Using the the demo programs

The following instructions will allow you to run the demo programs without having to read the whole instruction manual. By running the programs you will gain an appreciation of the various capabilities provided by the graphics support package. This will provide you with a frame of reference when reading the manual. Several functions in these programs assume that there is a joystick connected to A-to-D channels 0 and 1. If you do not have a joystick simply roll a finger over the joystick connector pins on the graphics board. Your body resistance will cause minor changes in the digital values read by the program. Since the A-to-D circuits are low voltage (10 volt) dividers you should not experience any sensations in your finger. When working with the H89 graphics board it is recommended that you at least connect one of the flat ribbon cables to the top most connector (blue stripe up) and then roll your finger over the free end of the cable. This will allow you to do the tests and still keep your hand out of the H89 terminal. If you have any reservations about performing the tests, using your finger as described above, the general idea and capabilities of the graphics boards can be gained without them.

Running the SOUND demo

Load SOUND8 or SOUND89 and follow the next steps.

Select option 1 and depress (CR). The message on the 25th line will describe the type of noise which will be generated when you press the (CR) again. You will depress the (CR) four times while executing the different sounds demonstrated in option 1.

Select option 2 and depress (CR). The prompt on the 25th line will indicate which hardware controlled amplitude shape will be demonstrated when you next press the (CR).

Select option 3 and depress (CR). This will demonstrate a sweep of the tones from 50HZ to 8000HZ.

Select option 4 and depress (CR). This demonstrates sweeping the tone period values from 50 to 8000.

If you have a joystick connected to A-to-D channels 0 and 1 select option 5 and depress (CR). The initial tone produced will be 1000HZ. As this demo runs it will display the joystick 'Y' value and frequency on the 25th line. If 'Y' is less than 100 the tone will go down in frequency and if 'Y' is greater than 140 the tone frequency will increase. When the joystick is about in the center postition (Y between 100 and 140) the tone frequency will remain constant. The lowest frequency which will be produced is 50HZ and the highest has been limited to 8000HZ. Depress the joystick switch to stop this portion of the demo.

Select one of the previous options or '0' to end the program.

Running the the TEST program

Load the TEST8 or TEST89 program as appropriate. Although the functions in TEST can be selected randomly without crashing the program, compatable

Using the the demo programs

The following instructions will allow you to run the demo programs without having to read the whole instruction manual. By running the programs you will gain an appreciation of the various capabilities provided by the graphics support package. This will provide you with a frame of reference when reading the manual. Several functions in these programs assume that there is a joystick connected to A-to-D channels 0 and 1. If you do not have a joystick simply roll a finger over the joystick connector pins on the graphics board. Your body resistance will cause minor changes in the digital values read by the program. Since the A-to-D circuits are low voltage (10 volt) dividers you should not experience any sensations in your finger. When working with the H89 graphics board it is recommended that you at least connect one of the flat ribbon cables to the top most connector (blue stripe up) and then roll your finger over the free end of the cable. This will allow you to do the tests and still keep your hand out of the H89 terminal. If you have any reservations about performing the tests, using your finger as described above, the general idea and capabilities of the graphics boards can be gained without them.

Running the SOUND demo

Load SOUND8 or SOUND89 and follow the next steps.

Select option 1 and depress (CR). The message on the 25th line will describe the type of noise which will be generated when you press the (CR) again. You will depress the (CR) four times while executing the different sounds demonstrated in option 1.

Select option 2 and depress (CR). The prompt on the 25th line will indicate which hardware controlled amplitude shape will be demonstrated when you next press the (CR).

Select option 3 and depress (CR). This will demonstrate a sweep of the tones from 50HZ to 8000HZ.

Select option 4 and depress (CR). This demonstrates sweeping the tone period values from 50 to 8000.

If you have a joystick connected to A-to-D channels 0 and 1 select option 5 and depress (CR). The initial tone produced will be 1000HZ. As this demo runs it will display the joystick 'Y' value and frequency on the 25th line. If 'Y' is less than 100 the tone will go down in frequency and if 'Y' is greater than 140 the tone frequency will increase. When the joystick is about in the center postition (Y between 100 and 140) the tone frequency will remain constant. The lowest frequency which will be produced is 50HZ and the highest has been limited to 8000HZ. Depress the joystick switch to stop this portion of the demo.

Select one of the previous options or '0' to end the program.

Running the the TEST program

Load the TEST8 or TEST89 program as appropriate. Although the functions in TEST can be selected randomly without crashing the program, compatable

functions should be selected if reasonable displays are to be obtained. For example, the VDP must be initialized in the G2 mode (G_INIT) before you can expect the high resolution line and circle drawing routines to produce lines and circles. The following steps will demonstrate some of the capabilities of the VDP and graphics support package. If you have not read the manuals some of the following terminology may seem confusing but you will still be able to gain a visual understanding of your hardware and this support package. Column one below indicates the selection which is to be made and column two discribes the result which should take place.

SELECTION	ACTION
1	Initialize in the multi color mode.
2	Draw multi color lines.
3	Draw multicolor circles.
11	Define four 8x8 sprites.
29	Change the sprites to a 2X magnification.
18	Change the color of the lower three sprites.
1	Reinitialize in the G2 mode (192 x 256).
11	Define four sprites (8x8).
29	Change sprites to 2X magnification.
22	Attach sprite 3 (blue) to the joystick on channels 0 and 1. Move the joystick around and notice how the blue sprite will slide over the yellow sprite (4) which is of a lower priority. The X-Y coordinates will be dynamically displayed on the 25th line of the H89/H19 CRT until the joystick switch is depressed. If the motion of the sprite seems jumpy it is due to the time it takes to write to the H89/H19 screen.
2	Draw G2 mode lines.
23	Fill a subarea of the display.
24	Change the on/off colors to white/magenta.
3	Draw G2 mode circles.
25	Clear the G2 mode screen. Notice that the sprites not cleared as they are not part of the pattern plane on which lines and circles are drawn.
19	Turn the VDP off.
20	Turn the VDP on.
5	Put the G2 mode line and circle drawing routines in the toggle mode. Nothing visiable will happen.
2	Toggle lines on.
3	Toggle circles on.

3 Toggle circles off.

23 Fill the subarea again.

27 Test for screen pixels being turned on. The upper left corner of the red sprite is used as the reference point for locating the sprite. Use the joystick to move the sprite around the screen. As the upper left corner of the sprite goes over a pixel which is turned on the color of the sprite in the center of the screen will change. Depress the joystick switch to end this function.

17 Turn on the 'early bit' for sprites one thru three. This moves the sprites 32 pixels to the left.

31 Reset the 'early bits'.

16 Move the first three sprites such that one half of them is off the screen.

1 Reinitialize in the G2 mode for 8x8 sprites.

11 Define four sprites.

29 Make them 2X in size.

13 Define three more sprite patterns not attached to attributes and thus they will not be seen on the screen.

15 Assign the new patterns to the attributes originally used by by sprites 1 thru 3.

26 Initialize in G2 mode for 16x16 sprites with magnification set for 2X.

12 Define four 16x16 sprites at a magnification of 2X.

28 Put sprites back to 1X magnification.

14 Define three new 16x16 sprite patterns. You should not see any change to the screen.

15 Assign the new patterns to sprites 1 thru 3.

21 Draw a fan of lines using MOVE and DRAW rather than PLOT and DRAW commands.

30 Initialize in the pattern mode.

2 Draw G2 mode lines. This is an incompatable operation and results in a messed up screen but does not crash the program or harm the hardware.

3 Draw G2 mode circles. Again, this is incompatable and further messes up the screen.

Now you are on your own to experiment with the functions in any order you like or you may stop the program by entering a selection of '0'.

The ACIRC and CCIRC programs are provided in the languages you ordered. The comments in these programs should assist you in understanding how the the graphics calls work and provide you with a foundation upon which you can write your own programs.

When run, the CCIRC program draws concentric circles and generates tones proportional to the radius of the circles. To stop the program depress any joystick switch which in connected to the 'A' input/output port on the graphics board. Hold the switch down until the circles are all erased. If you do not yet have a joystick then you will have to do a hard reset or ^Z^Z under HDOS to stop the program.

ACIRC draws circles and a 'rubber band' line using the toggle mode of line drawing. In the toggle mode the lines and circles are rapidly drawn and erased such that figures can be dynamically moved around the screen. In ACIRC, the color of the lines and 8x8 pixel sprite change each time the joystick switch is depressed. The radius of the circle is proportional to the Y coordinate of the sprite. One end of the 'rubber band' line is connected to the center of the display area and the other is connected to the X-Y coordinates read from the joystick position. To stop the program move the sprite off of the screen or use ^Z^Z under HDOS or a hard reset under CP/M.

NOTE: Due to the size fo the executable load modules, support programs and data sets for COBOL and MBASIC only the source code for ACIRC is included for these languages. Thus, users of COBOL will have to compile and link the ACIRC.COB file. MBASIC users will use the ACIRC.DAT file as input to BASLOAD to generate the ACIRC.MEM file and first part of the MBASIC program. Follow the instructions in the language supplement. Also, be sure that you do do this with data sets copied from the distribution disks. You will also have to generate an IOH8.REL or IOH89.REL file as described in Appendix A of the user manual. This will give you practice generating graphics programs as you will be starting with correct source code.

PASCAL MT+ LANGUAGE SUPPLEMENT

The components of this support package have been designed such that they may be used in a wide variety of system configurations. Programmers using Pascal MT+ may use the considerable flexability of that system to customize that compiler to their hardware. The GSLs are set up so that the same program code can be used to drive both the H8 and H89 versions of the graphics boards.

