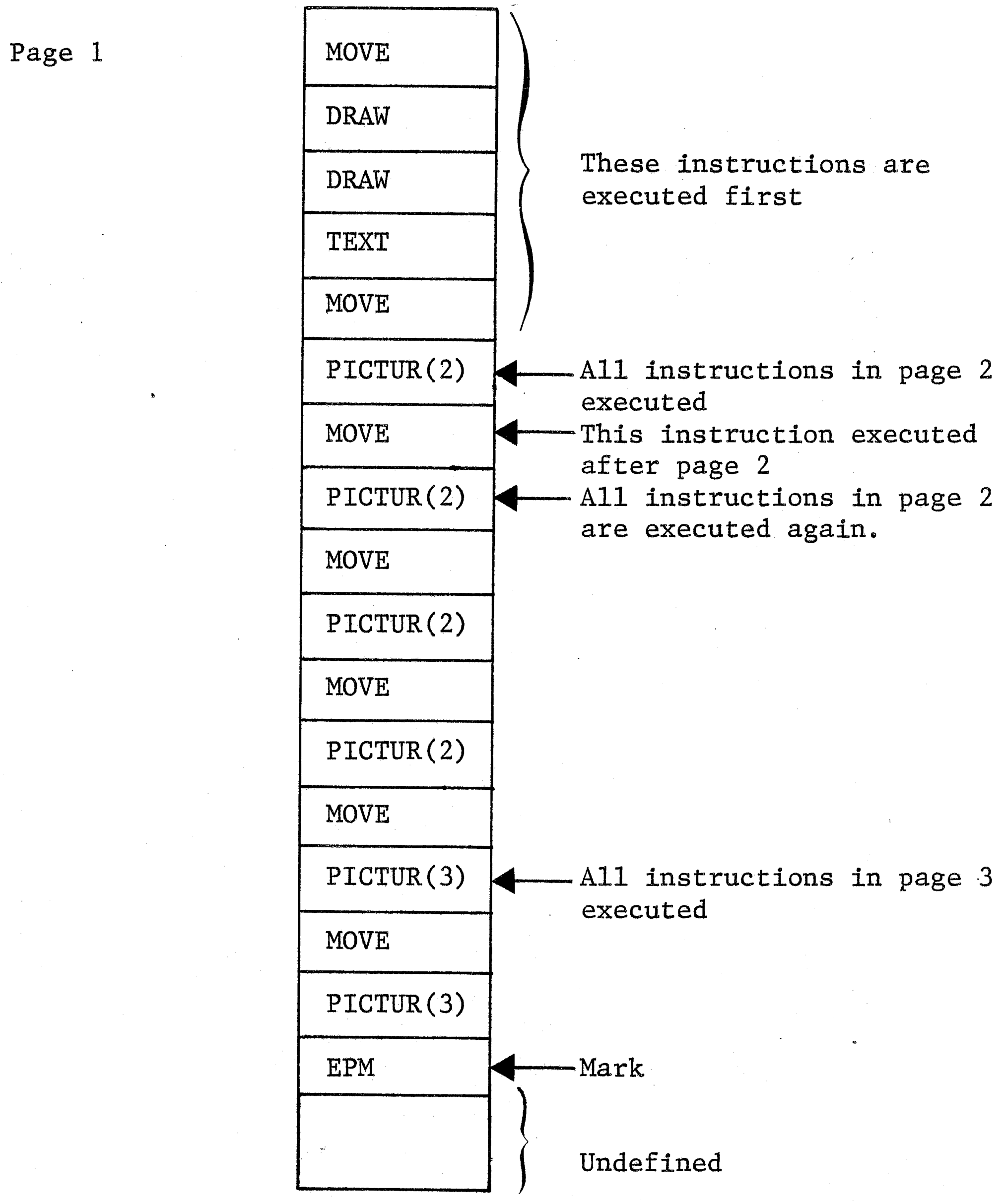
The call to ADDREF (1) at line 200 re-opens page 1 in the add mode. This call also sets up a GRAPHIC 7 memory address pointer to point to the last word in page 1 that contains the EPM. It also saves the last mark value associated with page 3.

After the calls to MOVE and PICTUR (2) at lines 210 and 220, page 1 looks as follows:

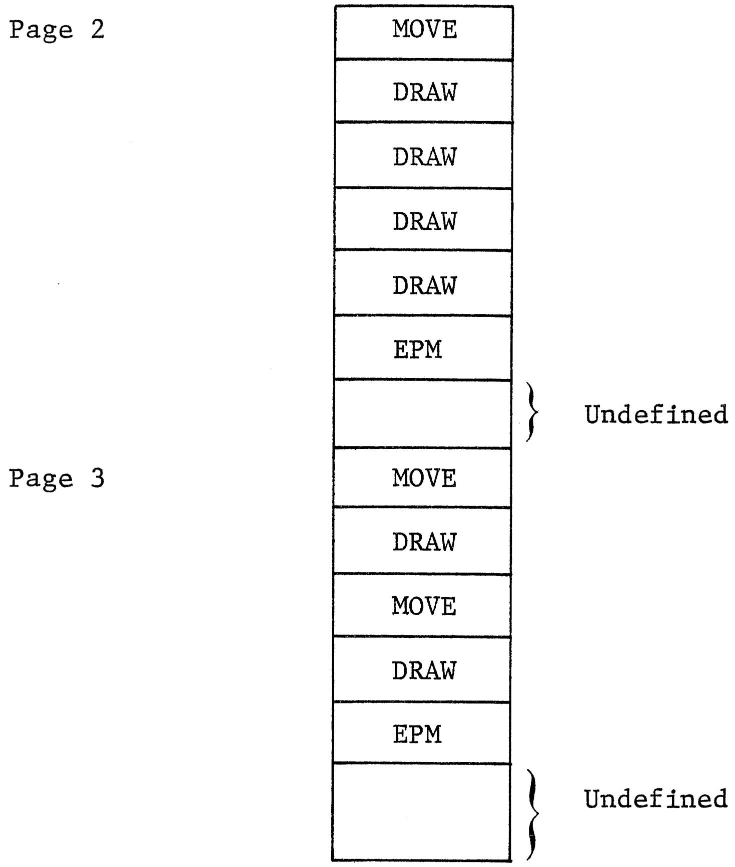


At this point a box is displayed at the top left side of the CRT indicator. The call to PICTUR (2) causes a subroutine call to be made to page 2. This causes the instructions in page 2 to be executed. When the EPM is executed in page 2, program control is returned back to page 1 (i.e., the EPM in page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

After the remaining statements in the FSP program (i.e., lines 230 to 320) are executed, the GRAPHIC 7 memory looks as follows:



Page 1	
MOVE	These instructions are executed first
DRAW	
DRAW	
TEXT	
MOVE	
PICTUR(2)	All instructions in page 2 executed
MOVE	This instruction executed after page 2
PICTUR(2)	All instructions in page 2 are executed again.
MOVE	
PICTUR(2)	
MOVE	
PICTUR(2)	
MOVE	
PICTUR(3)	All instructions in page 3 executed
MOVE	
PICTUR(3)	
EPM	Mark
	Undefined



At this point the image shown in figure 11-1 is displayed on the CRT indicator.

NOTE

> The page 1 instructions are the only instructions that are directly executed. All instructions in pages 2 and 3 are executed indirectly via the PICTUR subroutine linkage.

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'.

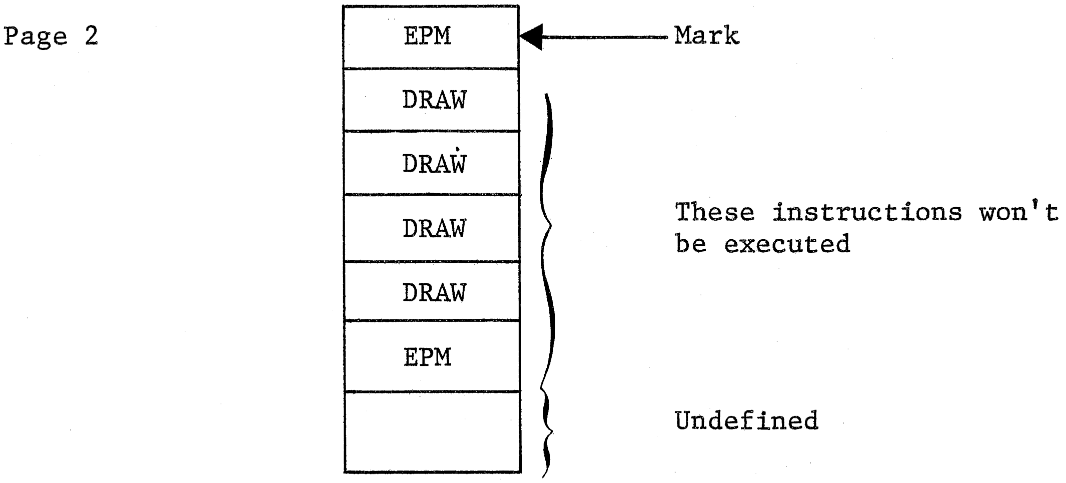
All operator inputs from the GRAPHIC 7 are returned via the EVENT subroutine. This subroutine is described in Section 13. To avoid confusion, let's say that the FSP program has been properly set up to detect function key responses.

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 7 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.)

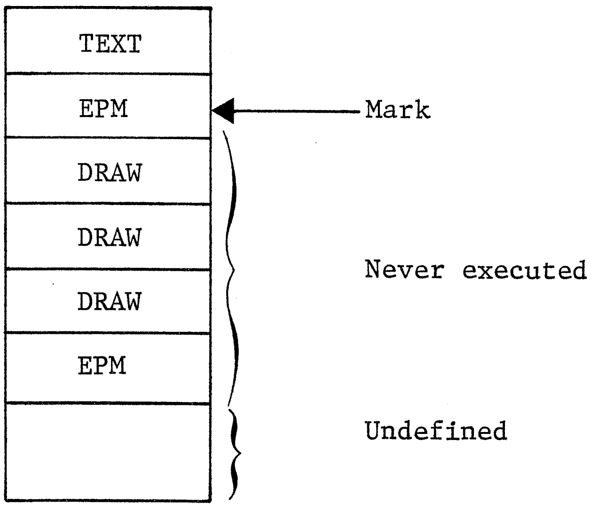
NOTE

When an ERASEP is executed, the page is also re-opened. 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 TEXT (   ),
```

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

Page 2



At this point the text contained in the TEXT array would be displayed.

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

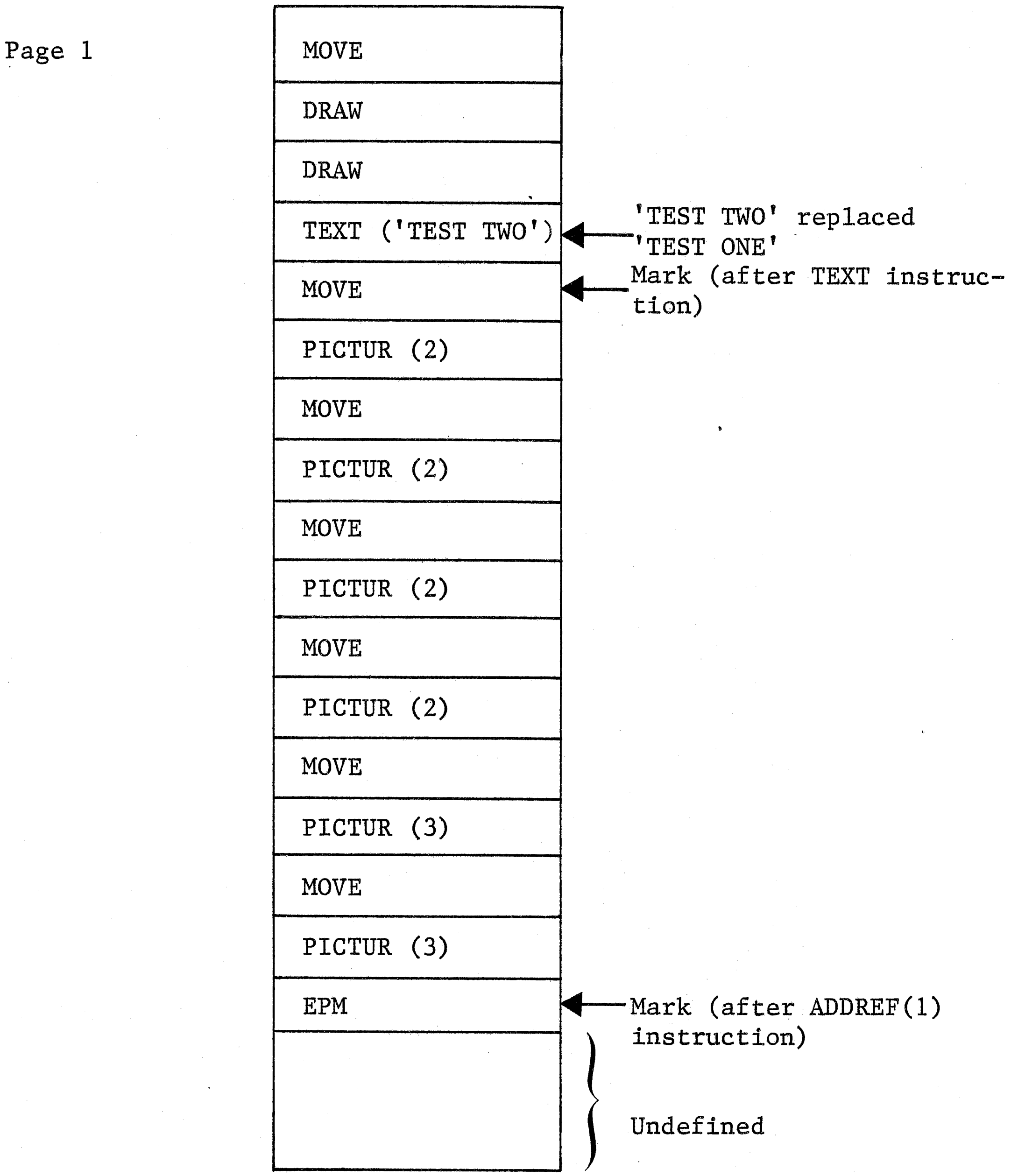
```
LINE NO.

  71           CALL REQMRK

  72     10    CALL EVENT (IEVNT)

  73           IF (IEVNT.NE.7) GOTO 10

  74           CALL GETMRK (MARK)
```

The above code would be inserted between lines 70 and 80. The call to REQMRK tells FSP to determine what the current mark value is. (Effectively, the mark points to the EPM which is where the TEXT instruction is stored.) When FSP determines the current mark value, it sets up the EVENT table to contain a mark event (i.e., event type 7). The event type 7 response indicates that the current mark value is now stored in the EVENT table. The CALL to GETMRK retrieves the current mark value from the event table. After the call to GETMRK, the variable MARK contains the current mark value. MARK is saved for future updating.

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 TEXT (       )

CALL ADDREF (1)
```

The call to UPDATE sets up the address pointer to point to the address where the TEXT instruction ('TEST ONE') is located. The call to TEXT ('TEST TWO') replaces the previous TEXT 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 TEXT instruction is added. After the TEXT instruction is executed, page 1 looks as follows:



Note that after the TEXT instruction is replaced in GRAPHIC 7 memory, the mark is pointing to the MOVE following the TEXT 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 TEXT instructions. In essence we would be destroying the refresh data in page 1.

When UPDATE is used, care must be used to ensure that 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 are intended for use by advanced FSP users. It is strongly recommended that new FSP users get some experience writing FSP programs before attempting to use the MOVEIM and COPYIM subroutines.

NOTE

Please read the subroutine descriptions for MOVEIM and COPYIM before continuing.

Normally a call MOVEIM is issued after the CALL COPYIM to remove the section of data that has been copied.

STEP A — Copy from MARKA to MARKB to end of page.

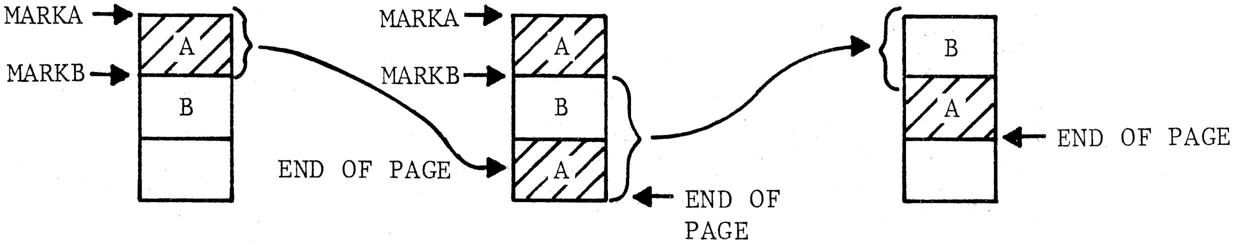
```
      CALL COPYIM (MARKA, MARKB)
```

STEP B — Remove copied refresh

```
      CALL MOVEIM (MARKB, MARKA)
```

STEP C — Get new page mark

```
   10          CALL EVENT (IEVENT)
               IF (IEVENT EQ. 7) GO TO 20
                .
                .
                .
               OTHER PROCESSING
                .
                .
               GO TO 10
C
C          GET NEW PAGE MARK
C
   20          CALL GETMRK (MARKC)
                .
                .
                .
                .
```



RESULTS OF STEP A

RESULTS OF STEP B

11.1 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).

11.2 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.

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

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 most likely cause an error.

11.3 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:

An "end of page" graphic order is placed at the present position of the "mark" causing all graphic orders following it to be removed from the graphic flow. If the program mode is in edit (UPDATE has been called), the program is taken out of the edit mode and into addition mode (equivalent to calling ADDREF).

11.4 GRAPHIC SUBROUTINE CALL

NAME: PICTUR

FUNCTION: Causes the contents of the specified page to be displayed at the current mark and beam position, i.e., a graphic subroutine order is inserted at the present page mark.

CALLING FORMAT: CALL PICTUR (IPAGE)

DESCRIPTION OF PARAMETERS:

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

-1 reserves space for a subsequent page call in the update mode.

DETAILED DESCRIPTION:

This routine causes the contents of page IPAGE to be called from the current mark and beam position. Note that the page calls should not be arranged so that a page can eventually call itself. See Section 6 for a more detailed description.

If IPAGE = -1, then three no operation instructions (NOP's) are inserted into the current page. The mark is advanced by 3.

11.5 REQUEST THE PRESENT PAGE MARK

NAME: REQMRK

FUNCTION: Allows caller to determine the location of the next available location on a page.

CALLING FORMAT: CALL REQMRK

DETAILED DESCRIPTION:

This routine causes the host program to request the current mark. The user then calls EVENT and GETMRK to get the value. The user may then use the value of the mark for subsequent updates.

11.6 GET MARK REQUEST INFORMATION

NAME: GETMRK

FUNCTION: Retrieves from the event tables the information requested by the REQMRK call.

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 value of the mark after a mark event has been received.

