REV. 5 hardware

MEGATEK 7000 SERIES
SOFTWARE MANUAL
FORTRAN

Revision Date: 23-DECEMBER-7

Document No. 250-0004-03

MEGATEK
CORPORATION
GRAPHIC SYSTEMS

MEGATEK 7000 SERIES
SOFTWARE MANUAL
FORTRAN

Revision Date: 23-DECEMBER-78
Document No. 250-0004-03

NOTICE

MEGATEK 7000 SERIES
SOFTWARE MANUAL
FORTRAN

CHAPTER 1

INTRODUCTION

MGS

MGS (MEGATEK Graphic Software) is a system of integrated, Fortran-callable subroutines which enable the Fortran programmer to manipulate and display graphical images on the MEGATEK 7000 Graphic Display System. The system is designed to minimize user attention to the screen coordinate system and to permit maximum flexibility in defining user coordinate systems for the various images displayed on the monitor. Bookkeeping for display processor addresses has also been minimized.

THE PICTURE

A "picture" is a display list in the memory of the 7000 and consists of three specific sections: the picture header, the picture components, and the picture trailer.

The picture header contains display list commands which set translation, transformation, special display functions. The first word the picture header is a jump instruction which is used to turn the picture "on" or "off". If the picture is off, the jump instruction destination address is the first word of the next picture. If the picture is on, the jump instruction destination address is the next word of the header (refer to Figure 1). The "P" routines (names starting with a "P") all modify the picture header.

The picture component section consists of display list commands inserted via calls to the "B" or "I" routines (those routines with names starting with the letter B or I). The "B" and "I" routines permit MOVE, DRAW, JUMP, etc., commands to be built in the 7000 memory. Those "B" routines which require X and Y coordinate information expect the values in user units. The "I" routines expect screen coordinates. The picture data base is referenced by the routine which converts user coordinates to screen coordinates before building the instruction in 7000 memory. Pointers are maintained in the picture data base and are used by the "B" and "I" routines to determine where the instruction will be placed. With the use of labels the user may locate specific picture components. Refer to LABELS for additional information.

The picture trailer is two words: one special function word and a JUMP instruction with the destination address being the first word of the next picture. The last picture's trailer instruction is a "STOP".

```
 ______________________________  ____
!                              !    !
!  JUMP (1 WORD)               !    !
!______________________________!    !
!                              !    !
!  DISPLAY FUNCTION (1 WORD)   !    !
!______________________________!    !
!                              !    !
!  TRANSLATION (1 WORD)        !    !
!______________________________!    !
!                              !    !
!                              !    !
!  TRANSFORMATION MATRIX       !    !
!        (6 WORDS)             !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!______________________________!    > HEADER
!                              !    !
!                              !    !
!  CLIPPING (2 WORDS)          !    !
!                              !    !
!______________________________!    !
!                              !    !
!  COLOR (1 WORD)              !    !
!______________________________!____!
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!  PICTURE COMPONENTS          !    > COMPONENTS
!  (ANY NUMBER WORDS)          !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!                              !    !
!______________________________!____!
!                              !    !
!  DISPLAY FUNCTION (1 WORD)   !    !
!______________________________!    > TRAILER
!                              !    !
!  JUMP (1 WORD)               !    !
!______________________________!____!
```

FIGURE 1 - PICTURE FORMAT IN 7ØØØ MEMORY

Once a picture is the current picture, MOVE, DRAW, JUMP, etc., picture components may be placed in the picture with the "B" or "I" subroutines. The components are inserted at the location pointed to by the insert component pointer (ICP) or the last component pointer (LCP) and replace the previous contents of the picture. Before returning, the subroutine may,depending upon the picture mode increment the ICP or LCP to point to the next available 7000 memory location.

It is possible to fill a picture with picture components, and with the use of labels to modify the contents of the picture by writing over old picture components. It is also possible to delete components from the picture. This is done with the BDELV subroutine. BDELV removes the specified components from the picture and compresses the result into minimum 7000 memory. For details on BDELV refer to Chapter 6.

LABELS

Labels are a convenient way to keep track of specific locations in a "picture". To mark a location in 7000 memory for later reference, a call is made to subroutine BPLBL. The current value of the memory pointer (ICP, if in insert or re-write mode; LCP, if in append mode), is placed in the label address vector (LAV) when BPLBL is called. Refer to the following example:

```
C *** EXAMPLE 1-1
C   * USING LABELS
C *** SET APPEND MODE
      CALL PMODE (Ø)
      CALL BPLBL (LBL1)
      CALL BDRWR (X, Y, Ø, Ø, Ø)
          .
          .
          .
C *** GO BACK AND RE-WRITE INSTRUCTION AT LBL1

C   * SET RE-WRITE MODE
      CALL PMODE (2)
      CALL LABGT (LBL1)
      CALL BDRWR (X1, Y1, Ø, Ø, Ø)
          .
          .
          .
```

In the above example a picture component was written at a location marked with a label "LBL1" and was then re-written with a new component at the same location.

The mode selected before the instruction is written is very important. Referring again to the previous example, notice the effect of the various modes as the instruction is written the second time. If the mode was not set to re-write, but instead remained in append mode, the instruction would be written at the end of the picture, not at the location LBL1.

```
C *** EXAMPLE 1-2
C   * USING LABELS
C *** THE FOLLOWING SEQUENCE HAS NO

C   * EFFECT ON THE INSTRUCTION AT

C   * THE LOCATION MARKED BY "LBL1"
      CALL PMODE (0)
      CALL LABGT (LBL1)
      CALL BDRWR (X1, Y1, 0, 0, 0)
```

As mentioned previously, when the mode is insert (1) the instruction will be written at the location marked by "LBL1" but all the instructions below this one will be moved first to make room. The user should be aware of two additional facts: the labels used must be initialized to zero before use, and components pointed to by labels are not lost after insert or delete operations. Example 1-3 illustrates the use of labels during an insert operation. A portion of the display list is "pushed down" as elements are added above it.

```
C *** EXAMPLE 1-3

C   * USING LABELS TO UPDATE ICP
        .
C *** DEFINE A CURSOR
      CALL LABPT  (ICURS)
      CALL IDRWR (-64, Ø, 15, Ø, Ø)
      CALL IMOVR (48, 16)
      CALL IDRWR (16, -16, 15, Ø, Ø)
      CALL IDRWR (-16, -16, 15, Ø, Ø)

C *** NOW MAKE IT MOVE AS LIST IS BUILT

C   * CHANGE TO INSERT MODE
      CALL PMODE (-1)
      DO 1ØØ I = 1,25Ø

C *** UPDATE THE ICP
      CALL LABGT (ICURS)
      CALL BDRWA (X(I), Y(I), IZ(I), Ø, Ø)
1ØØ   CONTINUE

C *** NOW DELETE THE CURSOR
C *** CHANGE TO RE-WRITE MODE
      CALL LABGT (ICURS)

C *** CHANGE TO RE-WRITE MODE
      CALL PMODE (2)
      CALL BWORD (ISTOP)
C *** MSWSTOP AND LSWSTOP WERE PREVIOUSLY DEFINED TO CAUSE
C      DISPLAY PROCESSOR STOP
```

Two things are illustrated by the previous example:

1. In insert mode (-1) the ICP is not incremented after each operation.
2. A label may be used to keep track of a specific display list element.

POINTERS

Labels are very handy when the user wants to "remember" where a picture component is, even after a delete or insert operation which affects the actual address in 7000 memory. When a large number of locations in the picture must be saved for future reference and insert or delete operations will not cause these components to change actual address, pointers may be used.