A way of having programs with the same basic name on the same disk is to use different extensions, such as .H8 and .H89 instead of .ABS. or .COM. Such programs can be run by changing the desired version so that it has the .ABS or .COM extension.

The file EXTDECS.PAS contains the declarations necessary to use the MT+ compatible H8PAS and H89PAS files. Note that not all of these declarations are needed for any particular program. Note also that if you try to compile a program using the whole EXTDECS.PAS file you may run out of symbol table space. The proper technique to use is to delete from a copy of EXTDECS.PAS the declarations for all functions and procedures which will not be used by the program. This will free up considerable symbol table space. To gain additional table space you can also use the "K" toggles which are documented in the Pascal MT+ user manual. See the TEST.* files on the demo disk for an example of a reduced EXTDECS.PAS include file. The reduced file is named TEST.DEC to indicate that the file is the external declarations for the program TEST.

Linking the GSL modules with your program is done using the standard MT+ linking techniques. Representative compile and link command lines are provided below.

```
Compile; PAS YOURPROG

Link; LINKMT YOURPROG,H8PAS/S,IOH8,PASLIB/S
```

Note the data set SYn:IOH8. This is the file you produced by running the SETHOS8 or SETCPM8 program as described in Appendix A of the main manual. If you are using an H89 then link in IOH89 instead of IOH8. If you are using floating point math you will need to include the appropriate library file for the math routines in between the IOH8 or IOH89 and PASLIB.

GRAPH-PAC-II

HARDWARE DEVICE DRIVER AND BASIC SUPPORT ROUTINES

FOR THE

HA-8-3 AND HA-89-3

COLOR GRAPHICS AND SOUND GENERATION BOARDS

Fred Pospeschil

Dave Troendle

CONTENTS
========

General Introduction

GRAPH-PAC-II provides an integrated set of utility procedures and functions which allow the programmer to directly access and manipulate the hardware components of the HA-8-3/HA-89-3 graphics boards from MBASIC and compiled MBASIC (CP/M), FORTRAN, COBOL, C80, Pascal MT+ and MACRO-80 assembler language code. These routines provide all of the capabilities contained in the HA-8-3/HA-89-3 distribution disk files. In addition, they provide higher level composite calls which facilitate sprite manipulation, line drawing, and circle generation within the display area.

Although the graphics hardware manuals should be read and studied before initial programming efforts are begun, the use of these routines will greatly reduce the depth of understanding needed to program the graphics board. Use of these routines will also speed up program development, improve program reliability, and improve code level communication between graphics users and programmers. If you are just beginning to use the graphics/sound board it is recommended that you:

1. Read this manual for general orientation to the capabilities of the board and support provided by this set of software routines.

2. Study the hardware manuals which were supplied with the board to gain familiarity with their organization and basic content.

3. Study each section of this manual along with the referenced portions of the hardware manuals.

4. Study the language supplement and Appendix A before trying to write or compile a program.

Considerable effort has been devoted to optimizing the algorithms and code used to control the hardware and to perform drawing lines and circles. Since these routines are written in 8080 assembly language, the final application program performance should be optimized thru the use of these routines. In addition, they can be used with any language which supports the MICROSOFT .REL load module format on both 8080- and Z80-based systems. Special interfaces have been developed for users of MBASIC. Refer to the language and run-time supplement for additional information data representations and program preperation considerations which apply to the host language you are using.

The documentation for these routines is arranged such that the basic, or most primitive, functions and procedures are described first. These are followed by the higher level composite functions and procedures which will tend to be used in the vast majority of graphic application programs. Thus, most programmers will find the initial portion of this documentation most useful as a transition, or bridge, from the hardware level descriptions and considerations covered in the manuals distributed with the HA-8-3/HA-89-3. References back to the hardware manuals are provided where appropriate.

As appropriate, the recommended system constants to be used with this package are included with the language specific supplements which are included with this manual.

For maximum cross-language compatibility, all procedure and function (subroutine) parameters are passed as 16 bit values. In some of the routines it was practical to allow MACRO-80 programs to pass arguments in the 8080/Z80 registers. By using this approach the calling and called routines can normally save several instructions per call. For each routine in which this technique can be used a special "MACRO-80 NOTE:" is provided. MACRO-80 programs should not include the underscore in subroutine names. For example, R_CHAN should be written as RCHAN. All assembly language programs must save any registers which have needed data in them as the subroutines do not save and restore registers upon entry and before exit. Since language implementations change over time, the coding and calling conventions in your programs may have to be changed as your language implementation changes.

For maximum speed and to keep these basic routines small, it is assumed that only legal values are passed to them. If the user of the application can input illegal or out-of-range values then the application program must provide appropriate editing of the input data before calling these routines.

Appreciation is extended to Mr. Larry Reeve with whom we have had many discussions on how best to support the graphics boards. Due to this common involvement, users of the Lucidata Pascal Graphics Compiler from Polybytes will find considerable similiarities between these packages. That these software packages should have a similiar approach, and in some cases use the same words, was an explicit design goal to facilitate communication between programmers and users of the graphics boards.

Appreciation is also extended to Phil Evans, Bob Cole, and Mark Kroska of the OMAHA Heath Users Group for their help in editing this manual and testing the software. The users of GRAPH-PAC-I with whom we have had many pleasant discussions are also to be commended for their many ideas and suggestions which lead to this version.

Although some of the early programs for the HA-8-3 used the H19 key pad as an input device, the use of joysticks is strongly recommended for the most natural and effective use of your system. Appendix B provides instructions on how to wire inexpensive, yet effective, joysticks to the system.

In Pascal MT+ and Tiny Pascal the underscore (_) can be used to improve program readability. It does not count as a significant character in variable and procedure/function names. Thus P_NAME is the same as PNAME. All names used in this support package were limited to six characters so as to be compatible with languages such as MACRO-80 and FORTRAN.

The routines provided in this package may be used on any number of CPU's directly owned by the purchaser and making backup copies is strongly recommended. Selling or giving these routines to others violates United States copyright laws. However, selling or sharing programs which have the relocatable modules linked into them is permitted and encouraged.

When writing with respect to specific coding problems please include as much printed documentation as possible and clearly indicate which language support package is being used. When practical, providing source code and executable load modules which demonstrate the problem will facilitate diagnosing problems. The user is fully responsible for testing and certifying all programs which use these routines.

Suggested changes and corrections to this document should be sent to:

Fred Pospeschil
3108 Jackson St.
Bellevue, NE 68005

TRADEMARKS

BASIC PRIMITIVES

READ ANALOG CHANNEL

```
PASCAL   ADVALUE := A_D_CHAN(CHANNEL);

FORTRAN ADVAL = ADCHAN(CHAN)

C80      advalue = adchan(channel);

MBASIC   CALL ADCHAN(CHAN%)
         CALL BASVAL(ADVAL%)

COBOL    CALL "ADCHAN" USING CHANNEL.
         CALL "COBVAL" USING ADVALUE.
```

The graphics board contains eight analog-to-digital input channels (numbered 0 thru 7) which can be used by the application program. The analog-to-digital hardware converts the analog input voltage to a digital value between 0 and 255. This value is read by the function A_D_CHAN. The calling program specifies the desired analog channel in the agrument "CHANNEL". Legal values for "CHANNEL" are "0" thru "7". A_D_CHAN returns an integer value of 0 thru 255 depending on the position and type of joystick being used. The argument "CHANNEL" can be either the numbers 0 thru 7 or a variable name of type integer which has been assigned a value of 0 thru 7. It is the responsibility of the calling program to ensure that only values 0 thru 7 are passed to the function.

The analog-to-digital converters require about 35 micro seconds, to convert an analog value to a digital representation. The A_D_CHAN procedure for the HA-8-3 uses a software timing loop which will handle both two- and four-megahertz CPUs. The HA-89-3 version of the procedure uses the HA-89-3's Programmable Interrupt Controller (PIC) to determine if the A-To-D conversion has been completed. The VDP initialization routines G_INIT, P_INIT and M_INIT initialize the PIC to support this operation.

MACRO-80 NOTE: The alternate entry point for A_D_CHAN is RCHAN. To read an A-to-D channel, load the desired channel (0-7) in the A register, CALL RCHAN, and the value will be returned on the stack. For example:

```
LDA  0          ;SET FOR READING CHAN 0
CALL RCHAN      ;CALL A-TO-D FUNCTION
POP  H          ;H = UNDEFINED, L = A-TO-D VALUE
```

READ VDP STATUS

```
PASCAL   STATUS := STATS;

FORTRAN  STATUS = STATS

C80      status = stats();

MBASIC   CALL STATS
         CALL BASVAL(STATUS%)

COBOL    CALL "STATS".
         CALL "COBVAL" USING STATUS.
```

STATS returns the contents of the eight bit read-only VDP status register. VDP status register layout is shown at the bottom of VDP manual page 10. The contents are described in paragraphs 2.5 thru 2.5.3, VDP manual page 12. One of the main uses of the VDP status register is to detect when sprites have pixels which overlap (the sprites have collided).

MACRO-80 NOTE: STATUS is returned on the stack. To obtain the contents of the VDP status register CALL STATS and then POP HL. After these instructions are executed, the contents of the VDP status register will be in the L register.

SET VDP OPTION REGISTER

PASCAL V_OPTS(OPTIONS);

FORTRAN CALL VOPTS(OPTS)

C80 vopts(options);

MBASIC CALL VOPTS(OPTIONS)

COBOL CALL "VOPTS" USING OPTIONS.