11.7 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 ≤ END OF PAGE

DETAILED DESCRIPTION:

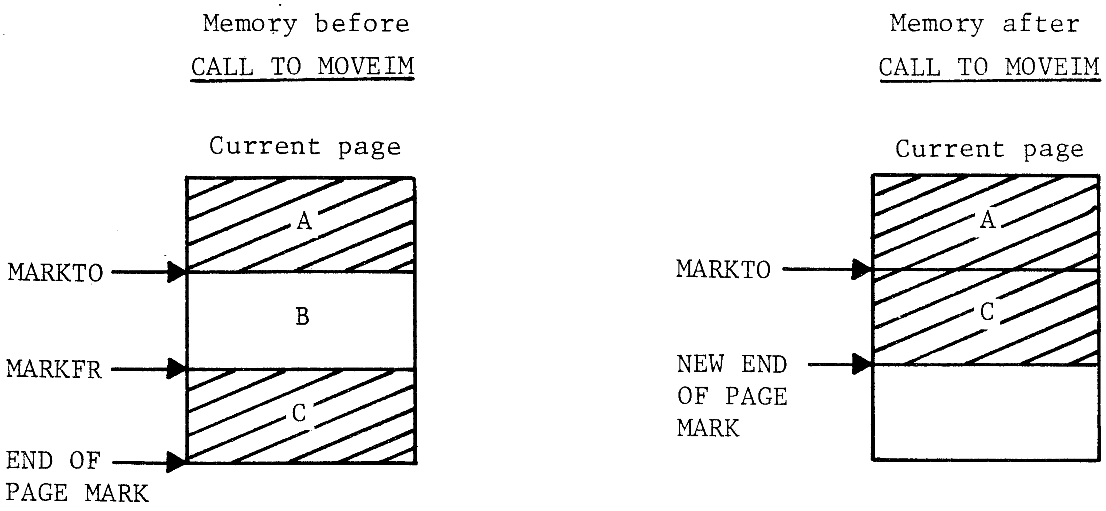
This routine moves data between a specified mark location (MARKFR) and the current end of page to a given mark location (MARKTO). The current end of page will be changed to equal MARKTO plus the length of the data move (old end of page minus MARKFR).

This routine automatically sends the current mark (new end of page) back to the host as if a CALL REQMRK had been issued by the host. The user calls EVENT and GETMRK to get the value. The user may use the value of the mark for subsequent updates.

The following error codes can be generated for MOVEIM:

ERROR CODE	MEANING
65	MARKTO> MARKFM
66	MARKFR> END OF PAGE

In terms of GRAPHIC 7 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.

11.8 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.

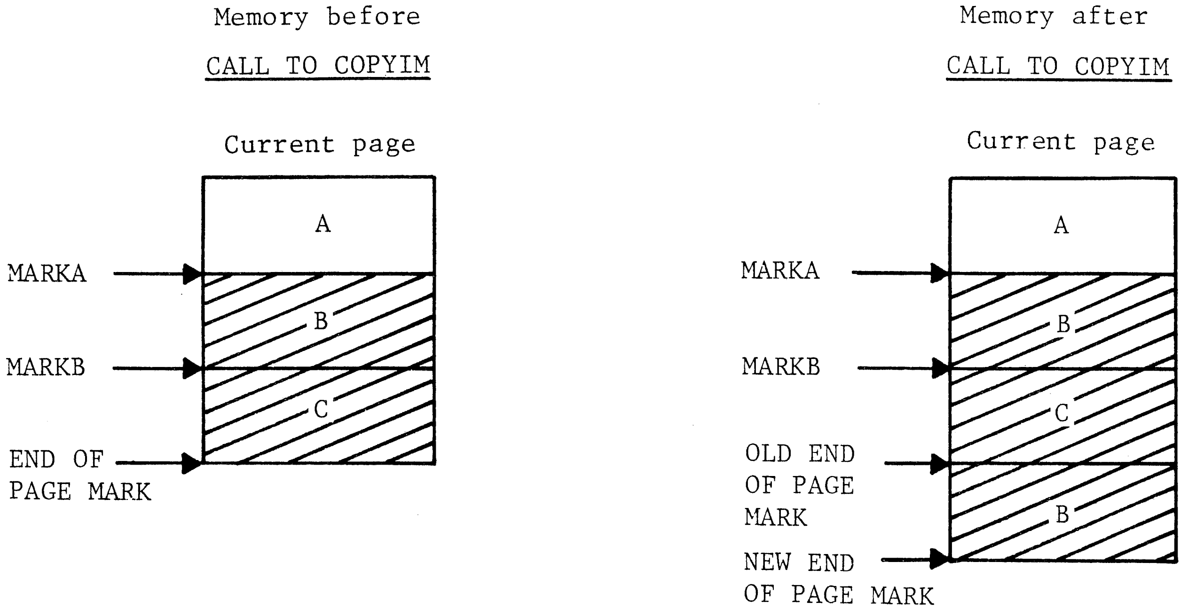
This routine allows the user to expand an existing image by copying or appending the image to the end of the current page where additional FSP functions may be performed.

This routine automatically sends the current mark (new end of page) back to the host as if a CALL REQMRK had been issued by the host. The user calls EVENT and GETMRK to get the new mark value.

The following error codes can be generated for COPYIM:

ERROR CODE	MEANING
62	MARKA > MARKB
63	MARKB > END OF PAGE
64	Not enough room on page for copy.

In terms of GRAPHIC 7 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 12

STATUS ROUTINES

The following routines described in this section allow the caller to define how the display data (graphic orders) will be seen, which PHOTOPENs are enabled, and which keyboard function keys are lit.

CPARM -- set character parameters for size, orientation, and spacing

DPARM -- set drawing and refresh rate and enable PHOTOPEN

STATUS -- set blinking, intensity, line style, and indicator selection(s)

LAMPON -- Turn keyboard function key on

LAMPOF -- Turn keyboard function key off

COLOR -- Select red, orange, yellow, or green

12.1 SET CHARACTER PARAMETERS

NAME: CPARM

FUNCTION: Allows the caller to select the character parameters: size, spacing, and orientation.

CALLING FORMAT: CALL CPARM (ICSIZE, ICROT, ICSPAC)

DESCRIPTION OF PARAMETERS:

ICSIZE = Integer variable selecting the character size desired.
Normal height 12" x 12" display
0 = size 0 (smallest) 0.125(inches)
1 = size 1 (1.5 times size 0) 0.187(inches) (default)
2 = size 2 (2.0 times size 0) 0.250(inches)
3 = size 3 (3.0 times size 0) 0.375(inches)

ICROT = Integer variable indicating the character orientation.
0 = normal (horizontal) (default)
1 = rotate 90° CCW

ICSPAC = Integer variable containing the spacing (in dits) between characters.

		Recommended Spacing Value
Size 0	--	10
Size 1	--	15 (default)
Size 2	--	20
Size 3	--	30

DETAILED DESCRIPTION

This routine generates a graphic order containing the caller specified character size, orientation, and spacing and places it at the mark position of the currently opened page. Since the GRAPHIC 7 uses a hardware character generator to display character, scaling has no impact on these character parameters.

12.2 SET DISPLAY PARAMETERS

NAME: DPARM

FUNCTION: Allows the caller to set the GRAPHIC 7 display parameters for drawing and refresh rate as well as enable and disable a selected PHOTOPEN.

CALLING FORMAT: CALL DPARM (ISP, ISYNC, IPEN)

DESCRIPTION OF PARAMETERS:

ISP = Integer variable selecting the drawing rate desired
0 = fast drawing rate (default)
1 = slow drawing rate

ISYNC = Integer variable selecting the refresh rate desired
0 = no change
1 = 60 Hz (normal) (default)
2 = 40 Hz
3 = 30 Hz

IPEN = Integer variable selecting the PHOTOPEN to be enabled
0 = Both PHOTOPENs disabled
1 = PHOTOPEN 1 enabled
2 = PHOTOPEN 2 enabled
3 = Both PHOTOPENs enabled (default)

DETAILED DESCRIPTION:

This routine generates a graphic order containing the caller specified drawing and refresh rates and enabled PHOTOPEN(s) and places it at the mark position of the currently opened page.

12.3 SET DISPLAY STATUS

NAME: STATUS

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

CALLING FORMAT: CALL STATUS (IBL, INT, IVT, IND)

DESCRIPTION OF PARAMETERS:

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

INT = Integer variable selecting 0-7 intensity levels
0 = invisible
1 = very dim
7 = (default) very bright

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

IND = Integer variable selecting which indicators are to be refreshed

0 - none
1 - #4
2 - #3
3 - #3 & 4
4 - #2
5 - #2 & 4
6 - #2 & 3
7 - #2, 3, & 4

8 - #1
9 - #1 & 4
10 - #1 & 3
11 - #1, 3, & 4
12 - #1 & 2
13 - #1, 2, & 4
14 - #1, 2, & 3
15 - #1, 2, 3, & 4 (default)
-1 = indicator not changed. Indicator specified in previous call to STATUS remains in effect.

DETAILED DESCRIPTION:

This routine generates a graphic order containing the caller specified display attributes for blinking, intensity, line style and indicator selection and places it in the currently opened page at the location pointed to by the page's mark pointer. The display will remain in the specified status until changed by another call to STATUS.

12.4 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 which of the two possible keyboards
1 = Keyboard 1
2 = Keyboard 2

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 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

12.5 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 which of the two possible keyboards
1 = Keyboard 1
2 = Keyboard 2

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

12.6 SET DISPLAY COLOR

NAME: COLOR

FUNCTION: Allows the caller to select one of four colors for the specified indicator(s).

CALLING FORMAT: CALL COLOR (ICOLOR, IND)

DESCRIPTION OF PARAMETERS:

ICOLOR = Integer variable containing the number of the color desired.
0 = red
1 = orange
2 = yellow
3 = green (default)

IND = Integer variable specifying the number of the desired indicator(s)
0 - none
1 - #4
2 - #3
3 - #3 & 4
4 - #2
5 - #2 & 4
6 - #2 & 3
7 - #2, 3, & 4
8 - #1
9 - #1 & 4
10 - #1 & 3
11 - #1, 3, & 4
12 - #1 & 2
13 - #1, 2, & 4
14 - #1, 2, & 3
15 - #1, 2, 3, & 4 (default)

DETAILED DESCRIPTION:

Routine creates a graphic order to change color and places it at the mark position of the currently open page. When possible, all codes of one color should be grouped together. A maximum of four color changes are allowed in an FSP program. If the color is changed more than four times, then the color change selection probably won't take place. The color CRT is protected against incorrect programming so the FSP programmer need not be overly concerned if an error is made and the FSP program contains 5 or 6 color changes.

SECTION 13

EVENT ROUTINE

An asynchronous (unpredictable) "event" may occur when one of the following operator actions takes place:

- A PHOTOPEN select is performed on a blank screen while in the "scan" mode.
- A PHOTOPEN select is performed on a graphic item while in the "item" mode.
- One of the 16 function keys on the alphanumeric/function keyboard is pressed.
- One of the 16 matrix keys on the alphanumeric/function keyboard is pressed.
- An alphanumeric key depression causes the "text input" buffer to become full.
- A CR (carriage return) key is pressed on the alphanumeric/function keyboard.
- The data tablet pen is pressed and released while in the "automatic" mode.

Another asynchronous event is the "error" event which is generated when any of the error conditions described in Appendix C are detected. These error conditions fall into the following categories:

- System errors

  A. Overload conditions

  B. Malfunctions
- Incorrect calling sequence
- User parameter errors
- Currently defined page refresh exceeded
- Miscellaneous

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 7 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
REQPXY - Request single PHOTOPEN scan
HCOPY - Request hardcopy
REQMRK - Request mark.

The FSP programmer becomes aware that an "event" has occurred only by calling the EVENT subroutine described in paragraph 13.1. 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.

13.1 POLL TERMINAL FOR EVENT OR REQUEST RESPONSE

NAME: EVENT

FUNCTION: Issues a poll request to GCP+ and waits for a poll response. 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 = There was a PHOTOPEN detect, call GETPEN.
6 = PHOTOPEN XY found, call GETPXY.
7 = Mark response available, call GETMRK.
8 = Error (XY overflow, halt, index out of range, etc.), call GETERR.
9 = Refresh dump available, call GETIM.
10 = Hardcopy complete.
11 = Unused.
12 = Illegal response from the terminal.

SECTION 14

ALPHANUMERIC 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

FSP allows and supports up to two alphanumeric/function keyboards. These keyboards combine a 32 function keyboard and an alphanumeric keyboard into one physical unit. A single keyboard system has the keyboard connected to port 3 of the first multiport serial interface. An additional keyboard may be added, and is connected to port 7 of the second multiport serial interface.

Multiport serial #1

PORT 1	
PORT 2	
PORT 3	- Alphanumeric/function keyboard #1
PORT 4	

Multiport serial #2

PORT 5	
PORT 6	
PORT 7	- Alphanumeric/function keyboard #2
PORT 8	

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

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.

14.1 ENABLE ALPHANUMERIC SCRATCH PAD

NAME: ENBPAD

FUNCTION: Specifies parameters for pad for entry of information from alphanumeric keyboard.

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

DESCRIPTION OF PARAMETERS:

IKEY = (1 or 2) keyboard number specified by the caller.

IND = (Ø thru 15) indicators on which the caller wishes alphanumeric information displayed. See STATUS routine for which values correspond to which indicators.

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 thru 86) user specified number of characters allowed in pad (default is 1).

DETAILED DESCRIPTION:

This routine allows the user to establish for each keyboard the location of the display of a single line of text entered from the alphanumeric keys. See GETTXT for further information. 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).

14.2 DISABLE ALPHANUMERIC SCRATCH PAD

NAME: DSAPAD

FUNCTION: Turns off the display of the alphanumeric keyboard information.

CALLING FORMAT: CALL DSAPAD (IKEY)

DESCRIPTION OF PARAMETERS:

IKEY = The keyboard number (1 or 2) 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.

14.3 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 8-bit ASCII character in the rightmost 8 bits of the element.

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 "NL" (new line) 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

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 two alphanumeric keyboards in the terminal is an 86-character input buffer (there are two 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 new line or the carriage return key is typed.

2. A call to GSS4 is made.

3. The buffer is full.

See ENBPAD for further information.

NOTE

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

14.4 GET FUNCTION KEY EVENT INFORMATION

NAME: GETKEY

FUNCTION: Retrieves from the event tables the function key event information.

CALLING FORMAT: CALL GETKEY (KBD, KEY)

DESCRIPTION OF PARAMETERS:

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

KBD = 1 = keyboard #1
KBD = 2 = keyboard #2

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

SECTION 15

PHOTOPEN ITEM ROUTINES

The PHOTOPEN can be used in two distinct ways:

- As a positional entry device
- As an item selection device

The two modes differ. Use as a positional entry device requires no graphics information on the screen; use as an item selection device requires existing graphics. Section 16 describes the positional entry mode. This section describes the item selection mode. The following subroutines are presented in this section:

ENBPEN	-	Enable global PHOTOPEN sensitization
DSAPEN	-	Disable global PHOTOPEN sensitization
ITEM	-	Define a graphic item
GETPEN	-	Get PHOTOPEN event information

A page or sections of a page (items) may either be PHOTOPEN responsive or non-PHOTOPEN responsive. A PHOTOPEN responsive item generates a PHOTOPEN event (event code 5) when the following operator actions are performed:

(1) The operator places the PHOTOPEN over any character, vector, conic, or point making up the item.

(2) The PHOTOPEN is pressed to activate its switch.

A PHOTOPEN event then makes available to the FSP programmer (via subroutine calls) the following information:

- PHOTOPEN number (1 or 2)
- Page and mark of exact graphic order within the item
- Type of graphic order (text, vector, code, point)
- Calling page if item is a refresh subroutine
- Text byte
- Item number

An item that is not PHOTOPEN responsive does not generate a PHOTOPEN event when the above operator actions are performed.

Two levels of sensitization are required to make an item PHOTOPEN responsive. Without these two levels enabled an item is not PHOTOPEN responsive.

Local Sensitization

This type of sensitization is provided by the DPARM subroutine and is used when the programmer wishes to sensitize an item locally. The same routine is used for desensitization, e.g.:

```
CALL DPARM
CALL MOVE  }
CALL DRAW               Sensitized (item 1)

CALL DPARM
CALL MOVE
CALL DRAW  }            Desensitized (item 2)
CALL DRAW

CALL DPARM
CALL POINT
     .     }            Sensitized (item 3)
     .
     .
```

A page built as a result of the above calls, when refreshed, is not yet PHOTOPEN responsive; i.e., an attempt to PHOTOPEN either item 1 or item 3 does not create a PHOTOPEN event. Global sensitization is required to make this possible.

Global Sensitization

The enabling and disabling of global sensitization is provided by the ENBPEN and DSAPEN routines. Global enabling (ENBPEN) allows all locally sensitized items in all pages to be PHOTOPEN responsive; i.e., a locally sensitized block when globally sensitized creates a PHOTOPEN event when penned.

Example:

```
                    CALL DSAPEN

                    CALL DPARM
                    CALL DRAW ]     Item 1 - Locally sensitized
Items 1,2,3         CALL DPARM
Not PHOTOPEN    {   CALL DRAW ]     Item 2 - Locally sensitized
responsive          CALL DPARM
                    CALL DRAW ]     Item 3 - Locally sensitized

                    CALL ENBPEN
Items 1,2,3         -
PHOTOPEN        {   -
responsive          -

                    CALL DSAPEN
Items 1,2,3         -
Not PHOTOPEN    {   -
responsive          -
```

The FSP programmer detects PHOTOPEN events by calling the EVENT routine and checking the returned event code for a value of 5. If a PHOTOPEN event is detected, then the FSP programmer must call the GETPEN subroutine, which returns to the caller all the PHOTOPEN event information.

Example:

```
CALL EVENT (IEVNT)

IF (IEVNT · EQ · 5) CALL GETPEN  (        )
```

15.1 ENABLE GLOBAL PHOTOPEN SENSITIZATION

NAME: ENBPEN

FUNCTION: Allows any locally sensitized pages or items to be PHOTOPEN responsive to the selected PHOTOPEN.

CALLING FORMATS: CALL ENBPEN (IPEN)

DESCRIPTION OF PARAMETERS:

$$\text{IPEN} = \begin{cases} 1 = \text{PHOTOPEN \#1 activated} \\ 2 = \text{PHOTOPEN \#2 activated} \end{cases}$$

DETAILED DESCRIPTION:

This call may be made before, during, or after the creation of a page or pages.

15.2 DISABLE GLOBAL PHOTOPEN SENSITIZATION

NAME: DSAPEN

FUNCTION: Causes all locally sensitized pages or items to be globally desensitized to the selected PHOTOPEN.

CALLING FORMAT: CALL DSAPEN (IPEN)

DESCRIPTION OF PARAMETERS:

$$\text{IPEN} = \begin{cases} 1 = \text{PHOTOPEN \#1 deactivated} \\ 2 = \text{PHOTOPEN \#2 deactivated} \end{cases}$$

DETAILED DESCRIPTION:

Performs opposite action of ENBPEN.

15.3 DEFINE A GRAPHIC ITEM

NAME: ITEM

FUNCTION: Allows the caller to associate an identifying number to all calls (MOVE, DRAW, etc.) which follow this call.

CALLING FORMAT: CALL ITEM (NUM)

DESCRIPTION OF PARAMETERS:

NUM = Integer number associated with the calls that follow.

DETAILED DESCRIPTION:

The caller can use this routine to give identifying numbers to items in the refresh file. This item number is returned to the caller via calls to GETPEN routine.

It is often desirable to group several vectors, points, characters, etc., into a logical entity or item and to make it PHOTOPEN responsive in such a way that penning any vector, point, character, etc. within the item causes a PHOTOPEN event, which in turn provides the item number.

Example:

```
CALL ITEM (25)

CALL DPARM

CALL DRAW ◀——ITEM 25

CALL ITEM (26)

CALL DRAW ◀——ITEM 26
```

15.4 GET PHOTOPEN EVENT INFORMATION

NAME: GETPEN

FUNCTION: Retrieves from the event tables all of the information concerning the PHOTOPEN event.

CALLING FORMAT: CALL GETPEN (IPEN,IPAGE,MARK,ITYPE,ICPAGE,IBYTE,ITMNUM)

DESCRIPTION OF PARAMETERS:

IPEN = An integer variable returned to the caller identifying which pen caused the event.

IPEN = 1 = PHOTOPEN #1
IPEN = 2 = PHOTOPEN #2

IPAGE = An integer variable returned to the caller identifying the page number in which the event occurred.

MARK = An integer variable returned to the caller giving the "mark" of the graphic order which caused the event.