Pointers are user provided integer variables which contain addresses of specific picture elements. The user can at any time obtain the value of the system pointers (ICP or LCP) with function IGPTR. Subroutine IPPTR is used to set the insert component pointer to a user defined value. Any number of user pointers may be used.

Example 1-4 illustrates the use of pointers during an insert operation. A portion of the display list is "pushed down" as elements are added above it, just as they were in the previous example.

```
C *** EXAMPLE 1-4

C   * USING POINTERS TO UPDATE ICP

C***  DEFINE A CURSOR
      IADR = IGPTR (Ø)
      CALL IDRWR (-64, Ø, 15, Ø, Ø)
      CALL IMOVR (48, 116)
      CALL IDRWR (16, -16, 15, Ø, Ø)
      CALL IDRWR (-16, -16, 15, Ø, Ø)

C *** NOW MAKE IT MOVE AS LIST IS BUILT

C   * CHANGE TO INSERT MODE
      CALL PMODE (1)
      DO 1ØØ I = 1,25Ø


C *** POINT TO CURSOR
      CALL IPPTR (IADR)
      CALL BDRWA (X(I), Y(I), IZ(I), Ø, Ø)
1ØØ   IADR = IADR + 2

C *** NOW DELETE THE CURSOR

C   * CHANGE TO RE-WRITE MODE
      CALL PMODE (2)
      CALL BWORD (MSWSTOP,LSWSTOP)

C *** MSWSTOP, LSWSTOP WERE PREVIOUSLY DEFINED TO CAUSE
C   * DISPLAY PROCESSOR STOP
```

CHAPTER 2

INITIALIZATION ROUTINES

ALLOCATING 7ØØØ MEMORY

MGS maintains a data base in the host processor in a Fortran common block (S7ØØØ). This data base contains the pointers to each picture and other information which controls the location and method for inserting display list commands in 7ØØØ memory.

SUBROUTINE DPSET

DPSET is called to inform MGS of the amount and location of 7ØØØ memory available to the Fortran program. It is called as follows:

```
CALL DPSET (IUNIT, IWRDS)
```

IUNIT is the unit number. IWRDS is the amount of 7ØØØ memory to use, in 1Ø24 word increments, eg: for a 4K system a typical call would look like:

```
CALL DPSET (Ø, 4)
```

If the user is unaware of how much memory is available, DPSET will select the maximum amount available by setting IWRDS = Ø before the call, eg:

```
CALL DPSET (Ø, Ø).
```

INITIALIZING THE PICTURE

All MGS display list commands are inserted in "pictures" as previously discussed. To inform MGS of the existence of each picture, subroutine calls are made which allocate memory space and set picture attributes (normal or text).

SUBROUTINE PINIT

PINIT sets the picture number and allocates picture space in 7000 memory. It is called as follows:

CALL PINIT (IP, IWDS).

PINIT parameters are:

IP - picture number (1-32)

IWDS - number of words of 7000 memory to reserve*

* IWDS may be set = 0 when actual ultimate picture size is not known. This parameter will be set to actual picture size when the next call is made to PINIT.

```
C *** EXAMPLE 2-1

C   * TYPICAL MEMORY ALLOCATION AND PICTURE INITIALIZATION
C     SEQUENCE

C   * DISPLAY LIST IN LOW MEMORY

      CALL DPSET (Ø,Ø)

C *** INITIALIZE A 1ØØØ WORD PICTURE
      CALL PINIT (1, 1ØØØ)

C *** INITIALIZE A PICTURE
C   * RESERVED MEMORY = Ø, SO MGS WILL UPDATE MEMORY USED
C     LATER
      CALL PINIT (2, Ø)
      .
      .
      .
```

SUBROUTINE DSET

DSET is a simple routine which is used to initialize the graphics processor for a single picture/display. It is called as follows:

```
CALL DSET (IUNIT)
```

Where IUNIT is the unit number. This routine can only be used for a single picture. It is equivalent to:

```
CALL DPSET (IUNIT, Ø)
CALL PINIT (1, Ø)
CALL PONOF (1, 1)
CALL DSTRT (IUNIT)
```

CHAPTER 3

VECTOR MOVE AND DRAW ROUTINES

COMMON PARAMETER DEFINITIONS

The following common parameters are used when calling vector routines:

IX - X or horizontal displacement or position in screen coordinates (-2048 through +2047)

IY - Y or vertical displacement or position in screen coordinates (-2048 through +2047)

IZ - intensity (less than or equal to +15)

IB - blink (-1, 0, 1)

ID - dash (-1, 0, 1)

X - X or horizontal displacement or position in user coordinates

Y - Y or vertical displacement or position in user coordinates

In general, negative values for intensity, blink, or dash cause the current value to be used (eg: blink = -1 means current blink). Values of IZ in the range (-15 through -1) cause current intensity with beam on, values of IZ less than -15 cause current intensity with beam off. For point-plot routines, blink can not be set unless the preset (negative IZ) intensity form is used. The absolute point-plot routines do not set a new current intensity with negative values of IZ. Dash is always reset when the preset Z point-plot mode is used. One additional fact: point-plot routines truncate the least significant bit of intensity. Displayed intensity is therefore 2, 4, 6, 8, etc.

ABSOLUTE VECTOR ROUTINES

Routines which cause movement of the display beam to a specified absolute position on the user or screen coordinate systems are referred to as absolute vector routines. If the beam intensity is non-zero when the beam is moved to the specified position, a visible line or "vector" is drawn on the screen. If the beam intensity is zero when the beam is moved, no visible line or "vector" is drawn on the screen. The result of this movement is referred to as a vector move or a hidden vector.

SUBROUTINES BDRWA, IDRWA

Subroutines BDRWA and IDRWA are called as follows:

```
CALL BDRWA (X, Y, IZ, IB, ID)
CALL IDRWA (IX, IY, IZ, IB, ID)
```

These routines build an instruction in the current picture which causes an absolute vector display. The vector will originate at the previous beam position and will terminate at the specified X,Y or IX,IY coordinates.

SUBROUTINES BMOVA, IMOVA

Subroutines BMOVA and IMOVA are called as follows:

```
        CALL BMOVA (X, Y)
        CALL IMOVA (IX, IY)
```

These routines operate in a similar fashion, but cause beam movement with the beam "blanked". Current values of blink, dash, etc., are not affected.

```
C *** EXAMPLE 3-1

C   * USE OF ABSOLUTE VECTOR ROUTINES TO DRAW A BOX
C   * (REFER TO FIG. 2).

C   * THIS EXAMPLE PRODUCES A BOX IN THE
C   * CENTER OF THE DISPLAY.
C   * EACH SIDE IS 5ØØ SCREEN UNITS IN LENGTH.

C   * MOVE TO LOWER LEFT HAND CORNER OF BOX
      CALL IMOVA (-25Ø, -25Ø)

C   * DRAW BOTTOM
      CALL IDRWA (25Ø, -25Ø, 15, Ø, Ø)

C   * DRAW RIGHT SIDE
      CALL IDRWA (25Ø, 25Ø, 15, Ø, Ø)

C   * DRAW TOP
      CALL IDRWA (-25Ø, 25Ø, 15, Ø, Ø)

C   * DRAW LEFT SIDE
      CALL IDRWA (-25Ø, -25Ø, 15, Ø, Ø)
```

```
     -25Ø,25Ø ___________________ 25Ø,25Ø
              !                  !
              !                  !
              !                  !
              !                  !
              !                  !
              !                  !
              !                  !
    -25Ø,-25Ø !__________________! 25Ø,-25Ø
```

FIGURE 2 - BOX DISPLAY PRODUCED BY EXAMPLES 3-1, 3-2

RELATIVE VECTOR ROUTINES

Routines which cause movement of the display beam relative to the current beam position are referred to as relative vector routines. If the beam intensity is non-zero when the beam is moved, a visible line or "vector" is drawn on the screen. If the beam intensity is zero when the beam is moved, no visible line or vector is drawn but the current beam position is updated with a hidden vector.