Procedure V_OPTS loads the integer value of "OPTIONS" into VDP registers 0 and 1. The most significant byte of "OPTIONS" is loaded into register 0 and the least significant byte into register 1. Refer to VDP manual pages 9-11 for the description of the bit positions used in these registers.

MACRO-80 NOTE: The alternate entry point for V_OPTS is VP.SOP. Load the bit pattern for VDP register 0 in the H register and the pattern for VDP register 1 in the L register. Follow these instructions with a call the VP.SOP. For example:

```
MVI     H,00000010B     ;Pattern for VDP reg 0
MVI     L,10000000B     ;Pattern for VDP reg 1
CALL    VP.SOP          ;Set the options
```

SET VDP ADDRESS OFFSET REGISTERS

In the following five procedures the argument "ADDRESS" is an integer value in the range of 0-16K. The calling program must ensure that the arguments are within this range and on the proper memory boundaries. These procedures perform the necessary transformations of the addresses before loading them into the VDP offset registers. To be on an acceptable boundary the address must be evenly divisible by the boundary value listed with each procedure. Initial recommended values (IRV) for each register are provided below. These values allow full use of the VDP while only using one quarter of the available graphics memory. Thus four complete sets of tables can be defined at one time when working in the pattern mode. The high resolution mode (G2) uses all of the 16K graphics memory and thus does not permit multiple images to be defined at the same time (see G_INIT below).

SET PATTERN NAME TABLE ADDRESS

```
PASCAL   P_NAME(ADDRESS);   Boundary = 1024   IRV = 1024

FORTRAN  CALL PNAME(ADDRS)

C80      pname(addrs);

MBASIC   CALL PNAME(ADDRS%)

COBOL    CALL "PNAME" USING ADDRESS.
```

SET COLOR GENERATOR TABLE ADDRESS

```
PASCAL   C_GEN(ADDRESS);   Boundary = 64      IRV = 1792

FORTRAN  CALL CGEN(ADDRS)

C80      cgen(addrs);

MBASIC   CALL CGEN(ADDRS%)

COBOL    CALL "CGEN" USING ADDRESS.
```

SET PATTERN GENERATOR ADDRESS

```
PASCAL   P_GEN(ADDRESS);    Boundary = 2048     IRV = 2048
FORTRAN  CALL PGEN(ADDRS)
C80      pgen(addrs);
MBASIC   CALL PGEN(ADDRS%)
COBOL    CALL "PGEN" USING ADDRESS.
```

SET SPRITE NAME TABLE ADDRESS

```
PASCAL   S_NAME(ADDRESS);    Boundary = 128     IRV = 1920
FORTRAN  CALL SNAME(ADDRS)
C80      sname(addrs);
MBASIC   CALL SNAME(ADDRS%)
COBOL    CALL "SNAME" USING ADDRESS.
```

SET SPRITE PATTERN GENERATOR TABLE ADDRESS

```
PASCAL   S_GEN(ADDRESS);    Boundary = 2048     IRV = 0
FORTRAN  CALL SGEN(ADDRS)
C80      sgen(addrs);
MBASIC   CALL SGEN(ADDRS%)
COBOL    CALL "SGEN" USING ADDRESS.
```

The VDP offers considerable flexibility in assigning memory usage. Because of this, the user is cautioned to carefully study the VDP manual before using the above five procedures (refer to VDP manual pages 9-12).

MACRO-80 NOTE: Alternate entry points for the above routines are:

```
P_NAME = VDPSPN
C_GEN  = VDPSCG
P_GEN  = VDPSPG
S_NAME = VDPSSN
S_GEN  = VDPSSG
```

To use these alternate entry points, load the HL register pair with the desired address and then call the appropriate routine. For example, to set the pattern name table address to 6144, use the following instructions: LXI H,6144 followed by CALL VDPSPN.

SET BORDER COLOR VALUE

```
PASCAL   BORDR(VALUE);

FORTRAN  CALL BORDR(VALUE)

C80      bordr(value);

MBASIC   CALL BORDR(VALUE%)

COBOL    CALL "BORDR" USING VALUE.
```

"VALUE" is an integer value in which the right most four bits (values 0 thru 15) set the border color for all VDP display modes. In the text mode they also set the display background color (color 0). Bits 5 thru 8 are the text color 1 value. In the non-text modes, bits 5 thru 8 are ignored. See VDP manual pages 10 and 12.

MACRO-80 NOTE: Alternate entry point for BORDR is VDPSTB and it expects the VALUE to be in the L register. Values in the H register are ignored.

READING AND WRITING VDP DISPLAY MEMORY (VRAM)

WRITE BYTE DIRECTLY TO VRAM

PASCAL W_B_DIR(DATA, ADDRESS);

FORTRAN CALL WBDIR(DATA, ADDRS)

C80 wbdir(data, addrs);

MBASIC CALL WBDIR(DATA%, ADDRESS%)

COBOL CALL "WBDIR" USING DATA, ADDRESS.

This procedure causes the "DATA" (must be a integer value in the range of 0 thru 255) to be written into VRAM at the specified address (an integer value value in the range of 0 thru 16K).

MACRO-80 NOTE: The parameters for this routine must be passed on the stack. Recommend using the following instructions:

```
LXI   H,DATA_VALUE
PUSH  H
LXI   H,ADDRESS_VALUE
PUSH  H
CALL  WBDIR
```

WRITE BYTE TO NEXT VRAM ADDRESS

```
PASCAL  W_NEXT(DATA);

FORTRAN CALL WNEXT(DATA)

C80     wnext(data);

MBASIC  CALL WNEXT(DATA%)

COBOL   CALL "WNEXT" USING DATA.
```

The W_NEXT procedure loads the "DATA" value in the VRAM location following the last one written. Since this procedure exploits the auto-increment feature of the VDP, it must be preceded by a call to W_B_DIR which sets the initial write address. After the initial call to W_B_DIR one or more calls to W_NEXT can be made.

MACRO-80 NOTE: The alternate entry point for this routine is VP.WRV. After using WBDIR to set the write address and write the first byte, successive bytes can be written by loading them into the A register and calling VP.WRV

READ BYTE DIRECTLY FROM VRAM

```
PASCAL   PROGRAM_VARIABLE := R_B_DIR(ADDRESS);

FORTRAN PVAR = RBDIR(ADDRS)

C80      pvar = rbdir(addrs);

MBASIC   CALL RBDIR(ADDRS%)
         CALL BASVAL(PVAR%)

COBOL    CALL "RBDIR" USING ADDRESS.
         CALL "COBVAL" USING PROGRAM-VARIABLE.
```

R_B_DIR causes the VRAM address pointer to be set to the indicated value (in the range of 0 to 16K) and then assigns the byte value found at that location to the program variable. This procedure has the necessary wait loop built into it to handle the VDP address settling time requirements. Thus, when reading a series of bytes from VRAM, RBDIR should be used for reading the first byte and RNEXT should be used for reading all following successive bytes.

MACRO-80 NOTE: To use this routine, PUSH the VRAM address onto the stack, CALL RBDIR, and then POP the stack into a register pair. If, for example, the stack is POPed into the HL registers, the VRAM data will be in the L register and the H register will contain 0.

READ BYTE FROM NEXT VRAM ADDRESS

```
PASCAL   PROGRAM_VARIABLE := R_NEXT;

FORTRAN  PVAR = RNEXT

C80      pvar = rnext();

MBASIC   CALL RNEXT
         CALL BASVAL(PVAR%)

COBOL    CALL "RNEXT".
         CALL "COBVAL" USING PROGRAM-VARIABLE.
```

The function R_NEXT assigns the value stored in the next VRAM memory location to the program variable. This function also uses the VDP auto-increment feature and therefore must be preceded by a call to R_B_DIR. After an initial call to R_B_DIR one or more calls to R_NEXT can be made.

MACRO-80 NOTE: Calling RNEXT puts the contents of the next VRAM location on the stack. If the stack is POPed into the HL register pair, the H register will contain 0 and the VRAM data will be in the L register.

ENABLE / DISABLE VIDEO DISPLAY

```
PASCAL   VDP_ON;    Turn VDP display on

FORTRAN  CALL VDPON

C80      vdpon();

MBASIC   CALL VDPON

COBOL    CALL "VDPON".


PASCAL   VDP_OFF;   Turn VDP display off

FORTRAN  CALL VDPOFF

C80      vdpoff();

MBASIC   CALL VDPOFF

COBOL    CALL "VDPOFF".
```

These procedures are used to control whether on not the patterns and sprites defined in the VRAM are displayed. The procedures can be used with the Pattern, Multicolor, and Graphics 2 modes. If major updates to the the color or pattern tables cause distracting displays, the procedures should be used to turn off the display momentarily.

CLEAR VDP SCREEN

```
PASCAL   G_CLEAR;

FORTRAN  CALL GCLEAR

C80      gclear();

MBASIC   CALL GCLEAR

COBOL    CALL "GCLEAR".
```

G_CLEAR will rapidly clear the display screen by writing 0's to all pattern generator table storage locations. It is designed to work in conjunction with the VRAM allocation set up by G_INIT and thus assumes that the pattern generator table begins at VRAM location 0. Clearing the display with G_CLEAR is considerably faster than using G_INIT to reinitialize all VRAM storage tables. In addition, it does not disable the sprites, thus permitting the pattern plane to be cleared without disturbing any sprites which may be in the display area.