ITYPE = An integer variable returned to the caller identifying the type of graphic order which caused the event.

1 = text
2 = vector
3 = circle
4 = point
5 = short vector

ICPAGE = An integer variable returned to the caller identifying the page number of the calling page.

IBYTE = An integer variable returned to the caller identifying the text byte causing the event.

ITMNUM = An integer variable returned to the caller giving the value of the item number at the event position.

DETAILED DESCRIPTION:

This routine retrieves the PHOTOPEN event information from the event tables. The page or item must have been locally sensitized by DPARM and globally sensitized by ENBPEN to allow this event to occur.

SECTION 16

PHOTOPEN SCAN ROUTINES

Section 15 described the PHOTOPEN routines available when the PHOTOPEN is used to pen existing graphic items on the screen. This section describes how the PHOTOPEN can be used to point to a blank area of the screen and obtain the X, Y position of that point. The following routines are described in this section:

ENBPXY	-	Enter multiple PHOTOPEN scan mode
DSAPXY	-	Leave multiple PHOTOPEN scan
REQPXY	-	Request single PHOTOPEN scan
GETPXY	-	Get PHOTOPEN scan event data

When PHOTOPEN scan is enabled (ENBPXY or REQPXY), PHOTOPEN events (event code 5) are not possible (see Section 15) but rather PHOTOPEN scan events are enabled. A PHOTOPEN scan event is created by the operator performing the following actions:

1. The operator points the PHOTOPEN to any position on the screen.

2. The operator presses the PHOTOPEN to produce a PHOTOPEN switch interrupt.

3. The operator observes a momentary flash as FSP displays a full screen of horizontal vectors to find the Y position and then a partial screen of vertical vectors to find the X position.

The FSP programmer detects PHOTOPEN scan events by calling the EVENT routine and checking the returned event code for a value of 6. If a PHOTOPEN scan event is detected, then the FSP programmer must call the GETPXY subroutine which returns to the caller the following PHOTOPEN scan information:

- Pen number

- X, Y position of pen

16.1 ENTER PHOTOPEN SCAN MODE

NAME: ENBPXY

FUNCTION: Sets the operating characteristics of the PHOTOPEN selected to the scan mode. In this mode, the PHOTOPEN acts as a positional entry device and can be pointed to a blank screen.

CALLING FORMAT: CALL ENBPXY (IPEN,ICRT)

DESCRIPTION OF PARAMETERS:

IPEN = $\begin{cases} 1 = \text{PHOTOPEN \#1 activated} \\ 2 = \text{PHOTOPEN \#2 activated} \end{cases}$

ICRT = An integer variable specifying which indicator to scan. See STATUS for the values of ICRT and the corresponding displays.

DETAILED DESCRIPTION:

This subroutine results in nothing visual but simply sets the PHOTOPEN to the scan mode. While in this mode, any number of scan events are possible. Both PHOTOPENs may be in the scan mode.

16.2 TERMINATE PHOTOPEN SCAN MODE

NAME: DSAPXY

FUNCTION: Sets the operating characteristics of the PHOTOPEN to the item mode.

CALLING FORMAT: CALL DSAPXY (IPEN)

DESCRIPTION OF PARAMETERS:

IPEN = $\begin{cases} 1 = \text{PHOTOPEN \#1 deactivated} \\ 2 = \text{PHOTOPEN \#2 deactivated} \end{cases}$

DETAILED DESCRIPTION:

This routine is normally visual in conjunction with ENBPXY.

16.3 REQUEST PHOTOPEN SCAN

NAME: REQPXY

FUNCTION: Leaves item mode and enters scan mode. Control is returned to the caller after a scan event has occurred. The item mode is then reestablished.

CALLING FORMAT: CALL REQPXY (IPEN,ICRT)

DESCRIPTION OF PARAMETERS:

IPEN = { 1 = PHOTOPEN #1 activated
2 = PHOTOPEN #2 activated

ICRT = An integer variable specifying which indicator to scan. See STATUS for the values of ICRT and the corresponding displays.

DETAILED DESCRIPTION:

This call is used to obtain a single scan event. The sequence of events performed by this call are as follows:

(1) Scan mode selected (CALL ENBPXY)

(2) Routine waits for operator to generate a scan event

(3) Item mode is reestablished (CALL DSAPXY)

To obtain the scan data, the FSP should call EVENT to verify an event code of 6 (PHOTOPEN scan) and then call GETPXY to obtain the X,Y position.

16.4 GET PHOTOPEN SCAN EVENT INFORMATION

NAME: GETPXY

FUNCTION: Retrieves from the event tables all information concerning the PHOTOPEN scan event.

CALLING FORMAT: CALL GETPXY (IPEN,X,Y)

DESCRIPTION OF PARAMETERS:

$$\text{IPEN} = \begin{cases} 1 = \text{PHOTOPEN \#1 caused scan event (returned)} \\ 2 = \text{PHOTOPEN \#2 caused scan event (returned)} \end{cases}$$

X,Y = Real variables in user coordinates indicating the X,Y position of the pen.

DETAILED DESCRIPTION:

Returns to the caller the PHOTOPEN scan information associated with the last PHOTOPEN scan event.

SECTION 17

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:

TBALL	-	Enable PED events
DISTB	-	Disable PED events
DTINIT	-	Assign data tablet as a PED
DTMODE	-	Select data tablet operating mode
REQTB	-	Request PED X, Y
GETTB	-	Get PED request information

A device which is used to input an X, Y coordinate to FSP is called a positional entry device (PED). FSP supports the following PEDs:

- Trackball
- Forcestick
- Data Tablet
- PHOTOPEN (scan mode)

NOTE

> The use of the PHOTOPEN as a PED is addressed in Section 16.

FSP allows and supports up to two PEDs. A single PED system has the PED connected to port 4 of the first multiport serial interface. An additional PED may be added, and is connected to port 8 of the second multiport serial interface.

Multiport serial #1

PORT 1	
PORT 2	
PORT 3	
PORT 4	- PED #1 (trackball or forcestick or data tablet)

Multiport serial #2

PORT 5	
PORT 6	
PORT 7	
PORT 8	- PED #2 (trackball or forcestick or data tablet)

FSP, when initialized (GSS4 call), assumes the PEDs to be either a trackball or a forcestick (default). A call to DTINIT and DTMODE is required if the PED or PEDs are data tablets. Following initialization, the X, Y position of the PED is set 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 or 2)
- 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 TBALL subroutine which locks 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 (TBALL)

(2) Change forcestick/trackball default condition if PED is a data tablet (DTINIT and DTMODE)

(3) Move PED to desired X, Y coordinate (operator action).

NOTE

Visual cursor follows PED movement.

(4) Request and get PED X, Y position (REQTB, EVENT, GETTB)

or, if PED is a data tablet in the automatic mode,

(4a) Get last PED position (EVENT, GETTB).

17.1 ENABLE PED EVENTS

NAME: TBALL

FUNCTION: Link PED movement to visual cursor and enable PED events.

CALLING FORMAT: CALL TBALL (ITBALL, IPAGE, MARK)

DESCRIPTION OF PARAMETERS:

ITBALL = Integer variable supplied by caller indicating the desired PED.

1 = PED #1
2 = PED #2

IPAGE = Page number containing user defined symbol. If Ø, a default symbol is used ("*" for PED #1 and "#" for PED #2)

MARK = The location of the absolute move preceding the symbol. If IPAGE = Ø, MARK is used to establish the indicators on which the default symbol is to be displayed (1 thru 15 - see STATUS for the values which correspond to the indicators).

DETAILED DESCRIPTION:

TBALL enables PED events and as such must be called before using the PED. In addition to enabling PED events, a default visual indicator is placed on the screen ("*" for PED #1 and "#" for PED #2). As the PED is moved (pen pressed on the data tablet), the cursor moves also to indicate the current X, Y position of the PED.

This routine also allows the user to link the PED to a MOVE absolute instruction in his page whose X, Y parameters will be modified in response to movement of the PED. The user should obtain the mark (REQMRK and GETMRK) of the MOVE absolute instruction before calling MOVE.

Following the absolute MOVE call, relative graphic instructions (TEXT or relative DRAWS) should be inserted. Now, when the absolute MOVE is updated, all of the relative vectors move also.

17.2 DISABLE PED EVENTS

NAME: DISTB

FUNCTION: Disconnects linkage between PED and visual cursor and disables PED events.

CALLING FORMAT: CALL DISTB (ITBALL)

DESCRIPTION OF PARAMETERS:

ITBALL = Integer variable supplied by caller indicating the desired PED.

1 = PED #1
2 = PED #2

DETAILED DESCRIPTION:

This routine reverses the action of a call to TBALL and no PED events are possible.

17.3 ASSIGN PED AS A DATA TABLET

NAME: DTINIT

FUNCTION: Informs FSP that a data tablet is present.

CALLING FORMAT: CALL DTINIT (IPEDNO)

DESCRIPTION OF PARAMETERS:

IPEDNO = Integer variable supplied by the caller indicating which PED position the data tablet is connected to.

1 = Connected to PED 1 position (port 4)
2 = Connected to PED 2 position (port 8)

DETAILED DESCRIPTION:

This routine must be called if a data tablet is present and should be called following the TBALL call. If not called, a trackball or forcestick is assumed, which will result in incorrect data when the data tablet is used.

NOTE

Data tablet messages are 10 characters in length. Trackball/forcestick messages are 2 characters in length. This call must be immediately followed by a "call" to DTMODE to select the "request" or "automatic" mode.

17.4 SELECT DATA TABLET OPERATING MODE

NAME: DTMODE

FUNCTION: Define the operating mode for the data tablet.

CALLING FORMAT: CALL DTMODE (IPEDNO, IMODE)

DESCRIPTION OF PARAMETERS:

IPEDNO = Integer variable supplied by the caller indicating which data tablet the mode change refers to.

1 = Data tablet #1 (port 4)
2 = Data tablet #2 (port 8)

IMODE = Integer variable supplied by the caller used to select the operating mode for the data tablet.

0 = Request mode
1 = Automatic mode

DETAILED DESCRIPTION:

This subroutine establishes the operating mode for a data tablet. There are two operating modes: the request mode and the automatic mode. In the request mode, data tablet X, Y position data messages are generated only in response to a REQTB call. In the automatic mode, data tablet X, Y position data messages are generated any time the user releases (for at least ¼ second) the switch on the data tablet pen (after it has been pressed). For both modes, the user calls EVENT to wait for the terminal response and then calls GETTB to obtain the values.

17.5 PED PROGRAMMING EXAMPLES

Example 1 -

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

                CALL GSS4
                CALL LAYOUT
                CALL TBALL (1,0,1)

C At this point in the program an "*" 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 #1 in automatic
C mode using default cursor on display #1

                CALL GSS4
                CALL LAYOUT
                CALL TBALL (2,0,2)

C At this point a "#" appears on display #1;
C however, the data tablet is not ready for use
C until the following two calls are made.

                CALL DTINIT (2)
                CALL DTMODE (2,1)

C At this point in the program the operator
C may move the "#" 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 #1 using
C a user defined cursor on display #1
C
C Create a single page 50 words in length

             CALL GSS4 (1,0,1)
             CALL LAYOUT (1,50)

C Define the cursor to be the letter "A"

             CALL MOVE (512.,512.,0)
             CALL TEXT (1,1HA)

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 TBALL (1,1,0)

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

17.6 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 = PED #1
2 = PED #1

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.

17.7 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 <u>returned</u> 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) GO TO 20
       CALL GETTB (NUMBER, X, Y)
       IF (NUMBER .NE. 2) GO TO 20
```

SECTION 18

MISCELLANEOUS ROUTINES

This section describes how to produce hard copies, how to request and subsequently how to receive a block of refresh data, and how to obtain error codes currently displayed on enabled display indicators.

The following routines are described in this section:

HCOPY - Initiate hard copy

REQIM - Request refresh image

GETIM - Receive refresh image

GETERR - Get error information

18.1 INITIATE HARD COPY

NAME: HCOPY

FUNCTION: Initiates a hard copy of all data currently being directed to the fourth display (output channel #4).

CALLING FORMAT: CALL HCOPY (-1)

DESCRIPTION OF PARAMETERS:

-1 = Start hard copy indicator

DETAILED DESCRIPTION:

The STATUS call is used for selecting display #4 as the output channel. Approximately 7 seconds after this call is made, a hard copy event (event code 10) is generated to indicate the hard copy is complete and that a new copy may be initiated.

Example:

```
C   Initiate hard copy and wait for
C   completion
       CALL HCOPY (-1)
10     CALL EVENT (IEVNT)
       IF (IEVNT.NE.10) GO TO 10
```

A direct approach for obtaining a hard copy of an image is to select two indicators in the call to STATUS. The first indicator is the display on which the graphics is to appear and the second indicator is display #4 which is used for hard copy.

Example:

```
C   Draw a line on display #1 and then
C   get a hard copy
       CALL GSS4 (1, 0, 1)
       CALL LAYOUT (1, 50)
       CALL STATUS (0, 7, 0, 9)
```

NOTE

The 9 in the above STATUS call selected indicators 1 and 4.

```
       CALL MOVE (512., 512., 0)
       CALL DRAW (1023., 1023., 0)
```

```
C    Now initiate hard copy
        CALL HCOPY (-1)
```

NOTE

If the STATUS call in the above example had selected indicator 1 only, then a blank piece of paper would have been produced. The following edit code would be required to obtain the copy:

```
        CALL UPDATE (1, 0)
```

NOTE

The mark associated with the STATUS call, since it is the first item in the page, is = 0

```
        CALL STATUS (0, 7, 0, 9)
        CALL HCOPY (-1)
```

18.2 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.

18.3 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)
```

18.4 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 19

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 insure 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
```

19.1 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.

19.2 PACKED VECTOR MOVE

NAME: PMOVE

FUNCTION: Allows the caller to move the CRT beam 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 to which the beam 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 10.1). The beam is moved to the X, Y coordinate specified by the caller. The X, Y coordinate then becomes the current beam 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).

The sample user program segment at the introduction to this section illustrates a use of PMOVE.

19.3 PACKED VECTOR DRAW

NAME: PDRAW

FUNCTION: Allows the user to draw a line (vector) from the current CRT beam 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 10.2). A line is drawn from the current beam position to the X, Y coordinate specified by the caller. The X, Y coordinate then becomes the current beam 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).

The sample user program segment at the introduction to this section illustrates a use of PDRAW.

19.4 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 insures that residual packed vector data will be sent (see introduction to this section).

SECTION 20

COORDINATE CONVERTER ROUTINES

Four subroutines enable the programmer to manipulate the coordinate converter. They are:

CCINIT	- Initialize coordinate converter
CCVAL	- Activate the coordinate converter with specific values
CCON	- Turn on the coordinate converter
CCOFF	- Turn off the coordinate converter

These subroutines enable the user to selectively rotate and translate a particular segment of the image, multiple segments of the image, or the total image displayed on the screen. For example, the picture displayed may have one part rotated 90 degrees, another 310 degrees, another 45 degrees, and another not rotated at all. The rotation can be performed around any point of the screen. This is made possible by positioning the point upon which an image is to be rotated at the center of the screen, rotating the image to the desired angle (0.2 degree resolution), and then translating the image to the position desired. For example, say the picture the programmer wants to display is as shown in figure 20-1.

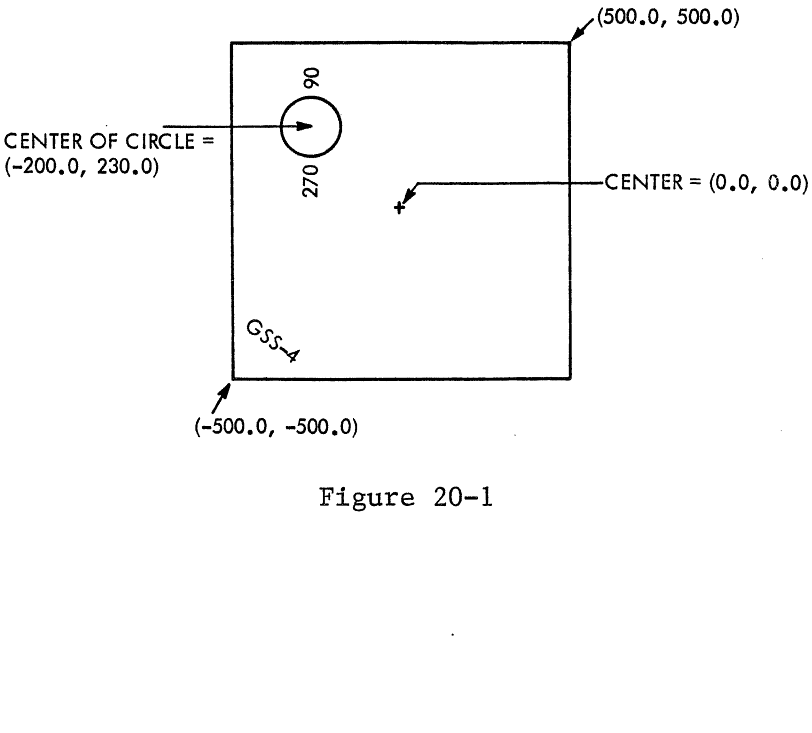


Figure 20-1

The picture contains three objects drawn at different positions and angles:

Object 1: +

Object 2: 270 ○ 90 (drawn at 90°)

Object 3: GSS-4 (drawn at 310°)

Object 1 is a small cross, not rotated, drawn at the center of the screen. Object 2 requires the coordinate converter. An example of the code necessary to rotate and position object 2 is as follows:

```
CALL CCVAL   (0, 90.0, -200., 230.,0)
CALL MOVE    (0., 0., 0)
CALL CIRCLE  (50.0, 0)
CALL MOVE    (-150., 0.0, 0)
CALL TEXT    (3, IT270) (IT270 = array of text '270')
CALL MOVE    (100.0, 0.0, 0)
CALL TEXT    (2, IT90) (IT90 = array of text '90')
CALL CCOFF   (0)
```

Pictorially, the events that take place are as follows:

1. First, the image is drawn around the origin as specified by calls to MOVE, CIRCLE and TEXT. See figure 20-2.

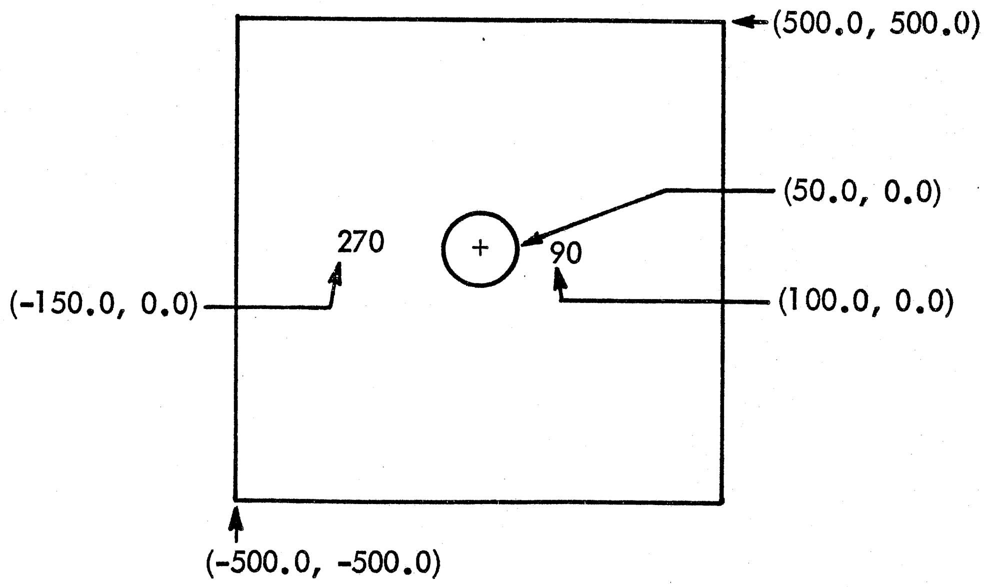


Figure 20-2

2. The coordinate converter rotates the image 90 degrees as specified by an argument to subroutine CCVAL. See figure 20-3.

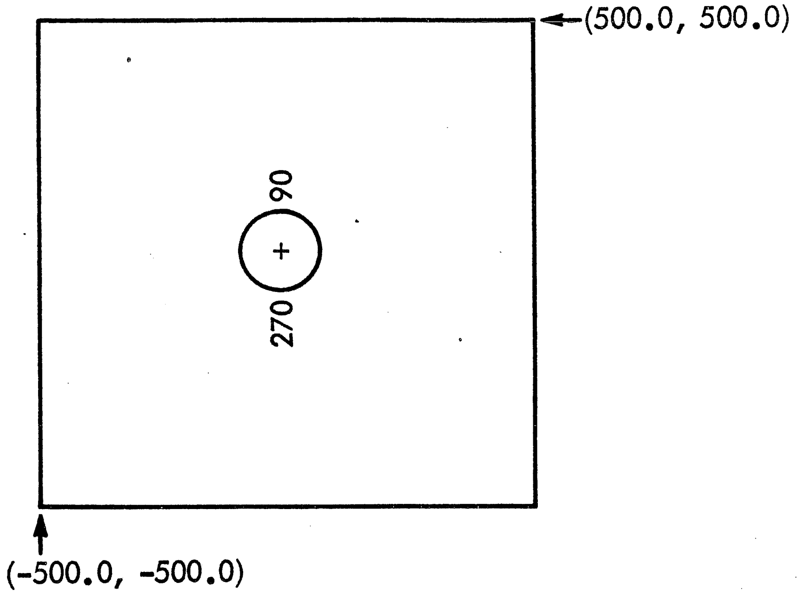


Figure 20-3

3. The coordinate converter then translates object 2 to the position specified by the arguments to CCVAL. See figure 20-4.

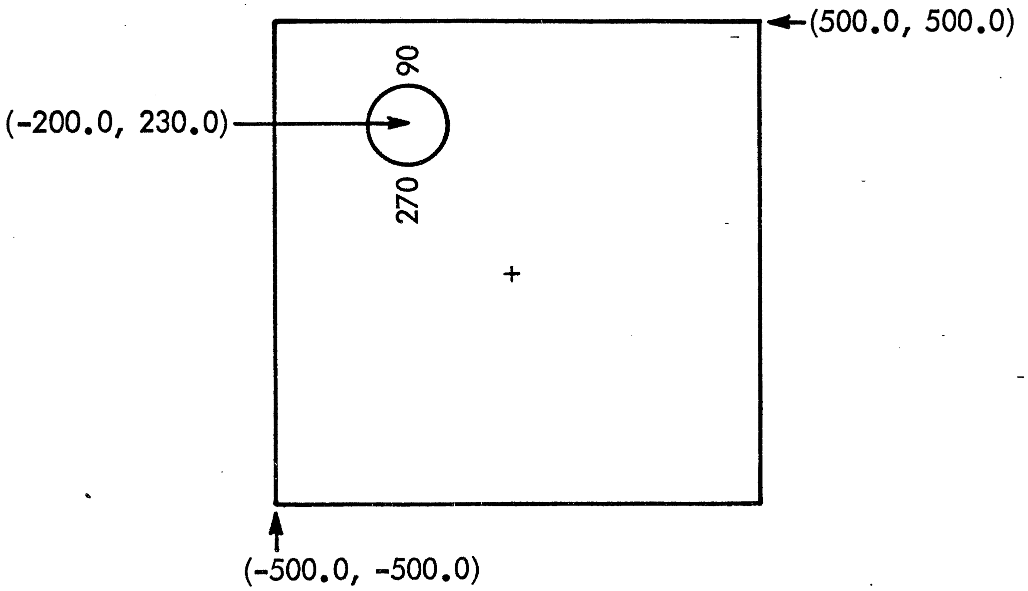


Figure 20-4

Case 1 - box translated
to (100.0, 100.0)

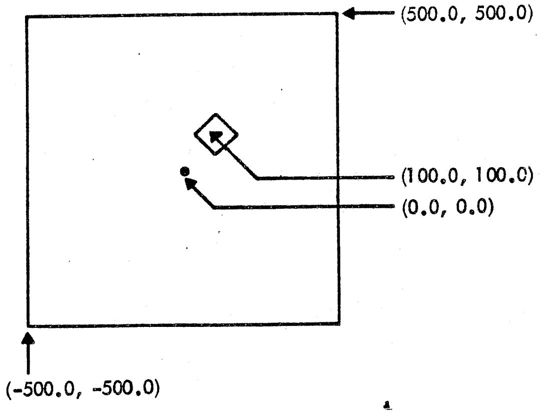


Figure 20-10

Case 2 - box translated
to (100.0, 100.0)

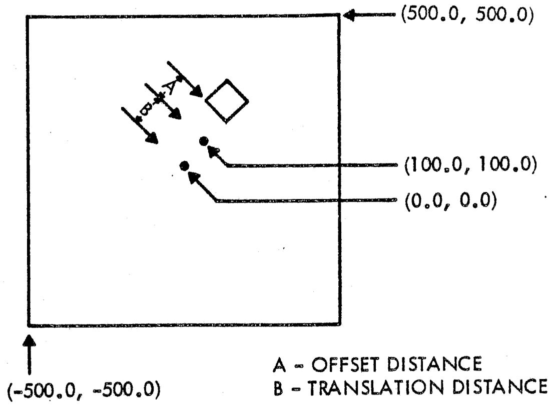


Figure 20-11

In case 1, the box ends up rotated about the point (100.0, 100.0). In case 2, the box ends up rotated and translated with respect to the center of the screen.

Four display registers are associated with the coordinate converter. Two control the angle of rotation: cosine and sine register; and two control the translation positions: X translation coordinate and Y translation coordinate. Four refresh commands load these registers and another command controls activating or deactivating the coordinate converter. Each command advances the mark by 2.
A detailed description of the four FSP subroutines that manipulate the coordinate converter are presented on the following pages.

20.1 INITIALIZE COORDINATE CONVERTER

NAME: CCINIT

FUNCTION: Initialize the coordinate converter to a standard initial state.

CALLING FORMAT: CALL CCINIT (XCEN, YCEN)

DESCRIPTION OF PARAMETERS:

XCEN = A floating point variable returned to the caller specifying the X-coordinate of the center of the display in user coordinates.

YCEN = A floating point variable returned to the caller specifying the Y-coordinate of the center of the display in user coordinates.

DETAILED DESCRIPTION:

This subroutine initializes the coordinate converter to the following initial state:

X translation = 0

Y translation = 0

Angle = 0 degrees

The returned parameters specify the location upon which all rotation occurs before translation. This subroutine advances the mark by 10.

20.2 ACTIVATE THE COORDINATE CONVERTER WITH SPECIFIC VALUES

NAME: CCVAL

FUNCTION: Inserts values into the coordinate converter registers and activates the coordinate converter.

CALLING FORMAT: CALL CCVAL (K, ANGLE, XTRAN, YTRAN, IREL)

DESCRIPTION OF PARAMETERS:

K = An integer constant defining angle format.

0.. angle in degrees
1.. angle in radians

ANGLE = Angle of rotation in degrees or radians.

XTRAN = X position of translation.

YTRAN = Y position of translation.

IREL = Specifies (XTRAN, YTRAN) format

0.. absolute position
1.. relative position from center

DETAILED DESCRIPTION:

All refresh data following this call, until another CCVAL or CCOFF call, is rotated and translated by the coordinate converter to the values the caller specified. This subroutine advances the mark by 10.

20.3 TURN ON THE COORDINATE CONVERTER

NAME: CCON

FUNCTION: Activates the coordinate converter.

CALLING FORMAT: CALL CCON

DETAILED DESCRIPTION:

This subroutine activates the coordinate converter. The values the coordinate converter responds to would have been previously set by calls to CCVAL or CCINIT. All refresh data following this call, until another CCVAL or CCOFF call, is rotated and translated by the coordinate converter to the values previously specified. To deactivate the coordinate converter, or change its values, the user would call CCOFF or CCVAL, respectively. This subroutine advances the mark by 2.

20.4 TURN OFF THE COORDINATE CONVERTER

NAME: CCOFF

FUNCTION: Deactivates the coordinate converter.

CALLING FORMAT: CALL CCOFF

DETAILED DESCRIPTION:

This subroutine deactivates (turns off) the coordinate converter. All following calls to FSP are not affected by the coordinate converter. The values set in the coordinate converter registers remain intact and may be reactivated or modified by calls to CCON or CCVAL, respectively. The mark is advanced by 2.

SECTION 21

IMAGE CONTROL ROUTINES

There are two image control routines available to the FSP programmer: CLIP, which is used to eliminate graphic data that lies outside a user specified window, and SMOOTH, which is used to smooth lines by eliminating changes in direction too small to be visible. These routines require no special hardware, but rather all of the operations are done in software on the host computer. Using these routines, the user is able to preprocess his graphic data before shipping it to the GRAPHIC 7 via the FSP subroutine calls.

CLIP

It is often desirable to define a box in the viewing area in which an image is to be confined. This requires that the user guarantee that all graphic instructions which move the beam (MOVE,DRAW) do not cause data to be displayed outside of the selected window. By using the Sanders-supplied CLIP subroutine, the FSP programmer is relieved of the burden of performing his own slope calculations to guarantee that all data intersecting with the window boundaries do not cross outside. Refer to paragraph 21.1 for a description of the CLIP routine.

SMOOTH

The smooth routine is used to remove 'kinks' in a line composed of a series of connected lines whose changes in direction with respect to one another are too small to be seen. In many cases, this routine reduces the amount of data required to 'refresh' the image (by collecting several 'draws' into one 'draw') and causes a clearer looking plot by smoothing out the line. See paragraph 21.2 for a description of the SMOOTH routine.

NOTE

> Please refer to examples 4 and 5 in Appendix E for sample FSP programs demonstrating the use of the CLIP and SMOOTH subroutines.

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

21.2 SMOOTH DISPLAYED LINES

NAME: SMOOTH

FUNCTION: This subroutine straightens lines whose kinks are too small to be noticeable on the screen, 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 the beam to X, Y.

3. IOP = 6: Call DRAW to draw a line from the present beam 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 Ø.ØØØ9765625 (1/1Ø24) of full screen width for the GRAPHIC 7. 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 Ø.Ø, SMOOTH produces output identical to its input with the exception of eliminating adjacent coincidental points.

SECTION 22

FSP INPUT/OUTPUT

Four subroutines are used by FSP for performing I/O to the GRAPHIC 7 terminal. They are:

G7INIT	-	Initialize host/GRAPHIC 7 I/O driver
G7TERM	-	Terminate host/GRAPHIC 7 I/O driver
MSGOUT	-	Output message to GRAPHIC 7 terminal
MSGIN	-	Input message from GRAPHIC 7 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, page 1-3) unless special arrangements have been made.

FSP OUTPUT

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

a. Create a message (header plus data) and place it in an output block.

b. Call MSGOUT to transmit the message to the GRAPHIC 7 terminal for execution.

FSP INPUT

The graphic control program enhanced (GCP+) in the GRAPHIC 7 sends data to the host when polled by the host, i.e., PHOTOPEN, 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 7

CALL MSGIN - Read POLL response from GRAPHIC 7

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

22.1 INITIALIZE HOST/GRAPHIC 7 I/O DRIVER

NAME: G7INIT

FUNCTION: To initialize the host/GRAPHIC 7 I/O driver.

CALLING FORMAT: CALL G7INIT (IUNIT)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 7 in the call to the GSS4 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 GSS4 subroutine.

For parallel hosts, when the application program makes a call to the FSP subroutine, several functions are performed to initialize the GRAPHIC 7 terminal to FSP mode. One function is to logically connect the application program with the GRAPHIC 7 within the architecture of the host operating system. The actual operations needed to perform the connection are host-dependent. Internal to the GSS4 subroutine a call is made to the G7INIT subroutine to provide the customer with a mechanism for performing the I/O driver initialization process. In terms of the GRAPHIC 7, the key function performed by G7INIT is to initialize (under program control) the terminal controller by pulsing the INIT control line to the GRAPHIC 7 parallel interface card. The INIT pulse resets the terminal controller to the system mode.

NOTE

The INIT control line is exposed at the host end.

For serial hosts, the G7INIT subroutine is not needed. To eliminate compilation errors, the customer should write a dummy G7INIT subroutine that only has a return statement.

For example:
```
SUBROUTINE G7INIT (IUNIT)
RETURN
END
```

For parallel hosts, the design of the G7INIT subroutine is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Host operating system
- GRAPHIC 7 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.

22.2 TERMINATE HOST/GRAPHIC 7 I/O DRIVER

NAME: G7TERM

FUNCTION: To terminate the host/GRAPHIC 7 I/O driver

CALLING FORMAT: CALL G7TERM (IUNIT)

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 7 in the call to the GSS4 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 GSS4 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 7 terminal to the TTY emulator mode. One function is to logically terminate the connection between the application program and the GRAPHIC 7. The actual operations needed to terminate the connection are host-dependent. Internal to the THEEND subroutine a call is made to the G7TERM subroutine to provide the customer with a mechanism for performing the I/O driver termination process.

For serial hosts, the G7TERM subroutine is not needed. To eliminate compilation errors, the customer should write a dummy G7TERM subroutine that only has a return statement.

For parallel hosts, the design of the G7TERM subroutine is influenced by the following factors:

- Host word length (16, 24, 32, 36, etc.)
- Host I/O system
- Host operating system
- GRAPHIC 7 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.

22.3 OUTPUT MESSAGE TO GRAPHIC 7 TERMINAL

NAME: MSGOUT

FUNCTION: Outputs a message to the GRAPHIC 7 terminal.

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

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 7 in the call to the GSS4 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 be 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 7 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 7 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.

22.4 INPUT MESSAGE FROM GRAPHIC 7 TERMINAL

NAME: MSGIN

FUNCTION: Inputs a message from the GRAPHIC 7 terminal.

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

DESCRIPTION OF PARAMETERS:

IUNIT = An integer variable containing the device number associated with the GRAPHIC 7 in the call to the GSS4 subroutine.

IBUF = An integer array into which data received from the GRAPHIC 7 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 7 I/O driver (provided by customer)

NOTE

> In the serial mode, the GRAPHIC 7 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 23

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 User's 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).

50, 2

SECTION 24

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 7 I/O driver must be written and made part of the operating system.

C. The MSGOUT, MSGIN, G7INIT, and G7TERM 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 GSS4 (IUNIT,Ø,IFACE)
```

IUNIT must be set to the logical unit number assigned to the GRAPHIC 7 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 7 is also configured to ignore command header errors. The polling mode configuration (set up by Sanders) works in the following manner:

   1. FSP user calls EVENT

   2. EVENT sends a POLL message to the GRAPHIC 7 via MSGOUT.

   3. The GRAPHIC 7 receives the POLL message and does the following:

      A. Sends out the next message in the O/P buffer.

      B. If the O/P buffer is empty, the GRAPHIC 7 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 7 returns a message.)

To minimize the number of host to GRAPHIC 7 messages, the CKPOLL subroutine can be modified so that the GRAPHIC 7 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 7 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 or problems can be encountered. (I.e., when the host computer receives a message from the GRAPHIC 7, it echos it back to the GRAPHIC 7, 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 7 occurs.

The CKPOLL subroutine can also be configured to operate in a special type of polling mode. In this mode, the sending of GRAPHIC 7 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 7 via MSGOUT.

3. EVENT sends a special character to the GRAPHIC 7 to indicate that the host is set up to read in the next message from the GRAPHIC 7.

4. When the GRAPHIC 7 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 7.

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 7 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 7 GCP+ Programmers Reference Manual.
>
> Normally when the THEEND subroutine is called, the GRAPHIC 7 is returned to the full-duplex teletype emulator. If half-duplex is being used, the THEEND subroutine can be modified to return the user to the half-duplex teletype emulator as follows:
>
> Change IOUTB(2)=30884 to IOUTB(2)=30880

SECTION 25

STARTUP PROCEDURE

The following paragraphs assume the following:

- All steps of the installation procedure have been performed.
- The GRAPHIC 7 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.

A. Press LOCAL button on front panel of the GRAPHIC 7 and observe built-in test pattern. Validate (using this pattern and its associated built-in diagnostics) that the terminal is in working order.

B. Press the "RETURN" key on the keyboard (causes pattern to disappear and "BØØM" to appear).

C. Press "Y" key followed by "RETURN" key to enter teletype emulation mode. At this point the GRAPHIC 7 performs as a teletype 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 teletype emulator is re-entered.

NOTE

Refer to the GRAPHIC 7 GCP+ Programmer's Reference Manual (H-79-0348) 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
11-15	ADDREF (IPAGE)	Open page for adding refresh data
20-7	CCINIT (XCEN, YCEN)	Initialize coordinate converter
20-8	CCVAL (K, ANGLE, XTRAN, YTRAN, IREL)	Activate the coordinate converter with specific values
20-9	CCON	Turn on the coordinate converter
20-9	CCOFF	Turn off the coordinate converter
10-8	CIRCLE (RADIUS, IQUAD)	Draw a circle
21-2	CLIP (IOP, X1, Y1, X2, Y2, XLC, YLC, XUC, YUC, XPOSB, YPOSB)	Remove off-screen data
12-7	COLOR (ICOLOR, IND)	Set display color
11-20	COPYIM (MARKA, MARKB)	Copy a block of graphic orders
12-2	CPARM (ICSIZE, ICROT, ICSPAC)	Set character parameters
17-5	DISTB (ITBALL)	Disconnect PED from symbol
12-3	DPARM (ISP, ISYNC, IPEN)	Set display parameters
10-4	DRAW (X, Y, MODE)	Draw a vector
9-7	DSABOX (IND)	Turn border display off
9-8	DSAERR (IND)	Turn error display off
14-3	DSAPAD (IKEY)	Disable alphanumeric scratch pad
15-4	DSAPEN (IPEN)	Disable PHOTOPEN detection mechanism
19-5	DSAPMD	Disable packed vector mode

FSP SUBROUTINES (Cont.)

PAGE	SUBROUTINE	DESCRIPTION
16-2	DSAPXY (IPEN)	Disable PHOTOPEN scan
17-5	DTINIT (IPEDNO)	Assign PED as a data tablet
17-6	DTMODE (IPEDNO, IMODE)	Select data tablet operating mode
9-7	ENBBOX (IND)	Turn border display on
9-8	ENBERR (IND)	Turn error display on
14-2	ENBPAD (IKEY, IND, X, Y, IMAX)	Enable alphanumeric scratch pad
15-4	ENBPEN (IPEN)	Enable PHOTOPEN detection mechanism
19-2	ENBPMD	Enable packed vector
16-2	ENBPXY (IPEN, ICRT)	Enable PHOTOPEN scan
11-16	ERASEP	Erase from page mark to end of page
13-2	EVENT (IEVNT)	Poll terminal for event or request
22-2	G7INIT (IUNIT)	Initialize host/GRAPHIC 7 I/O driver
22-3	G7TERM (IUNIT)	Terminate host/GRAPHIC 7 I/O driver
18-5	GETERR (IARRAY)	Get error information
18-4	GETIM (IARRAY, ISIZE, NINST, IPAGE)	Get refresh image
14-4	GETKEY (KBD, KEY)	Get function key event information
11-17	GETMRK (M)	Get mark request information
15-6	GETPEN (IPEN, IPAGE, MARK, ITYPE, ICPAGE, IBYTE, ITMNUM)	Get PHOTOPEN event information
16-4	GETPXY (IPEN, X, Y)	Get PHOTOPEN X, Y request information
17-10	GETTB (NUMBER, X, Y)	Get PED request information
14-3	GETTXT (IARRAY, ISIZE, NCHAR, KBD)	Get text event information
9-2	GSS4 (IUNIT, IDUM, IFACE)	Initialize terminal to FSP mode
18-2	HCOPY (-1)	Initiate a hard copy

FSP SUBROUTINES (Cont.)