Relative vector routines differ from absolute vector routines in their use of the provided X,Y data. The X and Y data specify a point relative to the previous beam position. Unless that previous position is known, the resultant beam position after the relative vector is drawn will not be known. If the previous beam position is known, the new beam position will be the addition of the previous beam position coordinates and the specified X and Y values.

SUBROUTINES BDRWR, IDRWR

Subroutines BDRWR and IDRWR are called as follows:

```
CALL BDRWR (X, Y, IZ, IB, ID)
CALL IDRWR (IX, IY, IZ, IB, ID)
```

The box in Figure 2 may be drawn using relative vectors as Example 3-2 below illustrates.

```
C    * EXAMPLE 3-2

C    * USE OF RELATIVE VECTOR ROUTINES TO DRAW A BOX.
     * REFER TO FIGURE 2).

C    * THIS EXAMPLE PRODUCES A BOX IN THE
C    * CENTER OF THE DISPLAY.
C    * EACH SIDE IS 500 SCREEN UNITS IN LENGTH.

C    * MOVE TO LOWER LEFT HAND CORNER OF BOX
     * ASSUME BEAM POSITIONED FIRST AT (0, 0)
C    * AND INTENSITY SET TO A NON-ZERO VALUE.
       CALL IMOVR (-250, -250)

C    * DRAW BOTTOM
       CALL IDRWR (500, 0, 15, 0, 0)

C    * DRAW RIGHT SIDE
       CALL IDRWR (0, 500, 15, 0, 0)

C    * DRAW TOP
       CALL IDRWR (-500, 0, 15, 0, 0)

C    * DRAW LEFT SIDE
       CALL IDRWR (0, -500, 15, 0, 0)

C    * MOVE TO CENTER OF SCREEN
       CALL IMOVR (250, 250)
```

SUBROUTINES BMOVR, IMOVR

Subroutines BMOVR and IMOVR are called as follows:

```
CALL BMOVR (X, Y)
CALL IMOVR (IX, IY)
```

BMOVR and IMOVR act exactly like BDRWR and IDRWR, except that intensity is zero and blink and dash information is not required.

INCREMENTAL VECTOR ROUTINES

An "incremental" vector is produced by specifying an absolute X or Y coordinate and a relative Y or X coordinate. An incremental-X vector is one in which a relative X displacement is specified with an absolute Y position. An incremental-Y vector is one in which a relative Y displacement is specified with an absolute X position.

SUBROUTINES BDXRY, IDXRY

Subroutines BDXRY and IDXRY build an instruction in the current picture which causes an incremental-Y vector display. If the current beam position is CX,CY, the new beam position will be NX,NY where:

NX = X value specified in call
NY = CY + Y value specified in call

Subroutines BDXRY and IDXRY are called as follows:

```
CALL BDXRY (X, Y, IZ, IB, ID)
CALL IDXRY (IX, IY, IZ, IB, ID)
```

One use of the incremental-Y vector is the carriage return/line feed sequence required between lines of text. The specified X value is used as the left hand margin, the Y value is used as line spacing.

SUBROUTINES BMXRY, IMXRY

Subroutines BMXRY and IMXRY are called as follows:

```
CALL BMXRY (X, Y)
CALL IMXRY (IX, IY)
```

An incremental-Y "move" may be performed with subroutines BMXRY and IMXRY. The operation of these routines is exactly like BDXRY and IDXRY except no intensity, blink, or dash information is required, and the resultant vector is "hidden".

SUBROUTINES BDYRX, IDYRX

Subroutines BDYRX and IDYRX are called as follows:

```
CALL BDYRX (X, Y, IZ, IB, ID)
CALL IDYRX (IX, IY, IZ, IB, ID)
```

These routines build an instruction in the current picture which causes an incremental-X vector display. If the current beam position is CX,CY the new beam position will be NX,NY where:

```
NX = CX + X value specified in call
NY = Y value specified in call
```

SUBROUTINES BMYRX, IMYRX

Subroutines BMYRX and IMYRX are called as follows:

```
CALL BMYRX (X, Y)
CALL IMYRX (IX, IY)
```

An incremental-X "move" may be performed with subroutines BMYRX and IMYRX. The operations of these routines are exactly like BDYRX and IDYRX except no intensity, blink, or dash information is required, and the resultant vector is "hidden".

POINT- PLOT ROUTINES

Routines which cause display of a single point are referred to as point-plot routines. These routines combine a "hidden" or zero intensity vector and a vector of zero length but with the beam intensity set to cause a visible mark on the screen.

SUBROUTINES BPYRX, IPYRX

Subroutines BPYRX and IPYRX cause a "hidden" incremental-X vector with a visible end point. The user specifies an absolute Y position and an incremental-X displacement.

Subroutines BPYRX and IPYRX are called as follows:

```
CALL BPYRX (X, Y, IZ, IB)
CALL IPYRX (IX, IY, IZ, IB)
```

If the current beam position is CX,CY then the final beam position where the point will be placed will be NX,NY where:

```
NX = CX + X value specified in call to BPYRX or
     IPYRX
NY = Y value specified in call to BPYRX or
     IPYRX.
```

Example 3-3 below illustrates how a series of points might be plotted using BPYRX.

```
C *** EXAMPLE 3-3

C   * POINT-PLOT DATA USING SUBROUTINE BPYRX

          .
          .
          .
      XINC = 10

      DO 10 I = 1, 100

10    CALL BPYRX (XINC, Y(I), 15, 0)

C *** THE PREVIOUS STATEMENTS CAUSED PLOTTING
C     OF DATA POINTS AT EVEN INCREMENTS OF TEN USER UNITS.
C     ONE HUNDRED POINTS WERE PLOTTED.
          .
          .
          .
```

SUBROUTINES BPXRY, IPXRY

Subroutines BPXRY and IPXRY cause a "hidden" incremental-Y vector with a visible end point. The user specifies an absolute X position and an incremental-Y displacement. These routines are called as follows:

```
CALL BPXRY (X, Y, IZ, IB)
CALL IPXRY (IX, IY, IZ, IB)
```

If the current beam position is CX,CY, then the final beam position where the point will be placed will be NX,NY where

NX = X value specifed in call
NY = CY + Y value specified in call

SUBROUTINES BPNTA, IPNTA

Subroutines BPNTA and IPNTA are used when absolute X,Y positioned points are to be displayed. These routines are called as follows:

```
CALL BPNTA (X, Y, IZ, IB)
CALL IPNTA (IX, IY, IZ, IB)
```

SUBROUTINES BPNTR, IPNTR

Subroutines BPNTR and IPNTR are used when points are to be displayed with positioning based on X,Y relative displacement. These routines are called as follows:

```
CALL BPNTR (X, Y, IZ, IB)
CALL IPNTR (IX, IY, IZ, IB)
```

If the current beam position is CX,CY the point position will be NX,NY where:

NX = CX + X value specified in routine call
NY = CY + Y value specified in routine call.

VECTOR STRINGS

Vector strings are a way to describe a character, figure, or point plot series, and are extremely memory efficient because common data for each vector in the string is stored in memory only once.

SUBROUTINES BRSTG, IRSTG

Subroutines BRSTG and IRSTG output a short relative vector string. The vectors produced by these routines are called short because only seven bits of X or Y information are used for each vector in the string versus the twelve bits used in other vector formats. Three additional bits are used for a multiplying factor (1-8) on each X or Y displacement value.