PAUSE

```
PASCAL   PAUSE(TIME);

FORTRAN  CALL PAUSE(TIME)

C80      pause(time);

MBASIC   CALL PAUSE(TIME%)

COBOL    CALL "PAUSE" USING TIME.
```

PAUSE provides an easy way of halting further program execution for a specified length of time. When using PAUSE, the argument TIME is in two milli-second increments to be consistant with the PAUSE procedure in Tiny Pascal. (also see "PSG CONTROL CONSIDERATIONS"). The system software clock MUST be running when this procedure is called.

COMPOSITE PRIMITIVES

The following procedures provide more advanced capabilities for controlling the images to be displayed by the VDP. Through the use of these procedures, the programmer can concentrate more on the actual application being developed and less on the VDP hardware characteristics. In all of the following functions and procedures the horizontal and vertical position arguments are defined with 0,0 being in the lower left corner of the screen. The range of the horizontal and vertical coordinates for each function depends on the mode being used. In Graphics 2 mode, or when controlling sprite positions, the horizontal coordinate (X) system varies from 0 at the left border of the screen, to 255 at the right border. In Multicolor mode, the horizontal coordinate system varies from 0 at the left border of the screen to 63 at the right border. The vertical coordinate (Y) system ranges from 0 at the bottom to 191 at the top in the Graphics 2 mode, and from 0 at the bottom to 47 at the top in the Multicolor mode. NOTE: The VDP hardware has the 0,0 point at the upper left corner of the screen (see VDP manual page 10). Since many existing algorithms and thought patterns are set up to work with the 0,0 point at the lower left, the VRAM address calculation routine has been written to logically shift the X=0 and Y=0 point to the lower left as it could be done faster in assembly language (one instruction) than in any higher level language routine or statement.

It should be noted that not all software and not all joysticks are set up for the 0,0 location being at the lower left corner of the screen. The question then becomes, how does one use a joystick wired for GRAPH-PAC, which has 0,0 at the lower left? It's really quite simple. One way is to exchange the "ground" and "vref" wires on the Y-AXIS pot of the joystick. Another is to install a small double pole switch and use it to switch these wires as needed. Yet another way is to change the software to retranslate the "Y" coordinate. For example, if you are reading a joystick plugged into analog channels 0 and 1 by using statements such as

```
X := A_D_CHAN(0);   Y := A_D_CHAN(1);
```

simply change the second statement to

```
Y := 191 - A_D_CHAN(1);
```

and the desired results will be achieved.

INITIALIZE VDP

PASCAL G_INIT(MAG, SIZE, COLOR0, COLOR1, BORDER); G2 MODE

FORTRAN CALL GINIT(MAG, SIZE, COLOR0, COLOR1, BORDER)

C80 ginit(mag, size, color0, color1, border);

MBASIC CALL GINIT(MAG%, SIZE%, COLOR0%, COLOR1%, BORDER%)

COBOL CALL "GINIT" USING MAG, SIZE, COLOR0, COLOR1, BORDER.

PASCAL P_INIT(MAG, SIZE, COLOR0, COLOR1, BORDER); PATTERN MODE

FORTRAN CALL PINIT(MAG, SIZE, COLOR0, COLOR1, BORDER)

C80 pinit(mag, size, color0, color1, border)

MBASIC CALL PINIT(MAG%, SIZE%, COLOR0%, COLOR1%, BORDER%)

COBOL CALL "PINIT" USING MAG, SIZE, COLOR0, COLOR1, BORDER.

PASCAL M_INIT(MAG, SIZE, COLOR1, BORDER); MULTICOLOR MODE

FORTRAN CALL MINIT(MAG, SIZE, COLOR1, BORDER)

C80 minit(mag, size, color1, border);

MBASIC CALL MINIT(MAG%, SIZE%, COLOR1%, BORDER%)

COBOL CALL "MINIT" USING MAG, SIZE, COLOR1, BORDER.

MAG ----- 0 for small sprites (NOMAG)
1 for large sprites (MAG)

SIZE ---- 0 for 8x8 sprites (SSIZE)
1 for 16x16 sprites (LSIZE)

COLOR0 -- Color for OFF pixels (0 - 15 in G1/G2 only)

COLOR1 -- Color for ON pixels (0 - 15 in G1/G2 modes)
This is the initial color of all pixels in the multi-color mode.

BORDER -- 0 thru 15 for desired color

NOTE: In the Pattern mode the pattern name and generator tables are initialized to a default value of "0". All 32 color table

entries are set to the color1/color0 values.

The procedures on the previous page allocate VRAM and initialize the VDP to the proper state for the specified mode. It is important to note that the G_INIT,P_INIT, and M_INIT procedures should not be used together, because one will cancel the actions of the other.

WARNING! The VDP must be properly initialized by calling one of the above procedures before calling any of the other graphics routines in this package or else a system crash can be expected. This initialization can be accomplished by calling any of the above routines or by using the more basic primitives to load the VDP registers and fill the graphics buffers with proper data.

VRAM ALLOCATION

G_INIT	P_INIT	M_INIT	
0	2048	2048	-- Pattern Generator Table
6144	1024	1024	-- Pattern Name Table
7168	1920	1920	-- Sprite Attribute Table
8192	1792	N/A	-- Pattern Color Table
14336	0	0	-- Sprite Generator Table

PLOTTING POINTS AND DRAWING LINES AND CIRCLES

PLOT POINTS

```
PASCAL   PLOT(X, Y);              - G2 mode

FORTRAN  CALL PLOT(X, Y)

C80      plot(x, y);

MBASIC   CALL PLOT(X%, Y%)

COBOL    CALL "PLOT" USING X, Y.


PASCAL   MCPLOT(X, Y, COLOR);     - Multicolor Mode

FORTRAN  CALL MCPLOT(X, Y, COLOR)

C80      mcplot(x, y, color);

MBASIC   CALL MCPLOT(X%, Y%, COLOR%)

COBOL    CALL "MCPLOT" USING X, Y, COLOR.

     X ------ Horizontal position

     Y ------ Vertical position

     COLOR -- Color for point in Multicolor mode only.
```

This procedure will plot the smallest definable cell in either plotting mode. In Multicolor mode the COLOR argument specifies the color which is to be used at the X,Y location.

MOVE THE X,Y COORDINATES

```
PASCAL   MOVEXY(X, Y);

FORTRAN  CALL MOVEXY(X, Y)

C80      movexy(x, y);

MBASIC   CALL MOVEXY(X%, Y%)

COBOL    CALL "MOVEXY" USING X, Y.
```

X ------ Horizontal position

Y ------ Vertical position

The MOVEXY procedure is used to change the X,Y coordinates to a new location without affecting the pixel at that location. It operates the same in both the multi-color and pattern modes.

TEST THE STATE OF A SPECIFIED G2 MODE PIXEL

```
PASCAL   PIXEL_STATE := TEST_XY(X, Y);

FORTRAN  PSTATE = TESTXY(X, Y)

C80      pstate = testxy(x, y);

MBASIC   CALL TESTXY(X%, Y%)
         CALL BASVAL(PSTATE%)

COBOL    CALL "TESTXY" USING X, Y.
         CALL "COBVAL" USING PIXEL-STATE.
```

X ------ Horizontal position

Y ------ Vertical position

The TEST_XY procedure is used to test the status of an X,Y location without affecting the pixel at that location. It is a boolean function in that it returns TRUE (1) if the pixel is on and FALSE (0) if the pixel is off.

DRAW LINES

```
PASCAL   DRAW(X, Y);                - G2 mode

FORTRAN  CALL DRAW(X, Y)

C80      draw(x, y);

MBASIC   CALL DRAW(X%, Y%)

COBOL    CALL "DRAW" USING X, Y.


PASCAL   MCDRAW(X, Y, COLOR);       - Multicolor Mode

FORTRAN  MCDRAW(X, Y, COLOR)

C80      mcdraw(x, y, color);

MBASIC   CALL MCDRAW(X%, Y%, COLOR%)

COBOL    CALL "MCDRAW" USING X, Y, COLOR.
```

X ------ Horizontal position

Y ------ Vertical position

COLOR -- Color for point in Multicolor mode only

This procedure draws a line from the last point referenced in either a DRAW/MCDRAW, PLOT/MCPLOT or MOVEXY call. In Multicolor mode COLOR defines the color of the line to be drawn.

G2 POINT PLOTTING AND LINE DRAWING CONTROL

```
PASCAL   G2_ON;       Turn pixels on
FORTRAN  CALL G2ON
C80      g2on();
MBASIC   CALL G2ON
COBOL    CALL "G2ON".

PASCAL   G2_OFF;      Turn pixels off
FORTRAN  CALL G2OFF
C80      g2off();
MBASIC   CALL G2OFF
COBOL    CALL "G2OFF".

PASCAL   G2_TOGG;     Toggle the pixels
FORTRAN  CALL G2TOGG
C80      g2togg();
MBASIC   CALL G2TOGG
COBOL    CALL "G2TOGG".
```

The G2_ON, G2_OFF,and G2_TOGG procedures are used to control the operation of the PLOT and DRAW procedures. After a mode is set, the PLOT and DRAW procedures will continue to turn on, turn off, or toggle (reverse or flip) the pixels until a different mode is set.

DRAW CIRCLE

```
PASCAL   CIRCLE(X, Y, R);          - G2 mode
FORTRAN  CALL CIRCLE(X,Y, R)
C80      circle(x, y, r);
MBASIC   CALL CIRCLE(X%, Y%, R%)
COBOL    CALL "CIRCLE" USING X, Y, R.

PASCAL   M_CIRC(X, Y, R, COLOR);  - Multicolor Mode
FORTRAN  CALL MCIRC(X, Y, R, COLOR)
C80      mcirc(x, y, r, color);
MBASIC   CALL MCIRC(X%, Y%, R%, COLOR%)
COBOL    CALL "MCIRC" USING X, Y, R, COLOR.
```

X ------ Horizontal position of circle center

Y ------ Vertical position of circle center

R ------ Radius of circle in screen pixels

COLOR -- Color of line used to draw circle in MC mode

These procedures are used to rapidly draw circles in both the G2 and Multicolor modes. When a portion of the circle would fall outside of the display area, the circle drawing routine will connect the visible portions of the circle with straight lines which run along the outermost pixels of the display area.