PAGE	SUBROUTINE	DESCRIPTION
15-5	ITEM (NUM)	Set item number
12-6	LAMPOF (KBD, LAMP)	Turn keyboard lamp off
12-5	LAMPON (KBD, LAMP)	Turn keyboard lamp on
9-3	LAYOUT (NPAGES, LNGARY)	Define graphic page layout
10-2	MOVE (X, Y MODE)	Move beam to the position specified
11-18	MOVEIM (MARKFR, MARKTO)	Move a block of graphic orders
22-5	MSGIN (IUNIT, IBUF, IELEMC)	Input message from GRAPHIC 7 Terminal
22-4	MSGOUT (IUNIT, IBUF, IELEMC)	Output message to GRAPHIC 7 terminal
19-4	PDRAW (X, Y)	Packed vector draw
11-16	PICTUR (IPAGE)	Graphic subroutine call
19-3	PMOVE (X, Y)	Packed vector move
10-7	POINT	Display a point
10-9	REFDAT (IARRAY, N)	Transfer a block of graphic orders
18-3	REQIM (NINST)	Request refresh image
11-17	REQMRK	Request the present page mark
16-3	REQPXY (IPEN, ICRT)	Request PHOTOPEN X, Y
17-9	REQTB (NUMBER)	Request PED X, Y
9-6	SCALE (XL, YL, XL, YU)	Define coordinates
21-4	SMOOTH (IOP, X, Y, ISAVE, XSAVE, YSAVE, MSAVE, EPS)	Smooth displayed lines
12-4	STATUS (IBL, INT, IVT, IND)	Set display status
17-4	TBALL (ITBALL, IPAGE, MARK)	Connect PED to symbol
10-6	TEXT (N, IARRAY)	Display text characters
9-9	THEEND	Terminate FSP mode
11-15	UPDATE (IPAGE, MARK)	Open page for editing refresh data

APPENDIX B

ASCII CODES

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
ØØØ	ØØ	NUL	@	NULL, CTRL SHIFT P, TAPE LEADER
ØØ1	Ø1	SOH	A	START OF HEADER, SOM
ØØ2	Ø2	STX	B	START OF TEXT, EOA
ØØ3	Ø3	ETX	C	END OF TEXT, EOM
ØØ4	Ø4	EOT	D	END OF TRANSMISSION, END
ØØ5	Ø5	ENQ	E	ENQUIRY, WRU, WHO ARE YOU
ØØ6	Ø6	ACK	F	ACKNOWLEDGE, RU, ARE YOU
ØØ7	Ø7	BEL	G	BELL
Ø1Ø	Ø8	BS	H	BACKSPACE, FEØ
Ø11	Ø9	HT	I	HORIZONTAL TAB, TAB
Ø12	ØA	LF	J	LINE FEED, NEW LINE, NL
Ø13	ØB	VT	K	VERTICAL TAB, VTAB
Ø14	ØC	FF	L	FORM FEED, FORM, PAGE
Ø15	ØD	CR	M	CARRIAGE RETURN, EOL
Ø16	ØE	SO	N	SHIFT OUT, RED SHIFT
Ø17	ØF	SI	O	SHIFT IN, BLACK SHIFT
Ø2Ø	1Ø	DLE	P	DATA LINK ESCAPE, DCØ
Ø21	11	DC1	Q	XON, READER ON
Ø22	12	DC2	R	TAPE, PUNCH ON
Ø23	13	DC3	S	XOFF, READER OFF
Ø24	14	DC4	T	TAPE, PUNCH OFF
Ø25	15	NAK	U	NEGATIVE ACKNOWLEDGE, ERR
Ø26	16	SYN	V	SYNCHRONOUS IDLE, SYNC
Ø27	17	ETB	W	END OF TEXT BUFFER, LEM
Ø3Ø	18	CAN	X	CANCEL, CANCL
Ø31	19	EM	Y	END OF MEDIUM
Ø32	1A	SUB	Z	SUBSTITUTE
Ø33	1B	ESC		ESCAPE, PREFIX
Ø34	1C	FS		FILE SEPARATOR

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
Ø35	1D	GS		GROUP SEPARATOR
Ø36	1E	RS		RECORD SEPARATOR
Ø37	1F	US		UNIT SEPARATOR
Ø4Ø	2Ø	SP		SPACE, BLANK
Ø41	21	!		
Ø42	22	"		
Ø43	23	#		
Ø44	24	$		
Ø45	25	%		
Ø46	26	&		
Ø47	27	'		APOSTROPHE
Ø5Ø	28	(		
Ø51	29	)		
Ø52	2A	*		
Ø53	2B	+		
Ø54	2C	,		COMMA
Ø55	2D	-		MINUS
Ø56	2E	.		
Ø57	2F	/		
Ø6Ø	3Ø	Ø		NUMBER ZERO
Ø61	31	1		NUMBER ONE
Ø62	32	2		
Ø63	33	3		
Ø64	34	4		
Ø65	35	5		
Ø66	36	6		
Ø67	37	7		
Ø7Ø	38	8		
Ø71	39	9		
Ø72	3A	:		
Ø73	3B	;		
Ø74	3C	<		LESS THAN

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
Ø75	3D	=		
Ø76	3E	>		GREATER THAN
Ø77	3F	?		
1ØØ	4Ø	@		SHIFT P
1Ø1	41	A		
1Ø2	42	B		
1Ø3	43	C		
1Ø4	44	D		
1Ø5	45	E		
1Ø6	46	F		
1Ø7	47	G		
11Ø	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
12Ø	5Ø	P		
121	51	Q		
122	52	R		
123	53	S		
124	54	T		
125	55	U		
126	56	V		
127	57	W		
13Ø	58	X		
131	59	Y		
132	5A	Z		
133	5B			SHIFT K
134	5C			SHIFT L
135	5D			SHIFT M

ASCII CODES (Cont)

OCTAL	HEX	CHARACTER	CONTROL KEYB. EQUIV.	ALTERNATE CODE NAMES
136	5E	^		↑ SHIFT N
137	5F	–		← SHIFT O, UNDERSCORE
14Ø	6Ø	;		ACCENT GRAVE
141	61	a		
142	62	b		
143	63	c		
144	64	d		
145	65	e		
146	66	f		
147	67	g		
15Ø	68	h		
151	69	i		
152	6A	j		
153	6B	k		
154	6C	l		
155	6D	m		
156	6E	n		
157	6F	o		
16Ø	7Ø	p		
161	71	q		
162	72	r		
163	73	s		
164	74	t		
165	75	u		
166	76	v		
167	77	w		
17Ø	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

Errors encountered by FSP subroutines are relayed to the user in two ways: visually and under program control (see paragraph 1.5).

FSP error codes fall in the following categories:

1. System errors

    A. Overload conditions

    B. Malfunctions

2. Incorrect calling sequence

3. User parameter errors

4. Current defined page refresh exceeded

5. Miscellaneous

All of the FSP error codes require the user to correct the error described and rerun the user program. FSP continues to execute the user program after an error has occurred, but undefined results may be observed.

The normal running error code is ØØ and is displayed in the upper left corner of the indicator(s) unless modified by a call to ENBERR or DSAERR (see Section 9).

C-1. SYSTEM ERRORS

A. Overload Conditions - the following errors are caused when GCP+ buffers are overloaded and GCP+ cannot accept messages from or send messages to the host. These errors are detected when the EVENT routine requests a message from the GRAPHIC 7.

Error Code	Description	Action Initiating The Request
02	GCP+ output buffer full-PED coordinates cannot be sent to the host.	CALL REQTB
03	GCP+ output buffer full-GCP error message cannot be sent to the host.	-
04	GCP+ input buffer full-messages cannot be sent to GCP+	-
05	GCP+ output buffer full-function key or A/N keyboard information cannot be sent to the host	Pressing function or A/N key(s)
06	GCP+ output buffer full-hardcopy status or PHOTOPEN data messages cannot be sent to the host	CALL HCOPY or PHOTOPEN
08	GCP+ output buffer full-return image cannot be sent to the host	CALL REQIM

B. Malfunctions

Error code 13 is caused by the user calling routine ERASEP to delete refresh code when the current mark is beyond the end of the defined page. This situation can be encountered in edit mode.

The user should check the code before the offending call to ERASEP to insure that correct FSP edit commands have been used.

C-2. INCORRECT CALLING SEQUENCE

The following errors are caused by calling routines out of sequence. The user should check the order of the code in his program for the incorrect sequences described below.

Code	Cause	Routine Causing Error
21	PED is not connected to a symbol, check for call to TBALL to connect PED to a symbol.	REQTB, DTMODE
22	Call made to packed vector routine before call to ENBPMD or successive calls to DSAPMD.	PMOVE, PDRAW, DSAPMD
99	Error is caused when LAYOUT is called without first calling GSS4 routine or multiple calls to LAYOUT in the same routine.	LAYOUT

C-3. USER PARAMETER ERRORS

The following parameter errors are detected in user calls to FSP routines. The user should reread descriptions and check calls to named routines for the described errors.

Code	Description	Routine
30	Page number out of range	ADDREF
31	Page number or mark out of range	UPDATE
32	Page number out of range	PICTUR
33	Specifications too large	LAYOUT
34	Symbol cannot be moved by PED, mark specified is out of range	TBALL
35	Lamp number out of range	LAMPON, LAMPOF
37	Number of words greater than 20	REQIM
38	PED number out of range	DTINIT,DTMODE

C-4. CURRENT DEFINED PAGE REFRESH EXCEEDED

The following errors are caused by insufficient refresh on the page where the error is detected. The user should increase the offending page size in the call to LAYOUT and rerun the user's program.

Code	Routine Called when Refresh Overflow Occurred
40	MOVE or DRAW
41	TEXT
42	CIRCLE
43	POINT
44	PICTUR
45	CPARM
46	DPARM
47	STATUS
48	TBALL
49	REFDAT
50	PMOVE or PDRAW
51	COLOR
52	ITEM

C-5. MISCELLANEOUS

Error Code	Routine Causing Error
60	HCOPY
62	COPYIM
63	COPYIM
64	COPYIM
65	MOVEIM
66	MOVEIM

APPENDIX D

MEMORY USAGE

This table shows commonly used FSP user available subroutines which generate graphic orders in the GRAPHIC 7 memory. Note, however, that certain FSP internal routines may insert refresh in the user's pages. The number of words of refresh created by internal routines is relatively small for normal applications.

FSP SUBROUTINE	NUMBER OF (16 BIT) WORDS OF GRAPHIC 7 REFRESH MEMORY FILLED	ASSOCIATED GRAPHIC ORDERS (See GRAPHIC 7 GCP+ Programmer's Reference Manual for further description of the graphic orders mentioned.)
MOVE	0, 1, or 2	Depending on optimization, sends appropriate "load" and "move" graphic orders.
DRAW	0, 1, or 2	Depending on optimization, sends appropriate "move" and "draw" graphic orders.
TEXT	(N+1)/2	N = number of characters of text.
POINT	1	Point plot graphic order
CIRCLE	2	LDKX, DRKY
CPARM	2	LDDP, LDTI
DPARM	1	LDDP
STATUS	1	LDDZ
COLOR	2	LDRI and indicator/color
ITEM	2	LDRI and item (ID) number
CCINIT	11	HREF and LDRIs
CCVAL	11	HREF and LDRIs
CCOFF	2	LDRI and Ø.
CCON	2	LDRI and 2.
PMOVE	2	(*see note)
PDRAW	2 max. per draw	(*see note)

FSP SUBROUTINE	NUMBER OF (16 BIT) WORDS OF GRAPHIC 7 REFRESH MEMORY FILLED	ASSOCIATED GRAPHIC ORDERS (See GRAPHIC 7 GCP+ Programmer's Reference Manual for further description of the graphic orders mentioned.)
PICTUR	3 max.	CALL address or HALT, LDRI combination
LAYOUT	-	Inserts one graphic "RETURN" per user page.

*PMOVE and PDRAW are generally used to create large blocks of X, Y move and draw data. The number of words of refresh created is a function of the number of PMOVEs and PDRAWs executed and the amount of optimization of X, Y move and draw data.

APPENDIX E

PROGRAMMING EXAMPLES

EXAMPLE 1

This example illustrates the use of subroutine calls GSS4, LAYOUT, SCALE and THEEND.

```
C
C     EXAMPLE PROGRAM 1
C
      DIMENSION LPAGES(5)
      DATA LPAGES /1ØØØ, 1ØØ, 1ØØ, 1Ø, 1Ø/

C     INITIALIZE AND USE LOGICAL UNIT NUMBER 5.  TRANSMISSION
C     WILL BE OVER THE PARALLEL INTERFACE
C
      CALL GSS4(5,Ø,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 1Ø24.
C
      CALL SCALE(Ø.Ø,Ø.Ø,1Ø23.Ø,1Ø23.Ø)
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 EXAMPLE 2-PICTURE 1 on the screen. It illustrates user calls to the following subroutines.

GSS4

LAYOUT

SCALE

ADDREF

MOVE

DRAW

POINT

PICTUR

TEXT

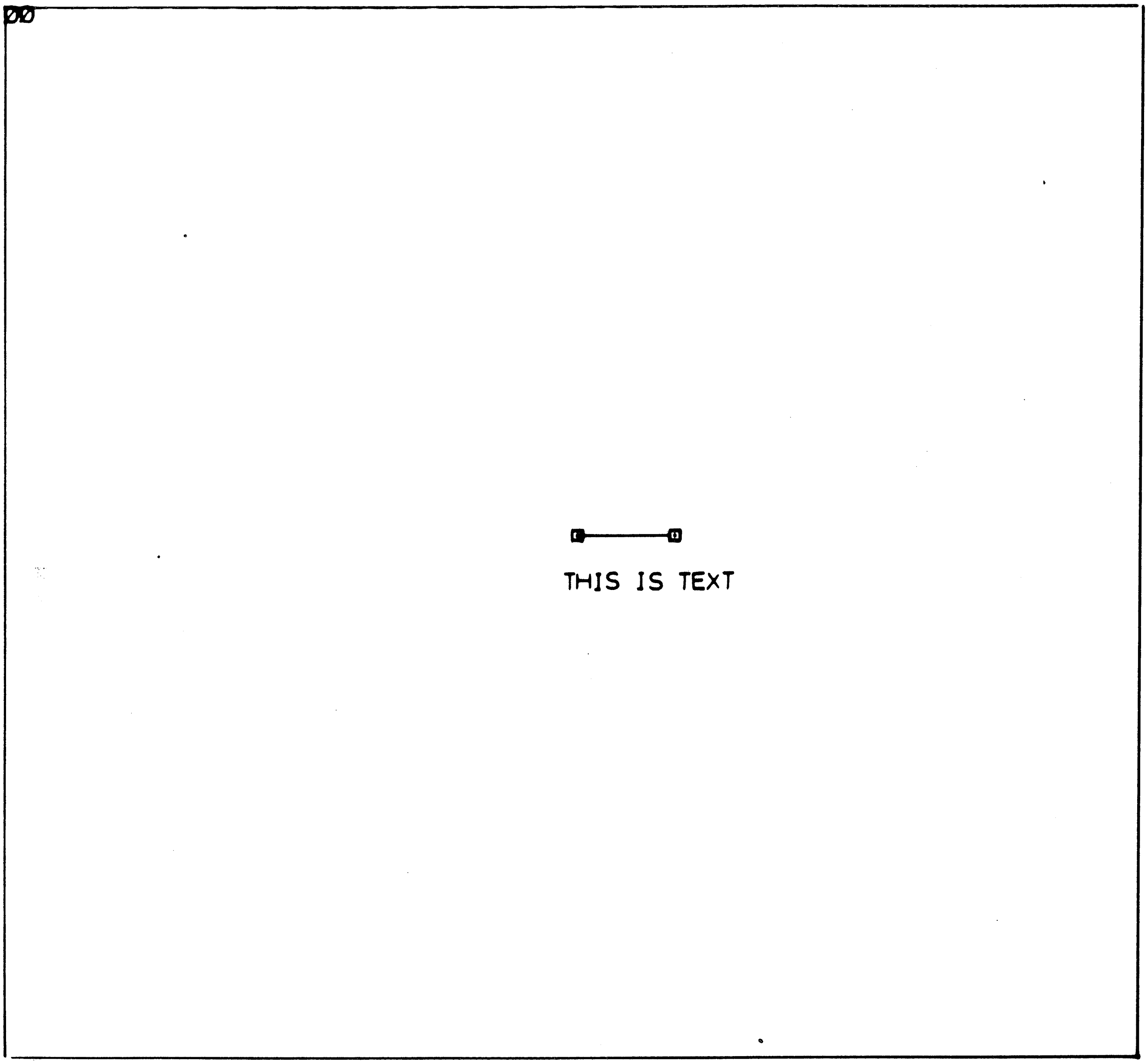
EVENT

GETKEY

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 most upper right function key.



EXAMPLE 2 - PICTURE 1

```
C
C ************************FSP SAMPLE-2  ****************************
C THIS SAMPLE PROGRAM TESTS SOME OF THE FEATURES OF THE FSP
C FORTRAN PACKAGE.
C
        DIMENSION LPAGES(5),ITEX(6)
        DATA LPAGES /1000,100,100,10,10/
C
C INITIALIZE AND USE LOGICAL UNIT NUMBER 5. TRANSMISSION
C WILL BE OVER THE SERIAL INTERFACE
C
        CALL GSS4(5,0,2)
C
C SPECIFY 5 PAGES OF USER DATA, EACH LENGTH
C BEING AS 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.,1023.)
C
C THE FOLLOWING EXAMPLE ILLUSTRATES THE USES OF ADDREF,
C MOVE,DRAW,TEXT,POINT,PICTUR,EVENT,GETKEY
C NOTE: TEXT IS DISPLAYED VIA A SUBROUTINE UNIQUE TO OUR
C HOST COMPUTER WHICH IS A PDP-10, THIS SUBROUTINE IS CALLED
C 'SETEXT'.
C
C OPEN PAGE 1 FOR TOP LEVEL DRAWING.
C
        CALL ADDREF(1)
C
C POSITION THE BEAM TO THE CENTER OF THE SCREEN
C
        CALL MOVE(512.,512.,0)
C
C DRAW A SMALL BOX IN PAGE 2
C NOTE THAT THESE MOVES AND DRAWS ARE ALL RELATIVE
C
        CALL ADDREF(2)
C
C UP AND RIGHT BY 5
C
        CALL MOVE(5.,5.,1)
C
C RIGHT SIDE OF BOX
C
        CALL DRAW(0.,-10.,1)
C
C
C BOTTOM OF BOX
C
        CALL DRAW(-10.,0.,1)
C
C LEFT SIDE OF BOX
C
        CALL DRAW(0.,10.,1)
C
C TOP OF BOX
```

```
C
        CALL DRAW(10.,0.,1)
C
C RETURN TO CENTER
C
        CALL MOVE(-5.,-5.,1)
C
C PUT A POINT IN THE CENTER
C
        CALL POINT
C
C NOW GO BACK TO BUILDING PAGE 1
C
        CALL ADDREF(1)
C
C INSERT A SUBROUTINE CALL TO CUR LITTLE BOX. THIS WILL
C SHOW IT IN THE CENTER OF THE SCREEN.
C
        CALL PICTUR(2)
C
C DRAW A LINE FROM THE CENTER OF THIS BOX TO THE RIGHT A LITTLE.
C
        CALL DRAW(600.,512.,0)
C
C NOW DRAW ANOTHER LITTLE BOX HERE.
C
        CALL PICTUR(2)
C
C       PLACE SOME TEXT JUST BELOW THE BOXES.
C
        CALL MOVE(500.,460.,0)
        CALL SETEXT('THIS ',0)
        CALL SETEXT('IS TE',0)
        CALL SETEXT('XT',12)
C
C PICTURE SHOULD BE FINISHED NOW
C
C USE THE 'EVENT' ROUTINE TO FIND A KEYBOARD STRIKE
C FROM FUNCTION KEY NUMBER 31 TO EXIT
C
100     CALL EVENT(I)
C IF I IS 4 THEN A KEY WAS HIT. OTHER WISE GO BACK TO EVENT AND
C WAIT FOR ONE.