These routines are called as follows:

```
CALL BRSTG (XA, YA, IZ, N, IBLAN, IR, ISIZ)
CALL IRSTG (IXA, IYA, IZ, N, IBLAN, IR, ISIZ)
```

Where:

XA, IXA are arrays of X-displacement values.
YA, IYA are arrays of Y-displacement values.
IZ is intensity (-15 through +15).
N is dimension of the the X, Y, and blanking arrays.
IBLAN is array of blanking data (Ø blank, 1 display)
IR is return-from-subroutine indicator (Ø = no return, 1 = return from subroutine).
ISIZ is size multiplying factor (Ø-7) Actual result is (1-8).

The X and Y values provided to the routine are truncated to seven bits and then the current multiplying factor is applied. A maximum length of 512 screen units (64 x 8) is thus possible for each vector in the string.

SUBROUTINES BIXTG, IIXTG

Subroutines BIXTG and IIXTG output an incremental-X series string. A single X-increment value is provided and an array of Y data. Each vector in the string is drawn from the current point to the new point using the incremental (relative) X value and an absolute-Y value taken from the Y-array.

Points may be plotted in lieu of vectors by setting the mode flag (M).

These routines are called as follows:

```
CALL BIXTG (X, YA, IZ, N, IBLAN, IR, M)
CALL IIXTG (IX, IYA, IZ, N, IBLAN, ,IR, M)
```

Where:

X, IX are the incremental-X value used for each vector or point in the string.
YA, IYA are arrays of absolute-Y positioning data for each vector or point.
IZ is intensity (-15 through +15).
N is dimension of the YA, IYA, and blanking arrays.
IBLAN is array of blanking data (Ø blank, 1 display)
IR is return-from-subroutine indicator (Ø = no return, 1 = return from subroutine).
M is mode; Ø = vector, 1 = point-plot. flags hyphenate

CHAPTER 4

ROTATION, TRANSLATION, SCALING, AND CLIPPING

HARDWARE REQUIREMENTS

MEGATEK 7ØØØ Series Graphic Systems are provided with hardware translation circuitry as standard equipment. This permits the user to move a complete "picture" or series of vectors on the display screen with a minimum amount of I/O traffic to the host CPU.

When provided with additional circuitry, the 7ØØØ Graphic System is also capable of hardware rotation, scaling, and clipping. With this additional capability, it is possible to perform complete two-dimensional transformations of a user "picture" with negligible host CPU action. The host provides the translation, rotation, scaling, and clipping parameters; the 7ØØØ Display Processor then applies these parameters to each affected vector before it is displayed. The parameters are all set in display list instructions, hence it is possible to apply different parameters to different "pictures" with no host CPU action.

SUBROUTINES BXLT, IXLT

Subroutines BXLT and IXLT are used to set hardware translation parameters for vectors following the call. These routines are called as follows:

```
CALL BXLT (XØ,YØ)
CALL IXLT (IXØ,IYØ)
```

Where:

XO is the new X-origin in user coordinates.
YØ is the new Y-origin in user coordinates.
IXØ is the new X-origin in screen units (-2Ø48 to +2Ø47).
IYØ is the new Y-origin in screen units (-2Ø48 to +2Ø47).

The effect of the translation call is to change the positioning of the origin (Ø, Ø) to a location other than the center of the screen. If an origin in the lower left hand corner of the screen is desired, this call might be used:

```
CALL IXLT (-2Ø48, -2Ø48)
```

SUBROUTINE DTRANS

The DTRANS routine allows the programmer to use the Fortran Graphics Display System without being burdened with the problems of scaling data to conform with the screen coordinate system. Parameters allow the user to specify:

1. A rotation angle (about the user origin).

2. The window boundaries (in user coordinates).

3. The screen boundaries within which the data will be displayed (clipping all data outside of the boundaries).

The transformation process proceeds logically as follows:

1. The picture is rotated through the indicated angle (about the user origin).

2. The picture is scaled and translated to match the user data with the corners of the screen window.

3. The picture is clipped to eliminate any vectors outside of the window.

The display buffer filled by this call should not be the current refresh buffer for the display (no allowances have been made for stopping the display while vectors are being added). DTRANS is called as follows:

```
CALL DTRANS (X, Y, Z, C, N, IP, IFL)
```

Where:

X = An array containing X values (in user coordinates).
Y = An array containing Y values (in user coordinates).
Z = An array containing intensity codes.
C = A nine-element control matrix.
IP = Picture number
N = Number of points (in X, Y, Z arrays).
IFL = Execution control word

The transformation control matrix should contain the following information:

> C(1), C(2) = Coordinates of the lower left corner of the user window (in user units).
>
> C(3), C(4) = Coordinates of the upper right corner of the user window (in user units).
>
> C(5), C(6) = Coordinates of the lower left corner of the screen boundaries (in screen units).
>
> C(7), C(8) = Coordinates of the upper right corner of the screen boundaries (in screen units).
>
> C(9) = Angle (in radians) to rotate data about user origin prior to scaling, translation, and clipping.

The execution control word has the following effect:

> IFL = Ø = Clipping calculations performed
>
> IFL = 1 = Clipping calculations eliminated

NOTE: DTRANS does not utilize any hardware rotation, translation, scaling, or clipping cabability of the 7ØØØ.

SUBROUTINE PCLIP

The PCLIP routine allows the programmer to change the picture clip boundaries when the optional hardware clip element is installed in the graphics system. PCLIP is called as follows:

```
CALL PCLIP (IP, LX, LY, HX, HY
```

WHERE:

IP = Picture number
LX = X-coordinate of lower left hand corner of clip window
LY = Y-coordinate of lower left hand corner of clip window
HX = X-coordinate of upper right hand corner of clip window
HY = Y-coordinate of upper right hand corner of clip window

NOTE: All coordinates are screen coordinates.

SUBROUTINE PTRAN

The PTRAN routine allows the programmer to change the transformation matrix elements in the picture header. When the graphics system is equipped with the optional 2-d transformation element (HRST), pictures may be quickly scaled, rotated, and translated with the PTRAN call.

PTRAN is called as follows:

```
CALL PTRAN (IP, SCLX SCLX, STRX, STRY, ROT,
RTRX, RTRY, TRX, TRY)
```

WHERE:

IP = Picture number
SCLX = X scale factor
SCLY = Y scale factor
STRX = Center of scale X
STRY = Center of scale Y
ROT = Rotation angle (radians)
RTRX = Center of rotate -X
RTRY = Center of rotate -Y
TRX = Translation -X
TRY = Translation -Y

NOTE: Refer to Display Command Format manual for details of of the transformation process.

CHAPTER 5

TEXT AND CHARACTER STRING MANIPULATION

STRING HANDLING ROUTINES

SUBROUTINE BSTNG

Subroutine BSTNG is used to place alphanumeric character strings in a picture with normal attribute. The string is located at the current beam position, and depending upon specified character rotation, may extend to the right, left, top, or bottom of the "picture".

Subroutine BSTNG is called as follows:

```
CALL BSTNG (IZ, ISIZ, IROT, ISTNG, NUMCHR)
```

Where:

IZ is intensity (-15 through +15)
ISIZ is character size (Ø through 7)
IROT is character rotation (Ø through 3)
ISTNG is string of alphanumeric characters
NUMCHR is number of characters in string

Character rotation is in 9Ø degree increments from Ø through 27Ø degrees counterclockwise. As in subroutine BTEXT, NUMCHAR may be set = -1, in which case the end of the string is detected at the occurence of a null character.