GRAPHICS 2 MODE AREA MANIPULATION PROCEDURES

AREA COLOR CONTROL

PASCAL G2_FILL(XMIN, XMAX, YMIN, YMAX, COLOR0, COLOR1);

FORTRAN CALL G2FILL(XMIN, XMAX, YMIN, YMAX, COLOR0, COLOR1)

C80 g2fill(xmin, xmax, ymin, ymax, color0, color1);

MBASIC CALL G2FILL(XMIN%, XMAX%, YMIN%, YMAX%, COLOR0%, COLOR1%)

COBOL CALL "G2FILL" USING XMIN, XMAX, YMIN, YMAX, COLOR0, COLOR1.

XMIN ---- Left side of the area

XMAX ---- Right side of the area

YMIN ---- Bottom side of the area

YMAX ---- Top side of the area

COLOR0 -- Color of OFF pixels in the area

COLOR1 -- Color of ON pixels in the area

This procedure defines the two colors that will be used for an area of the screen. All pixels within the bounds of the area defined by XMIN, XMAX, YMIN, and YMAX will be affected by this procedure. In addition, because of the method used by the 9918A VDP for color definition in the Graphics 2 mode, some areas to the left and to the right of the defined area may be affected. The actual left and right borders can be computed with the following equations:

ACTUAL_XMIN = (XMIN DIV 8) * 8

ACTUAL_XMAX = XMAX - (XMAX MOD 8) + 7

AREA COLOR FILL

PASCAL A_FILL(XMIN, XMAX, YMIN, YMAX);

FORTRAN CALL AFILL(XMIN, XMAX, YMIN, YMAX)

C80 afill(xmin, xmax, ymin, ymax);

MBASIC CALL AFILL(XMIN%, XMAX%, YMIN%, YMAX%)

COBOL CALL "AFILL" USING XMIN, XMAX, YMIN, YMAX.

XMIN -- Left side of the area

XMAX -- Right side of the area

YMIN -- Bottom side of the area

YMAX -- Top side of the area

This procedure is used to control the on/off state of pixels in a sub-area of the graphics display. Since it uses the line drawing routine to paint the sub-area, this routine will turn on/off/or toggle all of the points in the specified area depending on the last G2-mode set.

RANDOM NUMBERS

PASCAL	SEED(NEW_SEED);	RANDOM_NUMBER := RAND;
FORTRAN	CALL SEED(NSEED)	RNUM = RAND
C80	seed(nseed);	rnum = rand();
MBASIC	CALL SEED(NSEED%)	CALL RAND CALL BASVAL(RNUM%)
COBOL	CALL"SEED" USING NEW-SEED.	CALL"RAND". CALL "COBVAL" USING RAND-NUMB.

NEW_SEED - Any integer number other than 0 (zero). Used to initialize the stream of random numbers returned by successive calls to RAND.

The random numbers returned by RAND are in the range of +/- 32K. The numbers are basically CRC-16 numbers. The method used to compute the CRC-16 is an adaptation of the method described by Suresh Vasa in the May, 1976 issue of COMPUTER DESIGN. After once calling SEED, RAND can be called any number of times. SEED can also be called whenever desired to start a new series of random numbers.

RESET VDP G2 MODE "ON/OFF" COLORS

PASCAL SET_G2_C(COLOR_OFF, COLOR_ON);

FORTRAN CALL SETG2C(COFF, CON)

C80 setg2c(coff, con);

MBASIC CALL SETG2C(COLOROFF%, COLORON%)

COBOL CALL "SETG2C" USING COLOR-OFF, COLOR-ON.

COLOR_OFF - 0 thru 15. Sets color for "OFF" pixels.

COLOR_ON -- 0 thru 15. Sets color for "ON" pixels.

The SET_G2_C procedure will rapidly reset the G2 mode pattern plane colors without disturbing the sprite or border colors. Any lines, circles, etc., which are displayed will have their colors changed to the new color specified by COLOR_ON and the background will be changed to COLOR_OFF.

LINE AND POINT COLOR CONTROL

PASCAL LCOLOR(LINE_COLOR, BACKGROUND_COLOR);

FORTRAN CALL LCOLOR(LCOLOR, BCOLOR)

C80 lcolor(lcolor, bcolor);

MBASIC CALL LCOLOR(LCOLOR%, BCOLOR%)

COBOL CALL "LCOLOR" USING LINE-COLOR, BACKGROUND-COLOR.

LINE_COLOR ------- Color of lines and points drawn subsequent to calling this procedure. Color can be called as many times as needed to change and rechange the line and point colors.

BACKGROUND_COLOR - Color of background area surrounding the lines and points drawn subsequent to last calling this procedure.

The LCOLOR procedure allows establishing new colors for lines and points without resetting the color of points and lines already drawn. Due to the manner in which the 9918A manipulates the color plane some of the pixels close to the new lines and points may also be changed. If the background color specified in this procedure is different than that already displayed then an eight pixel wide area of the background will be set to the new background color. This process can be used to create some interesting effects. However, it can also cause some possibly undesired effects. Although difficult to explain in words, its is readily understood when the effects are seen. The ACIRC demonstration program is designed to illustrate the results of using this procedure.

GRAPHIC BARS

PASCAL G_BAR(XSTART, DELTAX, YSTART, DELTAY, COLOR);

FORTRAN CALL GBAR(XSTART, DELTAX, YSTART, DELTAY, COLOR)

C80 gbar(xstart, deltax, ystart, deltay, color);

MBASIC CALL GBAR(XSTART%, DELTAX%, YSTART%, DELTAY%, COLOR%)

COBOL CALL "GBAR" USING XSTART, DELTAX, YSTART, DELTAY, COLOR.

XSTART - Starting position of the bar on the X axis. Since this procedure changes the background colors on the pattern plane each X-increment of a bar is eight pixels wide. Thus there is a maximum of thirty two bar positions on the X axis. They are numbered 0 thru 31.

DELTAX - Width of bar in eight pixel increments. A DELTAX of 2 will produce a bar 16 pixels wide. XSTART + DELTAX should be less than or equal to 31.

YSTART - Starting position of the bar on the Y axis. YSTART can be in the range of 0 thru 191 where Y = 0 is at the bottom of the screen.

DELTAY - Length of the bar on the Y axis. DELTAY is expressed in one pixel increments. To avoid drawing off the top of the screen, YSTART + DELTAY should be less than or equal to 191.

COLOR -- Color of the bar - 0 thru 15.

GBAR draws the colored bars by changing the colors which are displayed on the background plane. This routine should only be used when the VDP is initialized in the G2 mode by using GINIT. Because the bars are drawn by changing the colors on the background plane there are a maximum of thirtytwo bars (0 thru 31) which can be drawn on the X axis. Vertical bar 0 corresponds to X axis pixels 0-7, bar 1 corresponds to X axis pixels 8-15, etc. The height of the bar is controllable in one pixel increments. Thus, the use of vertical bars is most appropriate when there are fairly few - 32 or less - bars but their height needs to be finely controlled down to one pixel increments. To draw a dark red vertical bar 8 pixels wide at X axis position 16 and which starts at the bottom of the screen and goes up 96 pixels use the call:

```
GBAR(16, 1, 0 96, DRED);
```

The bars do not have to start at the bottom of the screen. If you want to leave room for annotations or a message at the bottom of the screen then use a non-zero value, such as 12, for YSTART. When doing this, be sure to add the YSTART value to the DELTAY value or your bars will all be too short by a YSTART amount.

GBAR can also be used to draw horizontal bars. There can be up to 192 (0 thru 191) horizontal bars. They would, however, only be one pixel wide/high. Thus, you will usually want to use less bars and make them several pixels wide so they can be more easily seen. Note that when drawing horizontal bars it is the DELTAX parameter which controls the length of the bar. The length increases in eight pixel increments. Thus, the length of a bar will range from 0 thru 31 increments. To leave some space for annotations on the left side of the screen use a XSTART of one (1) or more depending on the amount of space you need. To draw a dark red horizontal bar half way across the middle of the screen use the call:

```
GBAR(0,16, 96, 3 DRED);
```

Note that, since the bars are drawn by changing the colors in the background plane, you can use the DRAW, PLOT, and GTYPE procedures to draw, plot, and write over the bars. The bars can also be redrawn without changing anything which was drawn, plotted, or written over them. The following program fragment will draw vertical bars eight pixels wide which slope up to the top of the right side of the screen. It will then fill the rest of the screen with four pixel wide horizontal bars. The horizontal bars will have a four pixel space between them.

```
FOR X := 1 TO 31 DO
  GBAR(X, 1, 0, X*6, X);    (DRAW VERTICAL BARS)
FOR Y := 31 DOWNTO 1 DO
  GBAR(1, Y, Y*6, 4, Y);    (DRAW HORIZONTAL BARS)
```

Because the COLOR parameter uses the loop control variable the bars will walk thru the graphics color range.