        WRITE(1,110)I
110     FORMAT(' EVENT: ',I5)
C
        IF(I.NE.4)GO TO 100
C
C WILL COME HERE FOR KEYBOARD HIT, NOW GO READ KEY NUMBER
C
        CALL GETKEY(KBD,KEY)
C
C NOW SEE IF IT WAS KEY NUMBER 31. IF NOT GO BACK TO
C EVENT AND WAIT FOR IT.
C
        IF(KEY.NE.31)GO TO 100
C
C WILL COME HERE IF KEY 31 WAS HIT. SO NOW WE ARE DONE.
C SHUT DOWN THE DISPLAY AND RETURN TO THE TTY EMULATOR.
```

```
          CALL THEEND
          END
C
C     *******************SETEXT SUBROUTINE FOR THE PDP-10********************
C
C     SUBROUTINE TO PUT TEXT RIGHT ADJUSTED IN 5-D ARRAY
C
C         THIS SUBROUTINE SETEXT IS USED TO SET THE ARRAY FOR TEXT
C     IN THE  FSP  ATP.  TEXT IS TRANSMITTED IN AN ARRAY WITH ONE
C     7 BIT ASCII CHARACTER, RIGHT ADJUSTED, IN EACH ELEMENT.
C     MOST OF THE TEXT IN THE  FSP  ATP IS CALLED USING SETEXT
C     THE FORMAT IS:
C         CALL        SETEXT('AAAAA',IA)
C     WHERE IA IS THE TOTAL NUMBER OF CHARACTERS AND AAAAA IS 1 TO A MAX
C     OF 5 CHARACTERS SURROUNDED BY SINGLE QUOTES.  THIS EVOLED FROM
C     THE PDP-10 WHOSE 36 BIT WORDS HOLD 5 ASCII CHARACTERS.  TO SEND
C     A TEXT STRING OF MORE THAN 5 CHARACTERS, MAKE SEVERAL CALLS
C     OF 5 CHARACTERS EACH  WITH IA=0.  THIS WILL ADD
C     THE 5 CHARACTERS TO THE END OF AN ARRAY.  ON THE LAST CALL
C     TO SETEXT, SET IA EGUAL TO THE TOTAL NUMBER OF CHARACTERS.
C     SETEXT DOES THE ACTUAL CALL TO THE TEXT SUBROUTINE IN HCP
C     WHEN THE USER IS DONE.
C
        SUBROUTINE SETEXT(IWRD,IA)
        DIMENSION IARRAY(5),IARY(100)
        DATA IAD/1/
        IWORD=IWRD
        ITEST=IWORD
C
C     THIS MOVES THE 5 ASCII CHAR 1 BIT TO THE RIGHT TO MAKE
C     IT RIGHT ADJUSTED
        IWORD=IWORD/2
C
C     THE ARRAY GOES FROM 5 TO 1 BECAUSE THAT IS THE ORDER THE
C     CHARACTERS ARE RETRIEVED FROM IWORD
         DO 100 I=1,5
C
C     TAKE THE 7 RIGHT BITS, THE NEXT CHARACTER
        IARRAY(6-I)=IWORD.AND."177
        ITEST=IWORD
C     SHIFT IWORD RIGHT 7 BITS TO GET THE NEXT CHAR RIGHT ADJUSTED
        IWORD=IWORD/(2**7)
C
C     THE PDP-10 TRUNCATES, SO CONTINUE IF IWORD IS POSITIVE
        IF(IWORD.GE.0)GO TO 100
C
C     IF IWORD IS NEGATIVE,MAKE SURE THERE ISN'T A TRUNCATION ERROR
        IF(ITEST.NE.IWORD*(2**7))IWORD=IWORD-1
  100   CONTINUE
C
C     TRANSFER THE ASCII CHARACTERS TO THE OUTPUT ARRAY
        DO 101 J=1,5
        IARY(IAD)=IARRAY(J)
        IAD=IAD+1
  101   CONTINUE
C
C     IF THIS IS JUST AN ADDITION TO THE OUTPUT ARRAY, RETURN
        IF(IA.EQ.0) RETURN
C
C     RESET IAD TO THE START OF THE OUTPUT ARRAY
```

```
17900            IAD=1
18000     C
18100     C   SEND OUT TEXT
18200            CALL       TEXT(IA,IARY)
18300            RETURN
18400            END
18500     C
```

EXAMPLE 3

This sample program, when executed, displays EXAMPLE 3 - PICTURE 1 on the screen. It illustrates user calls to:

GSS4

LAYOUT

SCALE

ADDREF

MOVE

DRAW

POINT

PICTUR

TEXT

EVENT

GETKEY

GETMRK

REQMRK

UPDATE

ENBPEN

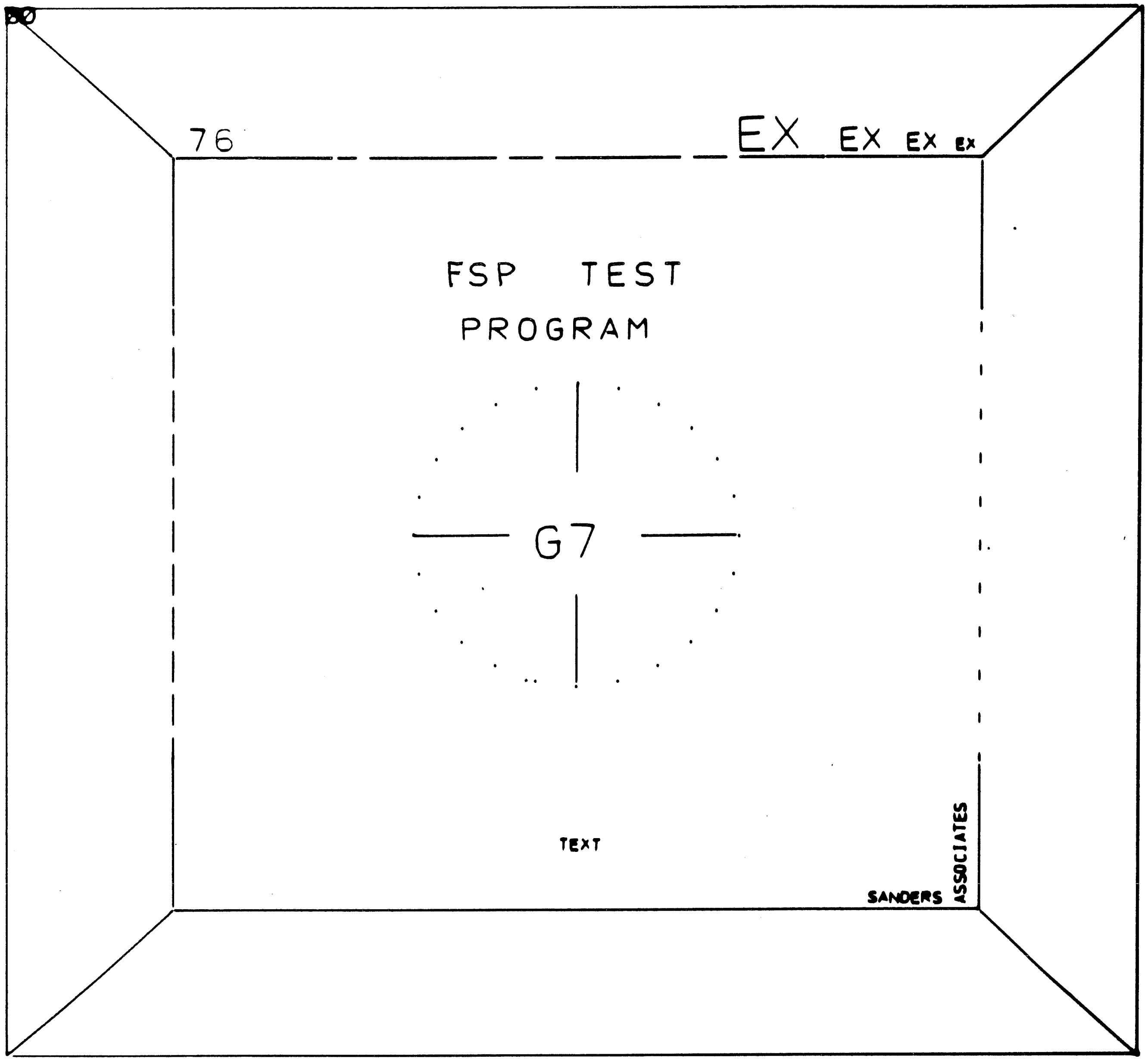
DPARM

CPARM

The text "TEXT" is sensitive to PHOTOPEN strikes and will BLINK or NOT BLINK with successive PHOTOPEN hits.

NOTE

> After full picture is displayed, to return to the TTY emulator, hit most upper right function key.



EXAMPLE 3 - PICTURE 1

```
C
C ************************ FSP SAMPLE-3*******************************
C
C    THIS PROGRAM IS A SAMPLE TEST PROGRAM THAT TESTS SOME
C    FEATURES OF THE FORTRAN PACKAGE FSP . IN ORDER TO USE THIS
C    PROGRAM A SUBROUTINE MUST BE WRITTEN TO CORRECTLY FORMAT THE
C    TEXT AND OUTPUT THE ARRAY. THE CALLING SEQUENCE IS:
C
C         CALL    SETEXT('AAAAA',IA)
C
C    WHERE IA IS AN INTEGER OF THE TOTAL NUMBER OF CHARACTERS BEING
C    SENT AND AAAAA IS 1 TO A MAX OF 5 CHARACTERS SURROUNDED BY
C    SINGLE QUOTES. THIS EVOLVED FROM USE ON THE PDP-1Ø WHOSE
C    36 BIT WORD HOLDS 5 ASCII CHARACTERS. NOTE: TO SEND A STRING
C    OF MORE THAN 5 CHARACTERS, MAKE SEVERAL CALLS OF 5 CHARACTERS
C    EACH WITH IA=Ø. THIS WILL ADD EACH ADDITIONAL 5 CHARACTERS
C    TO THE END OF AN OUTPUT ARRAY. ON THE LAST CALL TO SETEXT
C    SET IA EQUAL TO THE TOTAL NUMBER OF CHARACTERS TO BE OUTPUT
C    AND THE SUBROUTINE SETEXT WILL MAKE THE CALL TO THE SUBROUTINE
C    TEXT IN HCP.
          DIMENSION ILENG(2)
C INITIALIZE PARAMETER VALUES TO DEFAULT VALUES
C
C IBL=Ø - NO BLINKING
C INT=7 - INTENSITY OF 7
C IVT=Ø - SOLID LINE
C IND=15 - ALL INDICATORS
C
C ICSIZE=Ø - SMALLEST CHARACTER SIZE
C ICROT=Ø - NO ROTATION OF CHARACTERS
C ICSPAC=1Ø - SPACING BETWEEN CHARACTERS(IN DITS)
C
C ISP=Ø - FAST DRAWING RATE
C ISYNC=1 - 6Ø HZ
C IPEN=Ø - PHOTOPENS DISABLED
C
C ILENG= LENGTH OF REFRESH PAGES
C
          DATA IBL,INT,IVT,IND/Ø,7,Ø,15/
          DATA ICSIZE,ICROT,ICSPAC/Ø,Ø,1Ø/
          DATA ISP,ISYNC,IPEN/Ø,1,Ø/
          DATA ILENG/256,168/
C
C
C PLAG IS AN INDICATOR SPECIFYING WWHETHER
C THE TEXT 'TEXT' IS BLINKING OR NOT.
C IF PFLAG=Ø, IT IS NOT BLINKING
C IF PFLAG=1, IT IS BLINKING
C
          PFLAG = Ø
C  INITIALIZE  FSP  USING LOGICAL UNIT 5 WITH SERIAL INTERFACE
          CALL    GSS4(5,Ø,2)
C
C
C SPECIFY 2 PAGES OF USER DATA ,EACH LENGTH BEING AS DESCRIBED
C IN THE 'ILENG' DATA STATEMENT ABOVE.
          CALL    LAYOUT(2,ILENG)
C
C
C SPECIFY USER COORDINATE SYSTEM WITH LOWER LEFT
```

```
06100     C AT (0.,0.) AND UPPER RIGHT AT (700.,700.)
06200     C
06300                CALL    SCALE(0,0,0,0,700.,0,700.,0)
06400     C ENABLE PHOTOPEN 2
06500     C
06600                CALL    ENBPEN(2)
06700     C
06800     C TURN ALL LAMPS OFF OK KEYBOARD
06900     C
07000                CALL    LAMPOF(1,-1)
```

```
C
C START AT BEGINNING OF PAGE 1
C
C
C THIS SECTION TESTS THE VARIOUS LINE TYPES
C
C
C DRAW BORDER
C
              CALL     MOVE(1ØØ.,1ØØ.,Ø)
              CALL     DRAW(6ØØ.,1ØØ.,Ø)
              CALL     DRAW(Ø.,1ØØ.,1)
C DRAW DOTTED LINE
C
C CALL STATUS TO CHANGE LINE TYPE
C IVT=1 SPECIFIES DRAW DOTTED VECTORS
C
              IVT=1
 99           CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(Ø.,3ØØ.,1)
C BACK TO SOLID
C CALL STATUS TO CHANGE LINE TYPE
C IVT=Ø SPECIFIES SCLID VECTORS
C
              IVT=Ø
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(6ØØ.,6ØØ.,Ø)
              CALL     DRAW(-1ØØ.,Ø.,1)
C DRAW DOT-DASH
C
C CALL STATUS TO CHANGE LINE TYPE
C IVT=3 SPECIFIES DOT-DASH LINE
C
              IVT=3
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(-3ØØ.,Ø.,1)
C BACK TO SOLID
C CALL STATUS TO CHANGE LINE TYPE
C IVT=Ø SPECIFIES SCLID VECTORS
C
              IVT=Ø
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(1ØØ.,6ØØ.,Ø)
              CALL     DRAW(Ø.,-1ØØ.,1)
C DRAW DASH
C CALL STATUS TO CHANGE LINE TYPE
C IVT=2 SPECIFIES DASH VECTORS
C
              IVT=2
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(Ø.,-3ØØ.,1)
C BACK TO SOLID
C CALL STATUS TO CHANGE LINE TYPE
C IVT=Ø SPECIFIES SCLID VECTORS
C
              IVT=Ø
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     DRAW(1ØØ.,1ØØ.,Ø)
              CALL     DRAW(Ø.,Ø.,Ø)
              CALL     MOVE(7ØØ.,Ø.,1)
```

```
Ø61ØØ          CALL    DRAW(6ØØ.,1ØØ.,Ø)
Ø62ØØ          CALL    MOVE(6ØØ.,6ØØ.,Ø)
Ø63ØØ          CALL    DRAW(7ØØ.,7ØØ.,Ø)
Ø64ØØ          CALL    MOVE(-7ØØ.,Ø.,1)
Ø65ØØ          CALL    DRAW(1ØØ.,-1ØØ.,1)
Ø66ØØ          CALL    MOVE(25Ø.,35Ø.,Ø)
Ø67ØØ          CALL    DRAW(6Ø.,Ø.,1)
Ø68ØØ          CALL    MOVE(39Ø.,35Ø.,Ø)
Ø69ØØ          CALL    DRAW(6Ø.,Ø.,1)
Ø7ØØØ          CALL    MOVE(35Ø.,45Ø.,Ø)
Ø71ØØ          CALL    DRAW(Ø.,-6Ø.,1)
Ø72ØØ          CALL    MOVE(35Ø.,31Ø.,Ø)
Ø73ØØ          CALL    DRAW(Ø.,-6Ø.,1)
```

```
C
C PAGE 1 (CONT) - TEST PARAMETERS AND TEXT
C THIS SECTION TESTS THE VARIOUS CHARACTER SIZES, INTENSITIES
C AND CHARACTER ROTATION.
C
C
C
C START WITH INITIAL VALUES.,CHARACTER SIZE Ø, NCRCTATE,
C AND SPACING OF 1Ø DITS
C
          CALL     CPARM(ICSIZE,ICROT,ICSPAC)
          CALL     MOVE(53Ø.,1Ø5.,Ø)
C
C NOW DO SOME TEXT. SEE DESCRIPTION ABOVE FOR HOW SETEXT WORKS
C
          CALL     SETEXT('SANDE',Ø)
          CALL     SETEXT('RS  ',9)
C NOW TEST ROTATE
C CALL CPARM TO CHANGE CHARACTER ROTATE
C ICROT=1 SPECIFIES ROTATE CHARACTER 9Ø DEGREES CCUNTERCLOCKWISE.
C
          ICROT=1
          CALL     CPARM(ICSIZE,ICROT,ICSPAC)
          CALL     SETEXT('ASSOC',Ø)
          CALL     SETEXT('IATES',1Ø)
C NOW TEST FOUR CHARACTER SIZES AND SPACING
C CALL CPARM TO CHANGE CHARACTER SIZE
C ICSIZE=Ø SPECIFIES SMALLEST
C ICSIZE=1 SPECIFIES NEXT TO SMALLEST
C ICSIZE=3 SPECIFIES NEXT TO LARGEST
C ICSIZE=4 SPECIFIES LARGEST
C
C ICSPAC WILL BE CHANGE AS EACH CHARACTER SIZE IS CHANGED TO
C ALLOW FOR DECENT SPACING BETWEEN CHARACTERS.
C
          CALL     MOVE(45Ø.,6Ø5.,Ø)
C SET ROTATE OFF,CHAR:4,SPACING:3Ø
          ICROT=Ø
          CALL     CPARM(3,Ø,3Ø)
          CALL     SETEXT('EX ',3)
C SET CHAR:3,SP:2Ø
          CALL     CPARM(2,ICROT,2Ø)
          CALL     SETEXT('EX ',3)
C SET CHAR:2,SP:15
          CALL     CPARM(1,ICROT,15)
          CALL     SETEXT('EX ',3)
C SET CHAR:1,SP:1Ø
          CALL     CPARM(Ø,ICROT,1Ø)
          CALL     SETEXT('EX ',3)
C PRINT THE TITLE
          CALL     MOVE(25Ø.,516.,Ø)
C SET CHAR:3,SP:2Ø
          CALL     CPARM(2,ICROT,25)
          CALL     SETEXT('GSS4 ',Ø)
          CALL     SETEXT(' TEST',1Ø)
          CALL     MOVE(28Ø.,48Ø.,Ø)
          CALL     SETEXT('PROGR',Ø)
          CALL     SETEXT('AM ',7)
C TEST BRIGHTNESS
C CALL  STATUS TO CHANGE INTENSITY
```