SUBROUTINE GNUM

Subroutine GNUM is provided in order to permit conversion of a floating point number to a string of alphanumeric characters for display by subroutines BTEXT and BSTNG. GNUM is called as follows:

```
CALL GNUM (FNUM, IB, NDP, IL, ICRAY, IFMT)
```

Where:

FNUM is floating point number for display.
IB is radix of FNUM.
NDP is number of digits past decimal point.
IL specifies leading zeroes or blanks in output if necessary to achieve desired IFMT.
(Ø = leading blanks, 1 = leading zeroes).
ICRAY is array to receive characters.
IFMT is number of chars in output string

If the user desires to output a number which does not include any digits past the decimal, set NDP = Ø. If the decimal point itself is to be eliminated, set NDP = -1. Refer to Example 5-1 for typical use of GNUM.

```
C *** EXAMPLE 5-1

C   * USING GNUM TO DISPLAY FLOATING POINT NUMBERS
      .
      .
      .
C *** FIRST POSITION BEAM WHERE NUMBER TO BE DISPLAYED
C     (CENTER OF SCREEN)
      CALL IMOVA (Ø, Ø)

C *** CONVERT NUMBER TO A STRING (X IS THE VARIABLE
C     CONTAINING A FLOATING POINT NUMBER)
      CALL GNUM (X, 1Ø, 2, Ø, IBUF, 6)

C *** ASSUMING X = 99.99999, THE FOLLOWING IS EXPECTED:

C   * IBUF CONTAINS A STRING:  " 99.99"

C *** NOW DISPLAY THE NUMBER
      CALL BSTNG (15, 1, Ø, IBUF, 6)
      .
      .
      .
```

CHAPTER 6

FORMAT CONTROL AND LIST MANIPULATION

SUBROUTINE PMODE

As described in Chapter 1, display list "pictures" are built in three different modes: append mode, insert mode, and re-write mode. The appropriate mode is selected using a subroutine which modifies the MGS data base. The default mode is append.

Subroutine PMODE is used to change mode, and is called as follows:

```
CALL PMODE (IMODE)
```

Where:

IMODE is Ø for append mode, 1 for insert mode, 2 for re-write mode, -1 for insert mode without ICP update, -2 for re-write mode without ICP update. For information on effects of the various modes, refer to Chapter 1.

SUBROUTINE BDELV

Subroutine BDELV is used to delete vectors from the current picture.. When the operation has been completed, the resulting display list picture is compressed into minimum 7ØØØ memory. All MGS pointers together with user labels are updated by the call. All labels point to the same picture element after the operation, even if the delete operation caused movement of the picture element associated with a particular label. The calling sequence for BDELV is as follows:

> CALL BDELV (IWCNT) Where:
> IWCNT number of words to be deleted.
> from the current pointer.
>
> In append mode, vectors are deleted
> from before the LCP.

Example 6-1 below illustrates the use of BDELV.

```
C *** EXAMPLE 6-1

C   * USING SUBROUTINE BDELV TO DELETE VECTORS

C   * GET POINTER TO VECTORS TO BE DELETED
      CALL BGLBL (LABEL)

C   * NOW GO TO INSERT MODE SO THE ICP CAN BE USED
      CALL PMODE (1)

C   * DELETE 1Ø VECTORS
      CALL BDELV (1Ø)

C *** AT THIS POINT THE LCP IS TEN LESS THAN IT WAS
C   * BEFORE BDELV, THE ICP IS UNCHANGED,
C   * ALL LABELS BELOW "LABEL" HAVE BEEN
C   * UPDATED, AND THOSE ABOVE "LABEL" ARE UNCHANGED
```

SUBROUTINE LABPT

Subroutine LABPT is used to save a display address for later reference, and is called as follows:

```
CALL LABPT (ILABL)
```

Where:

ILABL is an integer variable in the user program (not an integer constant). Refer to Chapter 1 for typical uses of BPLBL.

SUBROUTINE LABGT

Subroutine LABGT is used to retrieve display list information previously saved with a call to subroutine LABGT. The information retrieved is placed in the insert component pointer (ICP). Refer to Chapter 1 for examples where LABGT is used

SUBROUTINE POPEN

Subroutine POPEN is used to change the "current" picture to a specified value. POPEN is called as follows:

```
CALL POPEN (ICPCT)
```

Where:

ICPCT is an integer value (1-32).

To make Picture 2 the current picture, the following call could be made:

```
CALL POPEN (2)
```

FUNCTION IGPTR

This function gets the actual value of the last component pointer (LCP) or the insert component pointer (ICP) of the current picture. IGPTR is called as follows:

```
IVALU = IGPTR (MODE)
```

Where:

IVALU will receive the pointer value.
MODE is the control (Ø: get value of LCP; not Ø: get value of ICP).

SUBROUTINE IPPTR

This routine puts a value in the insert component pointer of the current picture. IPPTR is called as follows:

```
CALL IPPTR (IVALU)
```

Where:

IVALU contains data for the ICP of the current picture.

SUBROUTINE BWORD

BWORD is used to insert information in the current picture without any formatting. BWORD is called as follows:

```
CALL BWORD (MSW,LSW)
```

Where:

MSW =is the Most Significant Word
LSW =is the Least Significant Word

CHAPTER 7

JUMP AND JUMP-SUBROUTINE CALLS

JUMP ROUTINES

Several routines are provided which build display list jump instructions in the current picture. These routines are of two types:

1. Jumps to absolute addresses.
2. Vectored jumps.

Using labels and pointers, it is possible to provide addresses to the jump routines which may then be used to affect the order in which the display list is processed.

SUBROUTINE BJMLB

BJMLB builds a jump-to-label instruction in the current picture. The address used is fetched from the LAV based upon the contents of the specified label. BJMLB is called as follows:

```
CALL BJMLB (LABEL,IZ)
```

Where:

LABEL is a user label variable previously set in a CALL BPLBL.

1 Z - Intensity control (< =15. <Ø is preset). Ø.

SUBROUTINE BJMAD

BJMAD builds a jump instruction in the current picture using either absolute or relative address formats. BJMAD is called as follows:

```
CALL BJMAD (IVAW, IZ)
```

Where:

IVALU is the address value.
1 Z is intensity control (<=15. <0 is preset)

SUBROUTINE BVJLB

Subroutine BVJLB adds a vector-jump-through-label instruction to the current picture. BVJLB is called as follows:

```
CALL BVJLB (LABEL, IZ)
```

Where:

LABEL is a user label previously initialized with a CALL BPLBL.
1 Z is intensity control (<= 15. <0 is preset)

The vector-jump-through-label is illustrated in Example 7-1 below.

```
C *** EXAMPLE 7-1

C   * USE OF A VECTOR JUMP

C   * RESERVE A LOCATION FOR THE VECTOR WORD
      CALL BPLBL (IVWORD)

C *** NOW PLACE A CONSTANT IN THE VECTOR WORD
      MSW (1) = Ø
      LSW (2) = 1
      CALL BWORD (MSW,LSW)

C *** BUILD THE VECTOR JUMP INSTRUCTION AND
C     POSSIBLE DESTINATION JUMPS.
      CALL BVJLB IVWORD,O)
      CALL BJMLB (LABLA,Ø)
      CALL BJMLB (LABLB,Ø)
      CALL BJMLB (LABLC,Ø)

C *** INSTRUCTION SEQUENCE IS COMPLETE.
C     WHEN EXECUTED,THE DESTINATION WILL BE THE LOCATION
C     POINTED TO BY LABEL "LABLB".


C *** TO CHANGE THE DESTINATION OF THE VECTOR JUMP
C     TO LABEL "LABLC",
C     THE FOLLOWING SEQUENCE MAY BE EXECUTED:
      CALL BGLBL (IVWORD)
      CALL PMODE (2)
      LSW (2) = 2
      CALL BWORD (MSW,LSW))


C *** NOTICE THAT ALL VECTOR JUMPS
C     WHICH JUMP THROUGH LABEL "IVWORD"
      WILL BE AFFECTED BY THE NEW CONSTANT.
```

JUMP-SUBROUTINE SEQUENCES

Just as the jump routines (BJMAD, BJMLB, etc) build jump sequences in the current picture, there are routines available which build jump-subroutine sequences in the current picture.