GRAPHICS TEXT AND SYMBOLS

PASCAL G_TYPE(SIZE, ORIENTATION, S_LENGTH, G_STRING);

FORTRAN CALL GTYPE(SIZE, ORIENTATION, SLENGTH, GSTRING)

C80 gtype(size, orient, sleng, gstr);

MBASIC CALL GTYPE(SIZE%, ORIENTATION%, SLENGTH%, GSTRING%);

COBOL CALL "GTYPE" USING SIZE, ORIENTATION, SLENGTH, GSTRING.

SIZE --------- Is an integer value in the range of 1 thru 8. A size of 1 produces the minimum sized character in a 7 by 9 pixel area with a blank column of pixels on the left of the character. Lower case letters with desenders are shifted down three pixels and thus, in effect, use a 7 by 12 font. To avoid character overlap, rows of size 1 text which use lower case letters need a row spacing of at least thirteen pixels. Size 2 letters use 14 by 18 pixels, size 3 use 21 by 28 pixels, etc.

ORIENTATION - 0 = 0 degrees (normal left to right)
1 = 90 degree rotation
2 = 180 degree rotation
3 = 270 degree rotation

LENGTH ------ This is the number of characters passed in the string. This includes any special characters included so as to use the special symbols, greek letters, or user defined letters or symbols. These special characters are discussed below. See the description of the FONT procedure for defining and using your own fonts.

STRING ------ This is the actual string of characters, plus any special characters discussed below, which are to be displayed on the graphics monitor.

The procedure GTYPE makes it fairly easy to write the ASCII character set, special symbols, and the greek alphabet on your graphics monitor. When you call GTYPE it causes the linker to also include in your program the font table which contains the bit patterns which define the above characters and symbols. Printing the normal printable characters and special symbols is

done by simply including them in the G_STRING argument. To get to the greek alphabet and the other special symbols in the font table requires the use of a special control character. GTYPE uses the tilde (~) as the control character to shift GTYPE into the first 32 (0 thru 31) patterns in the font table. Since the tilde has this special use you must use two tilds (~~) to cause the tilde to be printed. The programs FONT8 (for H8's) and FONT89 (for H89's) on the distribution disks will display all the characters and symbols in the standard GRAPH-PAC-II font table. It displays them in the order in which they are stored in the table. Source code for these programs is in FONT.C and shows the use of GTYPE to write on the monitor.

Using size 1 letters allows up to 28 letters or symbols on a line. Clearly, the larger the letters you use the fewer you can put on a given line. If you try to write too many characters on a line GTYPE will move back to the beginning of that line and keep on writing until all characters in the string are displayed. Although writing a string which is too long will cause a messed up display it will not cause the program to crash.

The MOVEXY procedure is used to position the text string on the screen. Thus, the position of the letters and symbols can be controlled on a pixel by pixel basis anywhere on the screen. The MOVEXY and GTYPE procedures update the same internal GRAPH-PAC-II variables as do PLOT and DRAW. These internal variables keep track of the last X,Y coordinates which were used. Thus, if you draw a line and then call GTYPE the first character in the string will be located at the end of that line. Similarly, if a DRAW command follows a GTYPE command then the line will be drawn from the end of the string to the X,Y specified in the DRAW command. Multiple calls to GTYPE do not require computing where the previous string ended as GRAPH-PAC-II keeps track of it for you. For example:

```
MOVEXY(5,10);
GTYPE(1,0,13,"GRAPH-PAC-II ");
GTYPE(1,0,9,"Standard ");
GTYPE(1,0,5,"Fonts");
```

will write

GRAPH-PAC-II Standard Fonts

in the lower left portion of the screen using size 1 letters in the normal left to right manner (0 degrees of rotation).

In addition to the characters and symbols described above, GTYPE can display any character sets which have been defined by the programmer using the FONT procedure which is described below. If these characters are defined to be above the standard characters in the font table, font table positions 128 thru 255, then the veritcal bar (|) must be used to add 128 (80 hex) to the character which follows it in the STRING. Because of this

use of the vertical bar it, like the tilde, must be preceded by a tilde to have it printed on the graphics monitor. Both the tilde and vertical bar must be included in determining the length of the string which is passed to GTYPE. They will, however, not take space on the monitor as GTYPE will remove them from the displayed string. You will see this when you compare the code in FONT.C and the video display generated by FONT8 or FONT89.

Each computer language usually has one or more characters which must be handled in a special way when they are included in a string. The character used to mark the beginning and end of the string is usually one of them. Otherwise, that character could not be included within the string. For example, in Pascal if an apostrophe is to be printed within a string then two of them must be used so that the compiler will know that you want to print an apostrophe instead of terminating the string. With the C language the backslash (\) must immediatly preceed a backslash, an open quote (`), or a double quote (") if such characters are to be printed. Such characters, when used for this purpose, must be included in the strings passed to GTYPE. They must not be included in the string length value passed to GTYPE as the compiler will remove them from the string passed to by GTYPE.

To summerize the above - when a tilde (~) or vertical bar (|) is used as a special control character it IS counted as part of the string length. When a quote (`), backslash (\) or other character is used because your programming language requires it as a control character then it is NOT counted as part of the string length.

The following table lists the character sequences used for the first thirty two entries in the font table and the corresponding character or symbol which will be displayed on the graphics monitor. The remaining part of the font table is the standard ASCII character set.

GRAPH-PAC-II STANDARD FONTS

Mnemonic	Description	Mnemonic	Description
^@	Up arrow	^P	pi
^A	alpha	^Q	rho
^B	beta	^R	sigma
^C	gamma	^S	tau
^D	delta	^T	upsilon
^E	epsilon	^U	phi
^F	zeta	^V	chi
^G	eta	^W	psi
^H	theta	^X	omega
^I	iota	^Y	OMEGA
^J	kappa	^Z	Right arrow
^K	lambda	^[	Left arrow
^L	mu	^\	Up arrow
^M	nu	^]	Divide sign
^O	omicron	^^	Approximatly = or double tilde
^|	Vertical bar	^~	Tilde

USER GENERATED FONTS

```
PASCAL   FONT(FTN, TOP_LINE, ... , BOTTOM_LINE);

FORTRAN  CALL FONT(FTN, TOPLINE, ... , BOTTOMLINE)

C80      font(ftn, topline, ... , bottomline);

MBASIC   CALL FONT(FTN%, TOPLINE%, ... , BOTTOMLINE%)

COBOL    CALL "FONT" USING FTN, TOPLINE, ... , BOTTOMLINE.
```

FTN --------- Format table number. This integer identifies which one of the 256 format table entries is to be defined. Table entries are numbered 0 thru 255. 0 thru 127 contain the standard fonts described above. Entries 128 thru 255 are set aside for user definitions. FONT can just as easily be used to redefine the standard ASCII characters totally or selectively as desired. Each format table entry uses nine bytes to store each symbol or character.

TOP_LINE, ...
BOTTOM_LINE - These nine values/parameters define the pixels which are to be turned on to define the character or symbol. The first value is the topline and the last value is the bottom line. The high order (left most) bit is normally 0 as that is the inter-character space position. Characters which have desenders have the high order bit of the last parameter/value turned on. This causes GTYPE to shift the 7 by 9 pattern down three positions.

The following example shows the generation of the upper and lower case letters J and j.

Bit Pattern	Decimal	Bit Pattern	Decimal
00011111	31	00000010	2
00000100	4	00000010	2
00000100	4	00000010	2
00000100	4	00000010	2
00000100	4	00000010	2
00000100	4	00000010	2
00000100	4	00000010	2
01000100	68	00000010	2
00111000	56	10111100	188

Note that the high order bit of the bottom row of bits for the lower case "j" is "on" to tell GTYPE that this is a desender character. This bit will not be displayed. The FONT calls for these letters are as shown below.

```
J = FTN of 74 -- FONT(74,31,4,4,4,4,4,4,4,68,56);
j = FTN of 106 - FONT(106,2,2,2,2,2,2,2,2,188);
```

The font tables use nine bytes of memory for each character. Therefore, to conserve memory, the GRAPH-PAC-II library font table contains only the standard fonts (first 127 characters). FONT can be used to redefine any or all of these characters. A seperate font table CHRTB (REL or ERL) is provided for those applications which require over 127 characters and symbols. Since this table will store twice the characters it uses twice the memory. To use this larger font table simply include H8CHRTB or H89CHRTB before the GRAPH-PAC-II library in your link command line or the MBASIC module name (.DAT) file. This will cause the linker or BASLOAD to include the expanded font table in your program and skip over the standard one in the graphics library. Remember that, regardless of which font table you use, FONT only changes the characters in the program in which it is used. Using FONT will not change the characters and symbols stored in the GRAPH-PAC-II libraries. This allows each program to have whatever fonts it needs without impacting the fonts used in other programs. It also insures that the code you get from other people will produce the same results for you as it did for the original author.

SPRITE SUPPORT

The sprite manipulation routines offer facilities for sprite definition and movement. The VDP supports the simultaneous display of 32 sprites, or pattern objects. The addressing capabilities of the VDP allow the definition of up to 256 separate sprite patterns (64 if SIZE=1 in the VDP initialization call).

Sprites should be initially created by defining their bit patterns with the procedures SPAT8 (8x8 sprites) or SPAT8 and SPAT16 (16x16 sprites). D_8_SPR or D_16_SPR is then called to load the bit patterns into the sprite table and define the other sprite properties such as initial color and location. Thus, defining an 8x8 sprite uses the call:

```
SPAT8( arguments );
```

16x16 sprites are defined by the calls:

```
SPAT8( arguments );
SPAT16( arguments );
```

The actual arguments for these calls are described on the following pages. These procedures allow the sprite position, color, and pattern to be defined in a logical manner. The D_8_SPR and D_16_SPR procedures associate sprite pattern #N to logical sprite #N. Additional patterns which are not connected to a specific set of sprite attributes can be created with the sprite definition procedures. An effectively instantaneous sprite pattern change can then be made with the ASG_SPR procedure. This is the main technique that is used to perform animation. For additional details on how the graphics controller internally handles sprite definition and movement refer to the VDP manual, pages 25-29.