```
C INTENSITY RANGES FROM 0 TO 7.
C  0 IS INVISIBLE
C  1 IS VERY DIM
C  7 IS THE BRIGHTEST
C
              CALL     MOVE(110.,605.,0)
              DO 200 J=1,7
C CHANGE INTENSITY TO J
              INT=8-J
              CALL     STATUS(IBL,INT,IVT,IND)
C
C WE WILL CALL A ROUTINE TO CONVERT OUT INTERGER 'J' TO
C AN ASCII CHARACTER WHICH WE CAN DISPLAY ON THE SCREEN.
C
C CALCULATE 7-BIT ASCII FOR INTENSITY NUMBER
              ITX=48+8-J
C
C CALL TEXT TO DISPLAY THE CHARACTER JUST COMPUTEC,WE USE 'TEXT'
C RATHER THAN 'SETEXT' BECAUSE THE CHARACTER IS SET UP IN THE
C INTEGER ARRAY PROPERLY ALREADY.
C
              CALL     TEXT(1,ITX)
200           CONTINUE
C TEST BLINK
C CALL STATUS TO CHANGE BLINK
C IBL=1 SPECIFIES BLINK
C
C
C TURN BLINK ON,SET INTENSITY TO 7; CHAR:4,SP:30
              CALL     CPARM(3,ICROT,30)
              INT=7
              IBL=1
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     MOVE(325.,335.,0)
              CALL     SETEXT('G7',2)
C TURN BLINK OFF; SET CHAR:1,SP:10
C CALL STATUS TO TURN BLINK OFF
C IBL=0 SPECIFIES NO BLINK
C
              IBL=0
              CALL     STATUS(IBL,INT,IVT,IND)
              CALL     CPARM(ICSIZE,ICROT,ICSPAC)
```

```
C
C PAGE 1 (CONT) - DRAW POINT CIRCLE
C THIS SECTION DEMONSTRATES THE USE OF SUBROUTINE PCINT
C WE WILL DRAW A CIRCLE CONSISTING OF 24 POINTS
C
C
        DO 100 I=1,24
C
C WE DO SOME ARITHEMETIC COMPUTATIONS TO COMPUTE X,Y VALUES AT
C WHICH WE WILL DRAW A POINT.
C
        ANGLE=((I*15.)/180.)*3.14159
        XPT=350.+100.*COS(ANGLE)
        YPT=350.+100.*SIN(ANGLE)
C DO A MOVE TO THAT X, AND Y AND PLOT A POINT.
        CALL    MOVE(XPT,YPT,0)
        CALL    POINT
C
C DO FOR ALL 24 POINTS
C
100     CONTINUE
C  END OF PAGE 1
```

```

C
C  PAGE 2
C THIS SECTION GENERATES GRAPHIC DATA ON PAGE 2, THIS PAGE WILL
C THEN BE CALLED BY PAGE 1 SHOWING SUBROUTINE LINKAGES, NOTE THAT
C PAGE 2 WILL NOT BE DISPLAYED UNLESS AND UNTIL PAGE 1 CALLS IT,
C
C
C
C OPEN PAGE 2 FOR DATA
C
      CALL ADDREF(2)
      CALL MOVE(340.,140.,0)
C  SENSITIZE DATA TC PHOTOPEN
C WE WANT TO USE THE PHOTOPEN TO MAKE THE TEXT 'TEXT' IN THIS
C SECTION BLINK OR NOT BLINK ON COMMAND, WE MUST, THERFORE
C SENSITIZE THE DATA WE WANT ,WE DO THAT BY CALLING
C DPARM WITH A PARAMETER SPECIFYING ENABLE PHOTPEN 2
C
      CALL DPARM(ISP,ISYNC,2)
C
C NOW WE WANT TO KEEP TRACK OF THE DATA WORD THAT CONTROLS THE
C BLINK, TO DO THIS WE REQUEST THE MARK OF THE NEXT AVAILABLE
C LOCATION INTO WHICH WE WILL PUT THE CALL TO STATUS TO CONTROL
C THE BLINKING,
C CALL REQMRK DOES THIS
C NEXT CALL EVENT TO SEE IF THE DATA IS READY TO BE SENT TO US.
C CALL EVENT (I) DOES THIS
C IF THE EVENT SPECIFIES THAT IT HAS A MARK RESPONSE,
C CALL GETMARK TO RETRIEVE IT,
C CALL GETMARK(INAL) DOES THIS
C SAVE THE MARK FOR FUTURE USE TO UPDATE THE BLINKING
C CONTROL WORD,
C
      CALL REQMRK
10    CALL EVENT(I)
C
C IF I=7, THEN EVENT HAS A MARK RESPONSE, OTHERWISE
C GO WAIT FOR IT,
C
      IF (I .NE. 7) GO TO 10
C EVENT SAYS MARK RESPONSE READY, GO GET MARK
      CALL GETMRK(INAL)
C INAL WILL BE USED LATER FOR UPDATING
C PUT IN BLINKING CONTROL WORD TO BE ALTERED
C IN RESPONSE TO PHOTOPEN STRIKES,
      CALL STATUS(IBL,INT,IVT,IND)
C
C THIS IS THE TEXT THAT WILL BLINK
C
      CALL SETEXT ('TEXT',4)
C
C IS SENSITIZED,
C
      CALL DPARM(ISP,ISYNC,0)
C
C
C
C  NOW MODIFY PAGE 1 TO CALL PAGE 2
      CALL ADDREF(1)
```

```
          CALL PICTUR(2)
C
C
C  NOW WAIT FOR PHOTOPEN STRIKE AND GET INFORMATICN
C OR FOR FUNCTION KEY 31 TO EXIT
C
C
2Ø        CALL EVENT(I)
C
C ON RETURN IF I=4, THEN THERE WAS A KEYBOARD STRIKE
C    GO GET KEYBOARD DATA
C IF I=5, THEN THERE WAS A PHOTOPEN STRIKE; GO GET DATA
C OTHERWISE, GO WAIT FOR ANOTHER EVENT
C
          IF (I .EQ. 4) GO TO 1Ø1
          IF (I .NE. 5) GO TO 2Ø
C  GOT PHCTOPEN STRIKE, GO GET DATA
          CALL GETPEN(IPEN,IMPAGE,MARK,ITYPE,ICPAGE,IBYTE,ITMNUM)
C  THIS INFORMATION CAN BE USED FOR VARIOUS PURPOSES.  THIS TEST
C  PROGRAM IS NOT CONCERNED WITH THE DATA AND DOES NOTHING WITH
C  IT.  TO ACKNOWLEDGE THAT WE GOT A PHOTOPEN STRIKE, THOUGH,
C 'TEST' WILL BLINK OR NOT BLINK, DEPENDING ON PRESENT STATE.
C IF PFLAG=Ø, 'TEXT' IS NOT BLINKING, GO BLINK IT
C IF PFLAG=1, 'TEXT' IS BLINKING, GO STOP BLINKING
C
          IF (PFLAG .EQ. 1) GO TO 3Ø
          PFLAG = 1
C
C TO ALTER WORD CONTROLLING BLINKING, CALL UPDATE(2,INAL)
C WHERE 2 SPECIFIES PAGE 2 AND INAL SPECIFIES THE MARK OF
C THE DATA WE ARE GOING TO CHANGE.
C
          CALL UPDATE(2,INAL)
C
C ALL DATA NOW WILL BE ENTERED ON PAGE 2 STARTING
C AT MARK INAL.
C
          CALL STATUS (1,INT,IVT,IND)
          GO TO 2Ø
C 'TEXT WAS BLINKING, STOP BLINKING NOW
3Ø        PFLAG = Ø
C
C TO ALTER WORD CONTROLLING BLINKING, CALL UPDATE(2,INAL)
C WHERE 2 SPECIFIES PAGE 2 AND INAL SPECIFIES THE MARK OF
C THE DATA WE ARE GOING TO CHANGE.
C
          CALL UPDATE (2,INAL)
          CALL STATUS (IBL,INT,IVT,IND)
          GO TO 2Ø
C
C EVENT SAYS WE GOT A FUNCTION KEY HIT
C
C  GO GET FUNCTION KEY NUMBER AND TEST FOR #31 TO EXIT
1Ø1       CALL GETKEY(KBD,KEY)
          IF (KEY .NE. 31) GO TO 2Ø
C
C IT WAS FUNCTION KEY 31, SHUT OFF DISPLAY AND RETURN TO
C TTY EMULATOR.
C
          CALL THEEND
```

```
12200           END
```

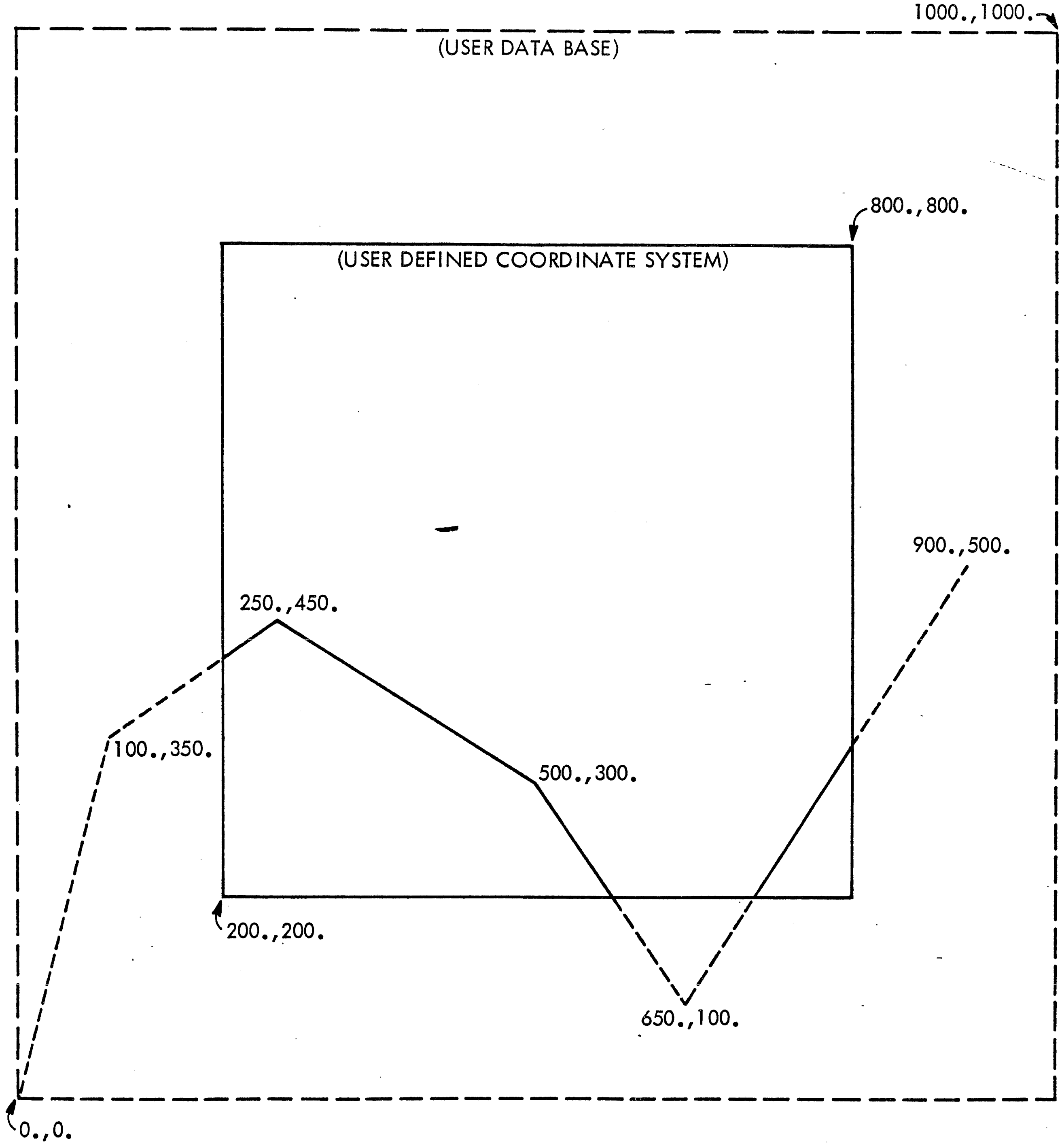
```
C
C     *******************SETEXT SUBROUTINE FOR THE PDP-10********************
C
C     SUBROUTINE TO PUT TEXT RIGHT ADJUSTED IN 5-D ARRAY
C
C         THIS SUBROUTINE SETEXT IS USED TO SET THE ARRAY FOR TEXT
C     IN THE  FSP  ATP.  TEXT IS TRANSMITTED IN AN ARRAY WITH ONE
C     7 BIT ASCII CHARACTER, RIGHT ADJUSTED, IN EACH ELEMENT.
C     MOST OF THE TEXT IN THE  FSP  ATP IS CALLED USING SETEXT
C     THE FORMAT IS:
C          CALL        SETEXT('AAAAA',IA)
C     WHERE IA IS THE TOTAL NUMBER OF CHARACTERS AND AAAAA IS 1 TO A MAX
C     OF 5 CHARACTERS SURROUNDED BY SINGLE QUOTES.  THIS EVOLED FROM
C     THE PDP-10 WHOSE 36 BIT WORDS HOLD 5 ASCII CHARACTERS.  TO SEND
C     A TEXT STRING OF MORE THAN 5 CHARACTERS, MAKE SEVERAL CALLS
C     OF 5 CHARACTERS EACH  WITH IA=0.  THIS WILL ADD
C     THE 5 CHARACTERS TO THE END OF AN ARRAY.  ON THE LAST CALL
C     TO SETEXT, SET IA EQUAL TO THE TOTAL NUMBER OF CHARACTERS.
C     SETEXT DOES THE ACTUAL CALL TO THE TEXT SUBROUTINE IN HCP
C     WHEN THE USER IS DONE.
C
          SUBROUTINE SETEXT(IWRD,IA)
          DIMENSION IARRAY(5),IARY(100)
          DATA IAD/1/
          IWORD=IWRD
          ITEST=IWORD
C
C     THIS MOVES THE 5 ASCII CHAR 1 BIT TO THE RIGHT TO MAKE
C     IT RIGHT ADJUSTED
          IWORD=IWORD/2
C
C     THE ARRAY GOES FROM 5 TO 1 BECAUSE THAT IS THE ORDER THE
C     CHARACTERS ARE RETRIEVED FROM IWORD
          DO 100 I=1,5
C
C     TAKE THE 7 RIGHT BITS, THE NEXT CHARACTER
          IARRAY(6-I)=IWORD.AND."177
          ITEST=IWORD
C     SHIFT IWORD RIGHT 7 BITS TO GET THE NEXT CHAR RIGHT ADJUSTED
          IWORD=IWORD/(2**7)
C
C     THE PDP-10 TRUNCATES, SO CONTINUE IF IWORD IS POSITIVE
          IF(IWORD.GE.0)GO TO 100
C
C     IF IWORD IS NEGATIVE,MAKE SURE THERE ISN'T A TRUNCATION ERROR
          IF(ITEST.NE.IWORD*(2**7))IWORD=IWORD-1
100       CONTINUE
C
C     TRANSFER THE ASCII CHARACTERS TO THE OUTPUT ARRAY
          DO 101 J=1,5
          IARY(IAD)=IARRAY(J)
          IAD=IAD+1
101       CONTINUE
C
C     IF THIS IS JUST AN ADDITION TO THE OUTPUT ARRAY, RETURN
          IF(IA.EQ.0) RETURN
C
C     RESET IAD TO THE START OF THE OUTPUT ARRAY
          IAD=1
C
```

```
C    SEND OUT TEXT
         CALL         TEXT(IA,IARY)
         RETURN
         END
C
```

EXAMPLE 4

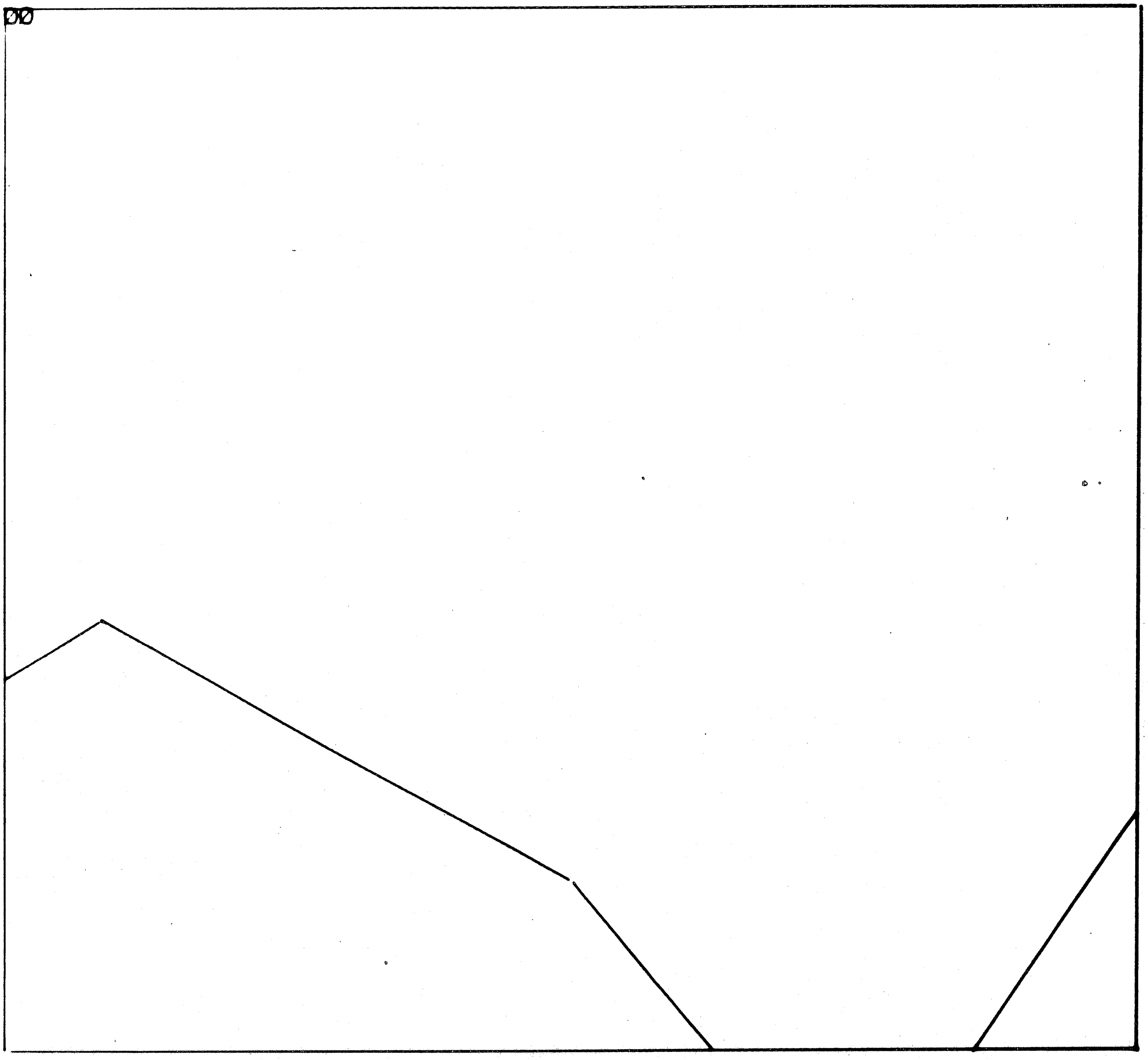
This sample program illustrates FSP subroutine CLIP. The data base and its relationship to the user defined coordinate system are shown in SAMPLE 4 - PICTURE 1. The actual displayed picture after clipping is shown in SAMPLE 4 - PICTURE 2.

After picture is displayed, to return to the TTY emulator, hit upper right most function key.



SAMPLE 4 - PICTURE 1

This diagram shows the data points in the USERS DATA BASE in relation to the user defined coordinate system. Picture 2 shows how the display will look with clipped data.



SAMPLE 4 - PICTURE 2

This is what the final display will look like with data clipped.