A jump-subroutine sequence differs from a jump sequence in that a return address (the location of the instruction in the display list plus one) is pushed on the display processor stack when the instruction is encountered by the display processor. The user may return to this location by setting the return bit in his display list "subroutine" (refer to subroutine BRETN, below).

Display list "subroutines" are one way of conserving display processor memory. Software symbols, etc., which are displayed more than once may be defined in subroutine sequences, then "called" as many times as required by the main-line display list sequence.

Caution should be exercised in locating display list "subroutines". These instruction sequences should not be executed in the main line display list, as they can only be executed properly by a "call" from the main display list. See Example 7-2.

SUBROUTINE BJSLB

BJSLB builds a jump-subroutine-to-label instruction in the current picture. The address used is fetched from the LAV based upon the contents of the specified label. BJSLB is called as follows:

```
CALL BJSLB (LABEL, IZ)
```

Where:

LABEL is a user label previously set in a CALL BPLBL
1 Z is intensity control (<=15. <Ø is preset) = Ø.

SUBROUTINE BJSAD

BJSAD builds a jump-subroutine-to-address instruction in the current picture. BJSAD is called as follows:

```
CALL BJSAD (IVALU, IZ)
```

Where:

IVALU is the address value
1 Z is intensity control (<= 15. <Ø is preset) Ø.

SUBROUTINE BRETN

BRETN is used to build the return-from-subroutine sequence. BRETN is called as follows:

```
CALL BRETN
```

```
C *** EXAMPLE 7-2

C   * USING DISPLAY LIST SUBROUTINES

C   * TURN OFF PICTURE 1
C     (THIS IS WHERE THE SUBROUTINES WILL BE PLACED).
      CALL PONOF (1, Ø)

C *** BUILD A SUBROUTINE SEQUENCE (AN "X")
      CALL POPEN (1)
      CALL BPLBL (LX)
      CALL BMOVR (-25., -25.)
      CALL BDRWR (5Ø., 5Ø., -15, Ø, Ø)
      CALL BMOVR (-5Ø., Ø.)
      CALL BDRWR (5Ø., -5Ø., -15, Ø, Ø)
      CALL BRETN
C *** NOW CALL THE SUBROUTINE TO DISPLAY THE "X"

      CALL POPEN (2)
      CALL BMOVA (-5ØØ., Ø.)
      CALL BJSLB (LX,-1)
      CALL BMOVA (Ø., Ø.)
      CALL BJSLB (LX,-1)
      CALL BMOVA (5ØØ., Ø.)
      CALL BJSLB (LX,-1)

C *** THERE ARE NOW THREE SYMBOLS ("X") ON THE SCREEN
```

CHAPTER 8

PICTURE CONTROL ROUTINES

As indicated in Chapter 1, there is a header on each picture which controls the way in which that picture is displayed. The picture control routines (those routines beginning with the letter "P") all modify the picture header.

SUBROUTINE PONOF

PONOF is used to control whether or not the picture is displayed. A picture is "on" when it is being displayed and is "off" when it is not being displayed. A picture is turned "off" by changing the destination of the jump instruction in the picture header. PONOF is called as follows:

```
CALL PONOF (IP, ICNTL)
```

WHERE:

IP is the picture number (1-32).
ICNTL is on/off control (Ø= off, 1= on).

To turn off picture 1, the following call is made:

```
CALL PONOF (1, Ø)
```

To turn it back on, the following call is made:

```
CALL PONOF (1, 1)
```

SUBROUTINE PXLT

PXLT is used to control the translation of a picture. PXLT operates exactly like BXLT, described in Chapter 4, except that the header translation word of the picture is modified. PXLT is called as follows:

```
CALL PXLT (IP, XO, YO)
```

WHERE:

IP is picture number (1-32)
XO is X origin in user units
YO is Y origin in user units

To set the origin in picture 2 to (-1ØØ., -1ØØ.) the following call is made:

```
CALL PXLT (2, -1ØØ., -1ØØ)
```

For a detailed discussion of tanslation, refer to BXLT in To set the origin in picture 2 to (-1ØØ., -1ØØ.) the following call is made:

SUBROUTINE PSCAL

PSCAL allows the user to set the ratio of screen units/ user units in both the X and Y directions. The routine is called as follows:

```
CALL PSCAL (IP, XRAT, YRAT)
```

WHERE:

IP = Picture number
XRAT = Ratio of screen units/user units in the X direction.
YRAT = Ratio of screen units/user units in the Y direction.

FOR EXAMPLE - If the user wished to have the screen divided into 1ØØ user units along the Y axis and 2ØØ user units along the X axis for picture 5-

The program should call PSCAL (5, 2Ø.48,4Ø.96)

SUBROUTINE PMAP

PMAP allows the user to translate the origin to any location on the screen and to set the ratio of screen units/user units in both the X and Y directions.

The routine is called as follows:

```
CALL PMAP (IP, XL, yl, USERX, USERY)
```

WHERE:

IP = Picture number
XL = X coordinate of the lower left corner of screen
YL = Y coordinate of the lower left corner of screen.
USERX= Number of user units on the X axis
USERY= Number of user units on the Y axis

FOR EXAMPLE - If the user wished to have the origin in the lower left corner of the screen, 1ØØ user units along the Y axis and 2ØØ user units along the X axis. The program should call all PMAP (1, Ø.,Ø., 2ØØ, 1ØØ).

SUBROUTINE PWORD

PWORD re-writes the special function word in the referenced picture. PWORD is called as follows:

```
CALL PWORD (IP, MSW, LSW)
```

Where

IP is the picture number
MSW is the Most Significant Word
LSW is the Least Significant Word

Each bit set in the conrol constants will set a corresponding bit in the special function word in the picture header. For a detailed discussion on the form of MSW and LSW refer to the Display Command Format documentation.

CHAPTER 9

DISPLAY PROCESSOR FUNCTION AND CONTROL

This chapter describes the display processor function and control routines. These routines control display refresh, which monitor (if any) is inhibited from display, etc. There are also routines which start and stop the display processor.

SUBROUTINE DSTRT

DSTRT is used to initialize and start the 7ØØØ display processor. It is called as follows:

```
CALL DSTRT (IUNIT)
```

Where IUNIT = unit # of screen (Ø-3)

DSTRT should not be called before a call to subroutine BIN7, which establishes a "skeleton" display list, and BINIT which establishes a "skeleton" picture.

SUBROUTINE DHALT

DHALT is used to stop the 7ØØØ display processor. It is called as follows:

```
CALL DHALT (IUNIT)
Where IUNIT = unit  of screen (Ø-3)
```

SUBROUTINE BSETZ

Display processor current intensity may be set without the use of one of the vector routines. This is accomplished with a call to subroutine BSETZ as follows:

```
CALL BSETZ (IZ)
```

Where:

IZ is the new display intensity (Ø-15).