The sprite handling routines use the same X,Y coordinate system (0,0 at the lower left corner of the display screen) as the line and circle drawing routines. As explained on page 27 of the VDP manual, the sprite X,Y coordinate values refer to the upper left corner of the sprite. Thus, the sprite will smoothly slide on/off the screen as the 'Y' values move thru the ranges of 8/16 to 0 and 192 to 221. Similiarly, the sprite will slide off the the screen as the 'X' coordinate approaches 255 or 0 if the 'early bit' is turned on (refer to VDP manual page 27).

Note that the manner in which sprites are defined in GRAPH-PAC-II is changed somewhat form that used in GRAPH-PAC-II. The current method is the result of requests and suggestions of GRAPH-PAC-I users.

DEFINE SPRITE BIT PATTERNS

DEFINE FIRST EIGHT ROWS OF A SPRITE

```
PASCAL   S_PAT8( 8 PATTERN VALUES );
FORTRAN  SPAT8( 8 PATTERN VALUES )
C80      spat8( 8 pattern values );
MBASIC   CALL SPAT8( 8 PATTERN VALUES%)
COBOL    CALL "SPAT8" USING 8 PATTERN VALUES.
```

DEFINE SECOND EIGHT ROWS OF A SPRITE

```
PASCAL   S_PAT16( 8 PATTERN VALUES );
FORTRAN  SPAT16( 8 PATTERN VALUES )
C80      spat16( 8 pattern values );
MBASIC   CALL SPAT16( 8 PATTERN VALUES% )
COBOL    CALL "SPAT16" USING 8 PATTERN VALUES.
```

PATTERN VALUES - These are eight integers which represent the bit patterns of each row of the sprite. The first value is the top row and the last value is the bottom row. If an 8x8 sprite is being defined then only the low order eight bits of the integer are used. When 16x16 sprites are being defined all sixteen bits of the integer are used. Thus, negative values will often be used when defining 16x16 sprites.

DEFINE SPRITES

DEFINE 8 X 8 SPRITES

```
PASCAL   D_8_SPR(SPRITE_NUMBER, X, Y, COLOR, EARLY_BIT);
FORTRAN  CALL D8SPR(SPRNUM, X, Y, COLOR, EBIT);
C80      d8spr(sprnum, x, y, color, ebit);
MBASIC   CALL D8SPR(SPRNUM%, X%, Y%, COLOR%, EBIT%)
COBOL    CALL "D-8-SPR" USING SPR-NUM, X, Y, COLOR, EBIT.
```

DEFINE 16 X 16 SPRITES

```
PASCAL   D_16_SPR(SPRITE_NUMBER, X, Y, COLOR, EARLY_BIT);
FORTRAN  CALL D16SPR(SPRNUM, X, Y, COLOR, EBIT)
C80      d16spr(sprnum, x, y, color, ebit);
MBASIC   CALL D16SPR(SPRNUM%, X%, Y%, COLOR%, EBIT%);
COBOL    CALL "D-16-SPR" USING SPR-NUM, X, Y, COLOR, EBIT.
```

```
SPRITE_NUMBER -- Sprite number IN [0..31]
X ------------- Sprite horizontal position
Y ------------- Sprite vertical position
COLOR ---------- Color for ON bits in the sprite definition
EARLY_BIT ------ 0 = sprite position controlled by X and Y
                 1 = 32 is subtracted from the displayed X
                 position
```

These procedures define or redefine sprites. Calls to GINIT, PINIT, and MINIT set up the sprite tables so as to disable all sprites. Because of the manner in which this is done, it is important that all sprites defined be consecutively numbered starting with '0'. Failure to do this will give unexpected results. When this support package is used, the VDP internal address adjustments mentioned on page 28 of the VDP manual are taken into account.

Sprite numbers in the range of 0..31 may be used. Sprite 0 has the highest priority on the video screen. Higher numbered sprites can be "hidden" behind lower numbered sprites. The VDP hardware takes care of this automatically based on the X,Y coordinates used to position the sprites.

Refer to page 27 of the VDP manual for additional explanation of the position and EARLY_BIT arguments in the sprite attribute table.

The pattern for the sprite will be placed in the sprite generator table with the same "name" as the sprite number; i.e., sprite #1 will be connected to sprite pattern #1.

For SIZE=0 sprites (8x8 pixels) the sprite pattern is defined in the sprite generator table with 8 bytes. In this case, only the least significant bytes of the integers are used in building the sprite patterns. When SIZE=1 (16x16 pixel sprites) both bytes of the 16 integers are used to build the sprite pattern. The sprite pattern consists of 16 rows of 16 columns (or bits) each. The first integer passed is the first row of the pattern, the second integer is the second row, continuing thru all 16 rows. When working with or defining 16x16 sprites the all-base calculator program available from the Heath Users' Group is most handy.

POSITION SPRITE

```
PASCAL   POS_SPR(SPRITE_NUMBER, X, Y);

FORTRAN  CALL POSSPR(SPRNUMB, X, Y)

C80      possprr(sprnumb, x, y);

MBASIC   CALL POSSPR(SPRNUMB%, X%, Y%)

COBOL    CALL "POSSPR" USING SPRITE-NUMBER, X, Y.
```

SPRITE_NUMBER - Sprite number IN [0..31]

X ------------ New horizontal position

Y ------------ New vertical position

This procedure is used to change the position of a previously defined sprite. Refer to VDP manual pages 26-27 for additional data on sprite movement.

EARLY BIT CONTROL

```
PASCAL   E_BIT(SPRITE_NUMBER, EARLY);

FORTRAN  CALL EBIT(SPRNUMB, EARLY)

C80      ebit(sprnumb, x, y);

MBASIC   CALL EBIT(SPRNUMB%, EARLY%)

COBOL    CALL "EBIT" USING SPRITE-NUMBER, EARLY.
```

SPRITE_NUMBER - Sprite number IN [0..31]

EARLY --------- 0 - Sprite position controlled by previously defined X and Y positions

1 - Subtracts 32 from X position value

This procedure changes the EARLY_BIT in the sprite attribute table. See VDP manual page 27.

SPRITE PATTERN CREATION

DEFINE 8 X 8 PATTERN

```
PASCAL   D_8_PAT(PATTERN_NUMBER);

FORTRAN  CALL D8PAT(PATNUM)

C80      d8pat(sprnumb);

MBASIC   CALL D8PAT(SPRNUMB%)

COBOL    CALL "D8PAT" USING SPRITE-NUMBER.
```

DEFINE 16 X 16 PATTERN

```
PASCAL   D_16_PAT(PATTERN_NUMBER);

FORTRAN  CALL D16PAT(SPRNUMB)

C80      d16pat(sprnumb);

MBASIC   CALL D16PAT(SPRNUMB%)

COBOL    CALL "D16PAT" USING SPRITE-NUMBER.
```

PATTERN_NUMBER - 0 thru 255 if SIZE = 0 (8x8 pixel sprites)
- 0 thru 63 if SIZE = 1 (16x16 pixel sprites)

These procedures are used to load a new sprite pattern, or to update an existing pattern. The method of loading the PATTERN VALUES is the same as discussed under the S_PAT8 and S_PAT16 procedures. The D_8_PAT and D_16_PAT procedures do not associate the pattern with entries in the sprite attribute table. See VDP manual pages 28 and 29 for additional information on the internal operation of graphics controller with respect to sprite handling. To define a new 8 x 8 pattern use:

```
SPAT8( arguments );
D8PAT( argument );
```

A new 16 x 16 pattern is defined with

```
SPAT8( arguments );
SPAT16( argruments );
D16PAT( argument );
```

SPRITE PATTERN ASSIGNMENT

PASCAL ASG_PAT(SPRITE_NUMBER, PAT_NUMBER);

FORTRAN CALL ASGPAT(SPRNUMB, PATNUM)

C80 asgpat(sprnumb, patnum);

MBASIC CALL ASGPAT(SPRNUMB%, PATNUM%)

COBOL CALL "ASGPAT" USING SPRITE-NUMBER, PATTERN-NUMBER.

SPRITE_NUMBER -- 0 thru 31

PATTERN_NUMBER - 0 thru 255 if SIZE = 0 (8x8 sprites)
- 0 thru 63 if SIZE = 1 (16x16 sprites)

This procedure assigns a previously defined pattern to the attributes of the specified logical sprite. The original sprite pattern is not changed but simply disconected from its entries in the sprite attribute table. This procedure is provided to facilitate rapid changing of the patterns associated with sprite attributes so as to simulate animation. See VDP manual pages 26 and 27 for additional details on the sprite attribute table and the pattern tables.

SPRITE COLOR UPDATE

PASCAL S_COLOR(SPRITE_NUMBER, COLOR);

FORTRAN CALL SCOLOR(SPRNUMB, COLOR)

C80 scolor(sprnumb, color);

MBASIC CALL SCOLOR(SPRNUMB%, COLOR%)

COBOL CALL "SCOLOR" USING SPRITE-NUMBER, COLOR.

SPRITE_NUMBER - 0 thru 31

COLOR - 0 thru 15. This value will control the color of the sprite ON bits.

This procedure is used to change the color of the sprite without changing its location or pattern/shape. See VDP manual page 26.