```
C
C ************************ FSP  SAMPLE-4*******************************
C
C THIS PROGRAM IS A SAMPLE PROGRAM THAT TESTS THE  FSP
C SUBROUTINE CLIP.
C
C THE USERS COORDINATE SYSTEM IS DEFINED AS (200.,220.)LOWER LEFT
C AND (800.,800.) UPPER RIGHT.
C X AND Y DATA RANGES FROM (0.,0.)LOWER LEFT TO (1000.,1000.)
C UPPER RIGHT.
C EACH SET OF DATA POINTS IS FED TO SUBROUTINE CLIP.
C CLIP PASSES BACK PARAMETERS TELLING THE USER WHETHER
C THE POINT WAS ON SCREEN OR OFF OR PART ON AND PART OFF,AND
C  WHETHER TO DO A MOVE,OR A DRAW OR A MOVE AND DRAW TO CREATE
C THE DESIRED CLIPPED PICTURE.
C
        DIMENSION X(6),Y(6)
        DATA ILENG/1000/
C
C DEFINE OUR DATA BASE OF X AND Y POSITIONS
C
        DATA X/0.,100.,250.,500.,650.,900./
        DATA Y/0.,350.,450.,300.,100.,500./
C
C
C INITIALIZE  AND USE LOGICAL UNIT NUMBER 5, TRANSMISSION
C WILL BE OVER THE SERIAL INTERFACE
C
        CALL GSS4(5,0,2)
C
C SPECIFY 1 PAGE OF USER DATA 'ILENG' IN LENGTH WHICH IS 1000 WORDS.
C
        CALL LAYOUT(1,ILENG)
C
C SPECIFY THE USER COORDINATE SYSTEM TO BE (200.,200.) AT LOWER
C LEFT CORNER AND (800.,800.) AT UPPER RIGHT CORNER.
C
        CALL SCALE(200.,200.,800.,800.)
C
C OPEN PAGE 1 FOR TOP LEVEL DRAWING
C
        CALL ADDREF(1)
C
C IOP=1 SPECIFIES TO SUBROUTINE CLIP, THAT THE POINT WE ARE
C PASSING IS THE INITIAL X,Y DATA.
C X1 AND Y1 ARE INPUT AS THE NEW BEAM POSITION.X2 AND Y2 ARE NOT USED.
C XPOSB AND YPOSB KEEP TRACK OF THE CURRENT BEAM POSITION.
C AND SHOULD BE THE SAME TWO VARIABLES ON EACH CALL TO CLIP.
C THESE ARE NEITHER SET NOR USED BY THE USER.
C
        IOP=1
C INITIAL POINT
        X1=X(1)
        Y1=Y(1)
        CALL CLIP(IOP,X1,Y1,X2,Y2,200.,200.,800.,800.,XPOSB,YPOSB)
C
C ON RETURN, IF IOP=8, THE SPECIFIED POINT IS OFF SCREEN, DO NOTHING
C IF IOP=7, THE SPECIFIED POINT IS ON SCREEN. THE USER
C SHOULD THEN CALL MOVE USING X1,Y1
C
```

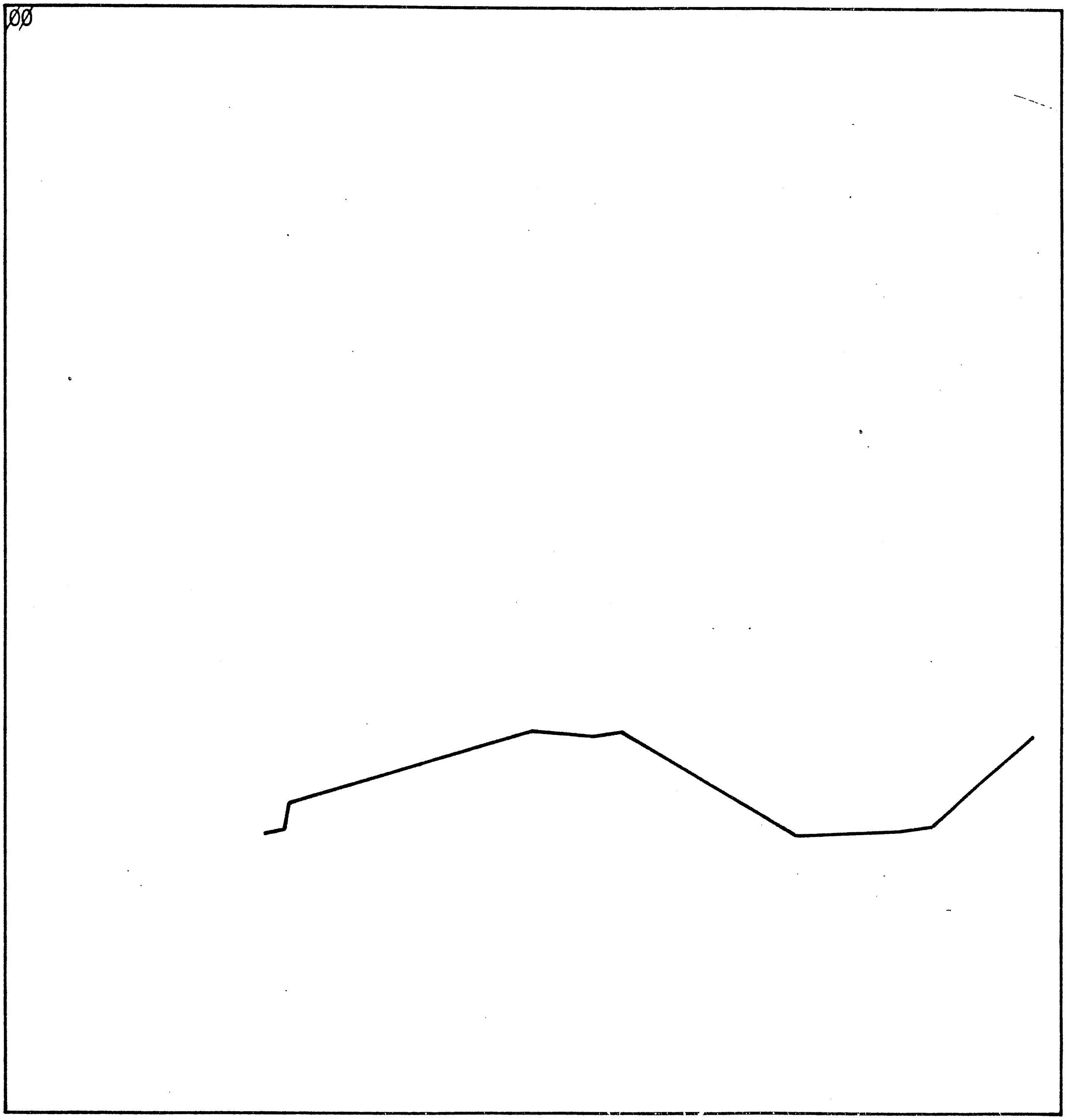
```
      IF(IOP.EQ.7)CALL MOVE(X1,Y1,0)
C
C NOW DO REST OF POINTS
C
      DO 100 I=2,6
C
C IOP=2 SPECIFIES ANY POINT OTHER THAN THE INITIAL POINT
C X2 AND Y2 ARE INPUT AS THE FINAL POSITION OF A VECTOR TO BE
C DRAWN FROM THE CURRENT POSITION.
C NOTE THAT WE MUST USE THE SAME XPOSB AND YPOSB AS IN THE INITAIL
C CALL TO CLIP.
C
      IOP=2
      X2=X(I)
      Y2=Y(I)
      CALL CLIP(IOP,X1,Y1,X2,Y2,200.,200.,800.,800.,XPOSB,YPOSB)
C
C ON RETURN, IF IOP=8 THE ENTIRE VECTOR IS OFF SCREEN
C    DO NOTHING.
C IF IOP=7, THE START POINT OF THE VECTOR IS ON THE SCREEN.
C    CALL DRAW USING X2 AND Y2
C IF IOP=9, THE START POINT WAS OFF SCREEN.
C    FIRST CALL MOVE TO MOVE THE BEAM TO X1,Y1..THAT IS RELOCATE
C    START POINT TO WHERE VECTOR COMES INTO USER COORDINATE RANGE
C    THEN CALL DRAW TO DRAW A LINE TO X2 AND Y2.
C
      IF(IOP.EQ.8)GO TO 100
      IF(IOP.EQ.7)GO TO 50
      IF(IOP.NE.9)GO TO 100
C
C ON RETURN IOP=9 SPECIFYING THE START POINT IS OFF SCREEN
C DO A CALL TO MOVE WITH X1,Y1 AND A CALL TO DRAW WITH X2,Y2
C
      CALL MOVE(X1,Y1,0)
      CALL DRAW(X2,Y2,0)
C NOW GO GET NEXT POINT
C
      GO TO 100
C
C ON RETURN IOP=7 SPECIFYING THE START POINT OF THE VECTOR IS
C ON SCREEN.
C DO A CALL TO DRAW WITH X2,Y2.
C
50    CALL DRAW(X2,Y2,0)
C
C NOW GO GET NEXT POINT
C
100   CONTINUE
C
C WHEN ALL DONE DRAWING,WAIT FOR FUNCTION KEY NUMBER 31 TO BE HIT.
C THEN SHUT DOWN DISPLAY AND EXIT
C
C
C FIRST CALL EVENT TO SEE IF THERE HAS BEEN ANY KEYBOARD INPUT.
C
150   CALL EVENT(II)
C IF II = 4, THEN A KEY WAS HIT, GO RETRIEVE IT, OTHER WISE
C WAIT FOR ANOTHER KEY STRIKE
C
      IF(II.NE.4)GO TO 150
```

```
C CALL GETKEY TO FIND OUT WHAT KEY WAS HIT
C
        CALL GETKEY(KBD,KEY)
C IF IT WAS KEY NUMER 31, THEN EXIT, OTHERWISE GO WAIT FOR
C ANOTHER KEY STRIKE.
C
        IF(KEY.NE.31)GO TO 150
C
C IT WAS KEY 31. SO SHUT DOWN DISPLAY AND EXIT TO THE TTY EMULATOR.
C
        CALL THEEND
        END
```

EXAMPLE 5

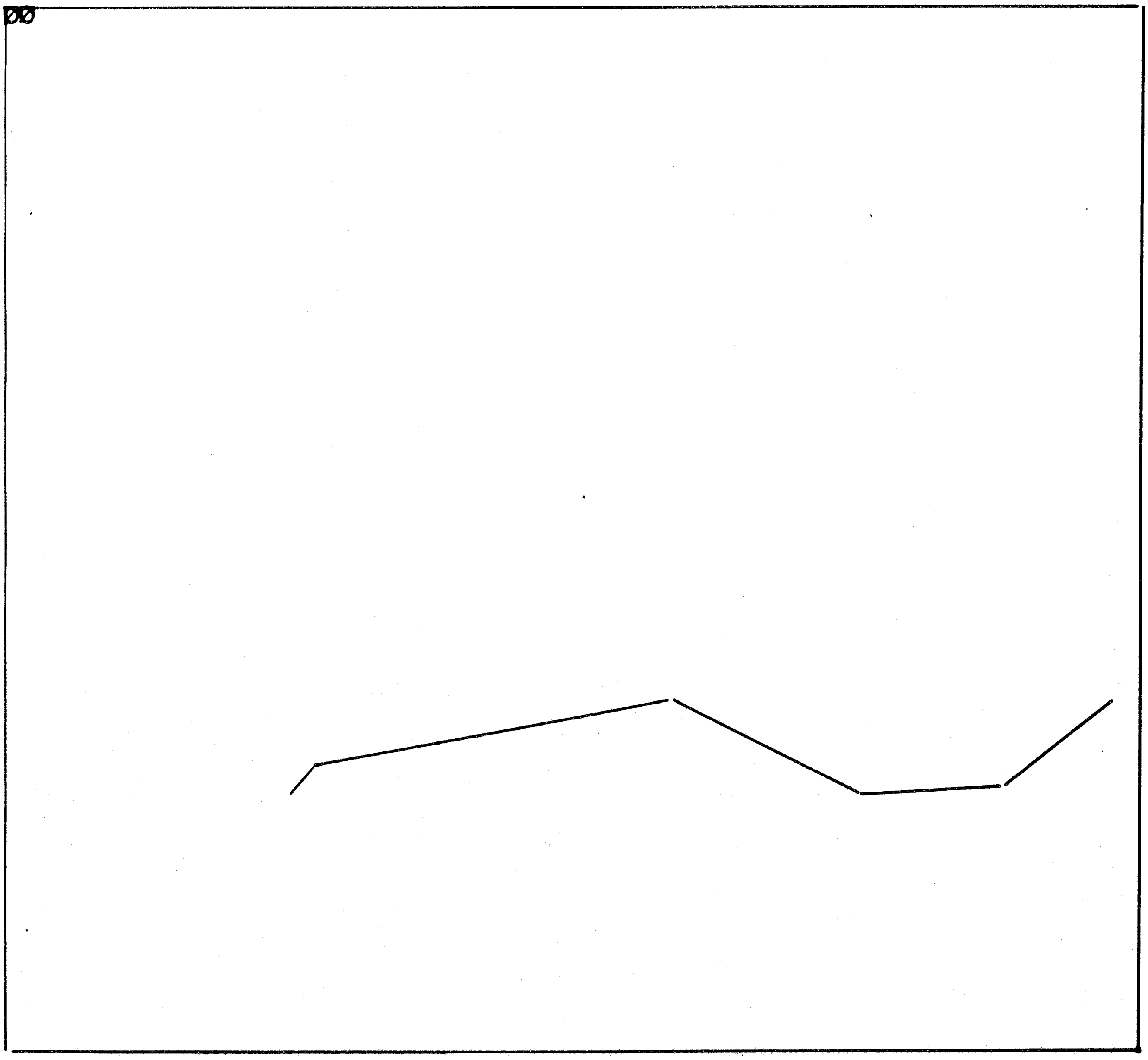
This sample program illustrates the use of FSP subroutine SMOOTH. EXAMPLE PICTURE 1 shows the displayed data before smoothing. EXAMPLE 5 - PICTURE 2 shows the same data displayed after smoothing.

To return to the TTY emulator after the picture is fully displayed, hit the upper right most function key.



EXAMPLE 5 - PICTURE 1

DISPLAYED DATA BEFORE SMOOTHING



EXAMPLE 5 - PICTURE 2

DISPLAYED DATA AFTER SMOOTHING

```
C**************************** FSP  SAMPLE - 5**************************
C
C THIS SAMPLE PROGRAM ILLUSTRATES THE USE OF SUBRCUTINE SMOOTH
C
C THE DATA BASE CONSISTS OF X,Y DATA USED TO FORM A GRAPH,
C EACH DATA POINT WILL BE FED TO SUBROUTINE SMOOTH
C AND AN ALOGRITM WILL DETERMINE WHICH POINTS OF
C THE DATA CAN BE DRAWN KEEPING WITH IN THE LIMITS
C SET BY A DEVIATION FACTOR FED TO SMOOTH,
C PARAMETERS RETURNED INSTRUCT THE USER IN WHAT USER
C ROUTINES TO CALL,THAT IS ,WHETHER TO CALL MOVE,DRAW OR
C @ERFORM NO OPERATION,
C
C
C THE DATA BASE CONSISTS OF 10 X,Y POINTS
C
          DIMENSION X(10),Y(10)
C XSAVE AND YSAVE ARE ARRAYS USED TO BUFFER DATA POINTS
C
          DIMENSION XSAVE(50),YSAVE(50)
C ILENG IS THE LENGTH OF PAGE 1
C
          DATA ILENG/1000/
          DATA X/256.,275.,280.,512.,570.,600.,768.,868.,900.,1000./
          DATA Y/256.,260.,285.,350.,345.,350.,256.,260.,265.,350/
C
C INITIALIZE, USE LOGICAL UNIT NUMBER 5, TRANSMISION
C WILL BE OVER THE SERIAL LINES
C
C SPECIFY ONE PAGE 1000 WORDS IN LENGTH
C SPECIFY USER COORDINATE SYSTEM AS 0.,0. LOWER LEFT
C 1023.,1023, UPPER RIGHT
C
          CALL GSS4(5,0,2)
          CALL LAYOUT(1,ILENG)
          CALL SCALE(0.,0.,1023.,1023.)
C
C INITIALLY SMOOTH IS CALLED WITH IOP=1, THIS INITIALIZES
C THE SUBROUTINE, X AND Y ARE NOT USED
C
C         FOR INITIALIZATION
          IOP = 1
          XX=0.
          YY=0.
C         CALL TO SMOOTH FOR INITIALIZATION
          CALL SMOOTH(ICP,XX,YY,ISAVE,XSAVE,YSAVE,50,20.)
C NOW START SMOOTHING FOR DATA BASE
C
C         DO FOR  10 DATA POINTS
          DO 100 I = 1,10
C X, AND Y DATA ARE FED TO SMOOTH WITH IOP=2.
C THIS INFORMS SMOOTH THAT THIS IS A NEW DATA POINT.
C XX,YY IS THE NEW DATA POINT
          XX = X(I)
          YY = Y(I)
          IOP = 2
          CALL SMOOTH(IOP,XX,YY,ISAVE,XSAVE,YSAVE,50,20.)
C ON RETURN IF IOP=4, THEN THE USER SHOULD TAKE NO ACTION
C IF IOP=5, CALL MOVE TO MOVE THE BEAM TO XX,YY
C
```

```
C IF IOP=6 CALL DRAW TO XX, YY
C
C NOTE! THE XX AND YY RETURNED WILL NOT BE THE SAME AS THE
C ONES JUST PASSED.
      IF(IOP .EQ. 4) GO TO 100
      IF(IOP .EQ. 5) CALL MOVE(XX,YY,0)
      IF(IOP .EQ. 6) CALL DRAW(XX,YY,0)
C CONTINUE FOR ALL DATA POINTS
100   CONTINUE
C DATA IS ALL DONE, NOW CLEAN UP WITH FINAL CALL TO SMOOTH
C THIS CALL IS NECESSARYTO FORCE OUT THE LAST POINT
C
C THIS FINAL CALL TO SMOOTH IS DONE WITH IOP=3 TO
C SPECIFY LINE END.
C XX AND YY ARE NOT USED
C
      IOP = 3
C     CALL SMOOTH AND DRAW LINE
      CALL SMOOTH(IOP,XX,YY,ISAVE,XSAVE,YSAVE,50,20.)
      CALL DRAW(XX,YY,0)
C
C NOW WE ARE DONE WITH PICTURE.
C TO EXIT BACK TO THE TTY EMULATOR , FUNCTION KEY NUMBER 31 MUST
C BE HIT.
C
CFIRST CALL EVENT TO SEE IF THERE HAS BEEN ANY KEYBOARD INPUT.
C
150   CALL EVENT(II)
C IF II = 4, THEN A KEY WAS HIT, GO RETRIEVE IT, OTHER WISE
C WAIT FOR ANOTHER KEY STRIKE
C
      IF(II.NE.4)GO TO 150
C CALL GETKEY TO FIND OUT WHAT KEY WAS HIT
C
      CALL GETKEY(KBD,KEY)
C IF IT WAS KEY NUMER 31, THEN EXIT, OTHERWISE GO WAIT FOR
C ANOTHER KEY STRIKE.
C
      IF(KEY.NE.31)GO TO 150
C
C IT WAS KEY 31, SO SHUT DOWN DISPLAY AND EXIT TO THE TTY EMULATOR.
C
      CALL THEEND
      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

Name:

Company:

Address:

Zip:

Phone:

Report Type

□ Logic error
□ Documentation
□ Suggestion
□ Inquiry

Priority

□ Low
□ Standard
□ High

□ Software □ Firmware □ Hardware

Is the problem reproducible?

□ Yes □ No

CPU:	Host-G7 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
Nashua, New Hampshire 03061

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

FOLD FOLD

SA SANDERS
ASSOCIATES, INC.

Name: ____________________

Company: ____________________

Address: ____________________

____

Telephone: [ ] ____________________

Date: ____________________

Sanders Equipment ____________________

Part Number ____________________

Software/Firmware System ____________________

Version ____________________

Host computer ____________________

Host operating system __________ Version __________

Host-GRAPHIC 7 interface ____________________

My problem is: hardware □ software □

firmware □ manual □

Description of problem (or suggestion for improvement):

Related tech manual number ____________________