SUBROUTINE DREFR

The display may be set to run in one of two modes: continuous or line-synchronized. When refresh mode is continuous, the display processor will immediately jump to the beginning of the display list when it encounters the end of the list. When the refresh mode is line-synchronized, the display processor halts when the end of the display list is encountered, and it re-starts at the beginning of the list when the next line sync pulse is received. Continuous refresh mode is used when maximum display intensity is desired and the number of vectors displayed remains relatively constant. A wide change in the number of vectors in the display list can cause variations in display intensity. For this reason, most display lists will be processsed in line-synchronized mode. Subroutine DREFR is used to change refresh mode, and is called as follows:

```
CALL DREFR (IUNIT, IREF)
```

Where:

IUNIT = unit # of screen (Ø-3)
IREF - Ø causes continuous refresh
IREF = 1 causes line-synchronized refresh

NOTE: DREFR also starts the display processor if not already running.

CHAPTER 10

GRAPHICS PERIPHERAL ROUTINES

JOYSTICK ROUTINES

Four routines are provided for controlling joysticks and retrieving data:

1. JOYON - Starts the joystick digitizing hardware.
2. JOYRD - Gets joystick X,Y coordinates and button status.
3. JOYOF - Turns off the joystick digitizing hardware.
4. JOYLM - Establish tracking limits for the joystick cursor.

SUBROUTINE JOYON

JOYON starts the joystick digitizing hardware. It is called as follows:

```
JOYON (IUNIT, ICRSR, ICTL, IER)
```

Where:

```
ICRSR = 0 for default cursor;  not = 0 for user
        cursor.  ICRSR if not = 0 should be a
        user label previously set with a call
        to subroutine BPLBL.
IUNIT = unit number (0 through 3).
ICTL = 0 - local tracking.
       1 - local track, interrupt button down.
       2 - local track, interrupt button up.
       3 - local track, interrupt button up and
           down.
       4 - local track, 100 Hz interrupt.
       5 - local track, 100 Hz interrupt button
           down.
IER = 1 for success;  -1 for no such device.
```

SUBROUTINE JOYRD

JOYRD retrieves the joystick data, and is called as follows:

```
JOYRD(IUNIT, IX IY, IPEN, IRDX, IRDY)
```

Where:

```
IX is X-cursor position in screen units.
IY is Y-cursor position in screen units.
IPEN is pen status;  0 = pen up, 1 = pen down.
IRDX is raw X data (-3 through +3).
IRDY is raw Y data (-3 through +3).
IUNIT is unit number (0 through 3).
```

SUBROUTINE JOYOF

JOYOF is called to turn off the joystick digitizing hardware. It is called as follows:

```
JOYOF (IUNIT)
```

Where:

```
IUNIT is unit number (0 through 3).
```

SUBROUTINE JOYLM

JOYLM is called to establish tracking limits for the joystick cursor. It is called as follows:

JOYLM (IUNIT, IYL, IXR, IYU, IXL)

~~IXL, IXU~~

IXL, IYL, IXU, IYU

Where:

IXL is left screen limit for cursor movement.
IYL is lower screen limit for cursor movement.
IXR is right screen limit for cursor movement.
IYU is upper screen limit for cursor movement.
IUNIT is unit number (0 through 3).

DATA TABLET ROUTINES

Four routines are provided for controlling data tablets and retrieving data:

1. TABON - Starts the data tablet digitizing hardware.
2. TABRD - Gets data tablet X,Y coordinates and pen status.
3. TABOF - turns off the tablet digitizing hardware.
4. TABLM - Establishes tracking limits for the joystick cursor.

SUBROUTINE TABON

TABON is called to start the data tablet digitizing hardware. It is called as follows:

CALL TABON (IUNIT, ICRSR, ICTL, ITRES, IER)

Where:

ICRSR = 0 for default cursor, not = 0 for user cursor (ICRSR = label previously defined by a call to subroutine BPLBL).
IUNIT = unit number (0 through 3).
ICTL = 0 - local tracking.
1 - local track, interrupt pen down.
2 - local track, interrupt pen up.
3 - local track, interrupt pen up and down.
4 - local track, 100 Hz interrupt.
5 - local track, 100 Hz if pen down.
ITRES = Number of bits of tablet data mapped to screen:
1 = 11 bits (2048 units)
→ 2 = 12 bits (4096 units)
3 = 13 bits (8192 units)
4 = 14 bits (16,384 units)

size of tablet

NOTE:
This is a function of tablet size. There are 200 tablet units per tablet inch. An 11-inch tablet has available 11 x 200 = 2,200 tablet units and is considered to have 11 bits of resolution for mapping to screen coordinates.

IER = 0 for success, -1 for error (no such device).

presently 1 for succes, another megabug

SUBROUTINE TABRD

TABRD is used to retrieve the data tablet X,Y coordinate data and pen status. TABRD is called as follows:

```
CALL TABRD (IUNIT, IX, IY,IPEN,IRDX,IRDY,IRDP)
```

Where:

IX is X-cursor position screen units.
IY is Y-cursor position screen units.
IPEN is pen status; 0 = pen up, 1 = pen down.
IRDX is tablet raw x-data.
IRDY is tablet raw y-data.
IRDP is tablet raw pen status data (5 bits).
LSB = dual tablet flag
Next LSB = pen flag 3
Next LSB = pen flag 2
Next LSB = pen flag 1
Next LSB = primary pen status
IUNIT is unit number (0 through 3).

SUBROUTINE TABLM

TABLM is called to establish the data tablet cursor tracking limits. TABLM is called as follows:

```
CALL TABLM (IUNIT IYL, IXR, IYU, IXL)
```

IXL, IYL, IXU, IYU)

Where:

IXL is left screen limit for cursor movement.
IYL is lower screen limit for cursor movement.
IXR is right screen limit for cursor movement.
IYU is upper screen limit for cursor movement.
IUNIT is unit number (0 through 3).

SUBROUTINE TABOF

TABOF is called to stop the data tablet digitizing hardware. TABOF is called as follows:

```
CALL TABOF (IUNIT)
```

Where:

IUNIT is unit number (0 through 3).

KEYBOARD HANDLING ROUTINES

Input from the keyboard is disabled until a call is made to KEYON. There are two routines which enable the user to retrieve keyboard characters:

1. KCHAR retrieves a single character from the keyboard; task waits until a key is struck).

2. KCHNW retrieves a single character from the keyboard (no suspension of task).

3. KLINE retireves a string of characters, terminated by a null, carriage return, or form feed. (Task waits until a line is returned).

Displaying the characters on the screen (echoing) is left to the user. Input from the keyboard can be disabled and the input buffer cleared by a call to KEYOF.

SUBROUTINE KEYON

Subroutine KEYON initializes keyboard input routine. The keyboard is enabled and the interrupt service routine is initialized. This call must be made before any keystrokes from the keyboard can be accepted. KEYON is called as follows:

```
CALL KEYON (IUNIT, IER)
```

Where:

IUNIT is unit number (0 through 7).
IER is error code. 1 = success; -1 = no

such device in system.

SUBROUTINE KEYOF

Subroutine KEYOF turns the keyboard off. The interrupt service routine is disabled and the input buffer is cleared. Any further calls to KCHAR or KLINE will cause a graphics error (see Appendix A). Subroutine KEYOF is called as follows:

```
CALL KEYOF (IUNIT)
```

Where:

IUNIT is unit number (Ø through 3).

SUBROUTINE KCHAR

Subroutine KCHAR gets a single character from the keyboard. The character is returned in the low order byte of the single integer argument. The user task is suspended until the character is available. KCHAR is called as follows:

```
CALL KCHAR (IUNIT, ICH)
```

Where:

IUNIT is unit number (Ø through 3).
ICH is a single integer variable
into which the character is to be stored.

SUBROUTINE KCHNW

Subroutine KCHNW, like KCHAR, gets a single character from the keyboard. Unlike KCHAR, KCHNW does not suspend the user task while waiting for the character.

KCHNW is called as follows:

```
SUBROUTINE KCHNW (IUNIT, ICH, IER)
```

WHERE:

IUNIT is unit number (0 through 7)
ICH is a single integer variable into which the character is to be stored.
IER is error code -1 =no char. available; + 1= success.

SUBROUTINE KLINE

Subroutine KLINE gets a string of characters from the keyboard (terminated by a carriage return, line feed, or null). The characters are packed two per word. The terminating character is also returned. A string is also terminated if 127 characters are received without a terminator. The user's task is suspended until this call can be completed (a complete string is available). KLINE is called as follows:

```
CALL KLINE (IUNIT, INCR, ISTNG, INCD)
```

Where:

IUNIT is unit number (0 through 3)
INCR is number of characters requested
ISTNG is array to receive characters
INCD is the returned value representing the actual number of characters returned.

APPENDIX A

ERROR REPORTING

7ØØØ system software makes checks for valid instructions, adequate memory, etc. When errors are detected, an error code is displayed on the terminal and the program pauses. Fatal errors cause program execution to terminate. A typical error message from a 7ØØØ graphics program looks like the following:

```
*** ERROR DETECTED
    PICTURE:    XX
    LCP:      XXXX
    ICP:      XXXX
    IERW1:    XXXX
    IERW2:    XXXX
    IMODE:       X

PAUSE IN ERROR ROUTINE
```

IERW1 and IERW2 are error codes (see below), and ICP and LCP are the insert and last component pointers, converted to displacement in the picture. IMODE (Ø, 1, or 2) is insert mode: append, insert or re-write.

ERROR	WORDS	DESCRIPTION
1	Ø	INSUFFICIENT MEMORY FOR PICTURE BEING INITIALIZED (FATAL ERROR) SUBROUTINE BINIT.
1	1Ø	CALL TO SUBROUTINE BTXAT WITHOUT FIRST INITIALIZING PICTURE FOR TEXT USE (FATAL ERROR)
1	11	PICTURE NUMBER SPECIFIED TO SUBROUTINE BTXAT IS LESS THAN 1 OR GREATER THAN 32 (FATAL ERROR)

ERROR	WORDS	DESCRIPTION
1	12	INVALID CHARACTER SIZE SPECIFIED TO SUBROUTINE BTXAT.
1	13	INVALID SCREEN COORDINATES SPECIFIED TO SUBROUTINE BTXAT.
1	20	PICTURE NOT INITIALIZED BEFORE CALL TO SUBROUTINE BPNT.
1	21	PICTURE NUMBER SPECIFIED TO SUBROUTINE BPNT IS LESS THAN 1 OR GREATER THAN 32.
1	31	INVALID INSERT MODE IN CALL TO SUBROUTINE BNSRT.
2	14	LABEL ADDRESS VECTOR OVERFLOW IN CALL TO SUBROUTINE BNSRT.
2	25	INVALID LABEL REFERENCE IN CALL TO SUBROUTINE BGLBL.
2	35	INVALID LABEL REFERENCE IN CALL TO SUBROUTINE BJMLB.
2	45	INVALID LABEL REFERENCE IN CALL TO SUBROUTINE BJSLB.
2	56	INVALID ADDRESS SPECIFIED IN CALL TO SUBROUTINE BJMAD.
2	66	INVALID ADDRESS SPECIFIED IN CALL TO SUBROUTINE BJSAD.
3	17	INSUFFICIENT MEMORY FOR CALL TO SUBROUTINE BTEXT.
5	15	INVALID LABEL REFERENCE IN CALL TO SUBROUTINE GTON.
5	26	INVALID ADDRESS SPECIFIED IN CALL TO SUBROUTINE BDELV.
5	36	INVALID ADDRESS SPECIFIED IN CALL TO SUBROUTINE IPPTR.
6	11	INVALID PICTURE NUMBER IN CALL TO SUBROUTINE PONOF.

6	10	REFERENCED PICTURE NOT INITIALIZED IN CALL TO SUBROUTINE PONOF.
6	20	REFERENCED PICTURE NOT INITIALIZED IN CALL TO SUBROUTINE PSFWD.
7	18	ATTEMPT TO RETRIEVE DATA FROM NON-INITIALIZED JOYSTICK.
7	28	ATTEMPT TO RETRIEVE DATA FROM NON-INITIALIZED DATA TABLET.

ERROR	WORDS	DESCRIPTION
7	38	ATTEMPT TO RETRIEVE DATA FROM NON-INITIALIZED KEYBOARD.
7000	100	INSUFFICIENT MEMORY FOR CURRENT GRAPHICS CALL
7000	101	ATTEMPT TO WRITE ON TRAILER OF CURRENT PICTURE

INDEX

January 19, 1979

MGS-7000 V2 REV 02

The following routines have been added/modified/deleted from the 7000 Software Manual and package:

OLD	V2	DESCRIPTION
BINIT	PINIT	PINIT replaces BINIT. PINIT initializes pictures "OFF" not "ON', and has only two arguments. BINIT is still available
BNSRT	PMODE	PMODE replaces BNSRT. Identical function, BNSRT still available.
BPNT	POPEN	Identical in function, BPNT still available.
BPLBL	LABPT	Identical in function, BPLBL still available.
BGLBL	LABGT	Identical in function, BGLBL still available.
BWORD PSFWD	BWORD PWORD	V2 call requires two arguments (MSW,LSW) in lieu of single two word array.
BTEXT BTXAT	---- ----	Routines not available Routines not available
CHREF*	DREFR*	DREFR replaces CHREF. Identical function, CHREF still available.
DSTRT*	DSTRT*	See Note
DHALT*	DHALT*	See Note
BJMLB BJMAD BJSLB BJSAD	BJMLB BJMAD BJSLB BJSAD	Second Parameter now used to control current intensity (LØ Preset Z; Ø-15 Set Z)
BVJLB	BVJLB	Second parameter added for intensity control (LØ Preset Z; Ø-15 Set Z)
SETZ	BSETZ	Name Change

* NOTE:
A UNIT NUMBER IS NOW REQUIRED.
E.G.: DSTRT(IUNIT)

OLD	V2	DESCRIPTION
KEYON	KEYON	Error Code = +1 For Success, -1 = no such device.
KEYOF	KEYOF	Added unit # parameter
----	KCHNW	Similar to KCHAR: Gets character without any task suspension.Calling sequence: Call KCHNW (IUNIT, ICH, IER)
	UNITS 4-7 are special function keys →	Where IUNIT is unit # (0-7) ICH contains returned CHAR IER = 1 for valid data = 1 for no data
KCHAR	KCHAR	Calling sequence revised to: Call KCHAR (IUNIT, ICH) where IUNIT is unit # (0-7 : 0-3 normal keys 4-7 special function keys) ICH is returned data
KLINE	KLINE	Calling sequence revised to: CALL KLINE (IUNIT, INCR, ISTNG, INCD) where: IUNIT is unit (0-3) INCR is # of characters requested ISTNG is string to receive characters INCD is # of characters actually returned.

RSX GRAPHICS PROGRAMS TASK BUILDING PROCEDURE.

Normal Programs (those without Rasterizer or Display Disk File I/O):

```
TKB PROG = PROG, GRAPHICS/LB
```

Programs using Rasterizer:

```
TKB PROG = PROG, GRAPHICS/LB

   UNITS = 8
   ASG = GP4:7
```

Programs using Disk Display File:

```
TKB PROG = PROG, GRAPHICS/LB

   Units = 9
   ASG = SY: 8
```