RESET SPRITE MAGNIFICATION STATE

```
PASCAL  MAG0;   Set the sprites to their defined size

FORTRAN CALL MAG0

C80     mag0();

MBASIC  CALL MAG0

COBOL   CALL "MAG0".


PASCAL   MAG1;   Set the sprites to twice their defined size.

FORTRAN CALL MAG1

C80     mag1();

MBASIC  CALL MAG1

COBOL   CALL "MAG1".
```

The MAG0 and MAG1 procedures permit the sprite magnification state to be reset during program execution without using one of the initialization routines described above. This allows the size/magnification of the sprites to be changed without disturbing the other aspects of the VDP display.

PROGRAMMABLE SOUND GENERATOR (PSG) SUPPORT

OVERVIEW

The following functions and procedures facilitate control of the PSG chip used on the color graphics boards. It is assumed that the reader is generally familiar with the operation of this chip from reading and studying the PSG hardware manual.

In the following function and procedure definitions, CHANNEL is a character parameter. CHANNEL is an A, B or C (upper case) corresponding to the PSG channel which is to be controlled. To reduce CPU loading, all data written to the PSG registers remains active and stable until a new value is written. All eight bits of the PSG parallel ports (registers 14 and 15) are data bits inasmuch as handshaking is not used or required. The P_OPTS procedure is used to set the ports to input or output under program control. The PSG chip provides internal pull-up resistors which keep all bits in a port configured for input, set high, unless an external device, such as a joystick switch, is closed. At such time the corresponding bit is brought low. As soon as the switch is released, the PSG chip returns the bit to a high condition. Although the port values follow the signals (switch closures, etc.) applied to the port, these values are available for use in the program only after a port READ operation is performed.

In a similar manner, when a port is in the output mode, the data will remain on the port until (1) new data are written to the port, (2) the port is switched to the input mode, or (3) the chip/system is reset. Refer to PSG manual pages 5, 8, 10, 11, 14, 17 and 28 for further details and diagrams.

Note that in the following procedures and functions, CHANNEL and CONTROL must be the upper case A, B, C, V, or F.

BASIC PRIMITIVES

DIRECT READ/WRITE OF PSG REGISTERS

For those who wish to interface directly with the PSG, the following function and procedure are provided. They simply read or write the specified register without providing any conversions. The one exception to simply reading or writing the register is when reading or writing the ENABLE register (register 7). This is the only register in which the bit patterns specified in the hardware manual must be inverted before being loaded into the register. Therefore, the WRITE_PSG_REGISTER (W_P_REG) procedure inverts the bits before writing to the ENABLE register. The READ_PSG_REGISTER (R_P_REG) function also complements the ENABLE register bits after reading them from PSG register 7.

WRITE PSG REGISTER/PORT

PASCAL W_P_REG(REGISTER, DATA);

FORTRAN CALL WPREG(REG, DATA)

C80 wpreg(reg, data);

MBASIC CALL WPREG(REGISTER%, DATA%)

COBOL CALL "WPREG" USING REGISTER, DATA.

MACRO-80 NOTE: The alternate entry point is W.REG, which expects the PSG register number to be in the accumulator and the DATA value to be in the E register.

READ PSG REGISTER/PORT

```
PASCAL   VALUE := R_P_REG(REGISTER);

FORTRAN VALUE =  RPREG(REG)

C80      value = rpreg(reg);

MBASIC   CALL RPREG(REGISTER%)

COBOL    CALL "RPREG" USING REGISTER.
```

MACRO-80 NOTE: The alternate entry point is R.REG which expects that the number of the PSG register to be read will be found in the accumulator. The value read from the specified PSG register is returned on the stack.

When using W_P_REG and R_P_REG described on the previous page, VALUE and DATA are integers in the range of 0 thru 255. REGISTER is is also and integer in the range of 0 thru 15. Since all PSG registers are eight bits wide, W_P_REG merely loads the eight least significant bits of data and discards the eight high order bits of the argument passed to it. Although the use of the PSG parallel ports A and B (registers 14 and 15) is considerably different from that of the other PSG registers, they are read and written in exactly the same manner. Program readability will be improved if the common definitions are used to equate PORTA to the numeric value "14" and PORTB to "15". When these common definitions or equates are used, the statement W_P_REG(PORTB, 108); would write the value "108" to PSG port 15. VALUE is an integer or byte and the function returns values in the range of 0 thru 255. If none of the switches connected to the port are depressed, the function returns a value of 255. If all of the switches are closed, it returns a value of 0. See PSG manual pages 7, 16-17, and 44-45.

NOTE: when using the following procedures, sound channel arguments are "A", "B", or "C" for the PSG channels A, B, C. An "F" or "V" is used to tell the procedures whether to use fixed or variable sound level control (see PSG manual page 24).

COMPOSITE PRIMITIVES

SET/RESET PSG OPTIONS

```
PASCAL  P_OPTS(ENA, ENB, ENC, ENNA, ENNB, ENNC, ENPAI, ENPBI);

FORTRAN CALL POPTS(ENA, ENB, ENC, ENNA, ENNC, ENPAI, ENPBI)

C80     popts(ena, enb, enna, ennb, ennc, enpai, enpbi);

MBASIC  CALL POPTS(ENA%,ENB%,ENC%,ENNA%,ENNB%,ENNC%,ENPAI%,ENPBI%)

COBOL   CALL "POPTS" USING ENA,ENB,ENC,ENNA,ENNB,ENNC,ENPAI,ENPBI.
```

The arguments are BOOLEAN variables or integer constants according to the following definitions where TRUE = 1 and FALSE = 0:

```
ENA ------ TRUE  Channel A enabled
           FALSE Channel A disabled

ENB ------ TRUE  Channel B enabled
           FALSE Channel B disabled

ENC ------ TRUE  Channel C enabled
           FALSE Channel C disabled

ENNA ----- TRUE  Noise enabled on channel A
           FALSE Noise disabled on channel A

ENNB ----- TRUE  Noise enabled on channel B
           FALSE Noise disabled on channel B

ENNC ----- TRUE  Noise enabled on channel C
           FALSE Noise disabled on channel C

ENPAI ---- TRUE  Port A enabled for input
           FALSE Port A enabled for output

ENPBI ---- TRUE  Port B enabled for input
           FALSE Port B enabled for output
```

See PSG manual page 21 for a complete description of the ENABLE register (7) loaded by this procedure.

SET TONE FREQUENCY

```
PASCAL   T_FREQ(CHANNEL, FREQUENCY);

FORTARN  CALL TFREQ(CHAN, FREQ)

C80      tfreq(chan, freq);

MBASIC   CALL TFREQ(CHANNEL%, FREQUENCY%)

COBOL    CALL "TFREQ" USING CHANNEL, FREQUENCY.


PASCAL   T_PER(CHANNEL, PERIOD);

FORTRAN  CALL TPER(CHAN, PERIOD)

C80      tper(chan, period);

MBASIC   CALL TPER(CHANNEL%, PERIOD%)

COBOL    CALL "TPER" USING CHANNEL, PERIOD.
```

FREQUENCY is an integer representing Hertz (cycles per second). The tone frequency can range from 27 Hz to 111,861 Hz although the highest part of this range is beyond the capability of most audio equipment. Satisfactory results from most equipment should be obtained if the frequencies are kept in the range of 40 Hz thru about 9000 Hz. In the T_PER procedure the period value is directly loaded into the frequency register of the specified channel. This is an alternate method of controling the frequency and is, at times, more convenient, depending on the nature of the program code/loop structures being used. See PSG manual pages 18-19.

SET CHANNEL AMPLITUDE

PASCAL C_AMP(CHANNEL, CONTROL, AMPLITUDE);

FORTRAN CALL CAMP(CHAN, CONT, AMP);

C80 camp(chan, cont, amp);

MBASIC CALL CAMP(CHANNEL%, CONTROL%, AMPLITUDE%)

COBOL CALL "CAMP" USING CHANNEL, CONTROL, AMPLITUDE.

AMPLITUDE is an integer in the range of 0 thru 15 were "0" is minimum and "15" is maximum volume. CONTROL is an "F" for fixed volume level and a "V" for variable level control. A channel can be turned off by sending it an amplitude value of "0" or by disabling it with the P_OPTS procedure. In the "V" mode the amplitude value is ignored since the PSG hardware electronics sweep the volume/amplitude of the sound thru the full range using the envelope shape set by the E_SHAPE procedure. See PSG manual pages 22-23.

SET NOISE GENERATOR FREQUENCY

PASCAL NOISE(FREQUENCY);

FORTRAN CALL NOISE(FREQ)

C80 noise(freq);

MBASIC CALL NOISE(FREQUENCY%)

COBOL CALL "NOISE" USING FREQUENCY.

FREQUENCY is an integer representing Hertz with a minimum value of 3609. See PSG manual page 20.

SET ENVELOPE CYCLE (DURATION) TIME

```
PASCAL  E_CYCLE(TIME);

FORTRAN CALL ECYCLE(TIME)

C80     ecycle(time);

MBASIC  CALL ECYCLE(TIME%)

COBOL   CALL "ECYCLE" USING TIME.
```

TIME is an integer representing tenths of seconds. This is the amount of time between repeating cycles of the envelope generator or the duration of a single tone/sound if envelope shapes 1, 3, 5, or 7 (see E_SHAPE) are used. Legal values for TIME are from 1 (.1 second) thru 93 (9.3 seconds). See PSG manual pages 24-25.

SET ENVELOPE SHAPE/CYCLE SHAPE

PASCAL E_SHAPE(SHAPE);

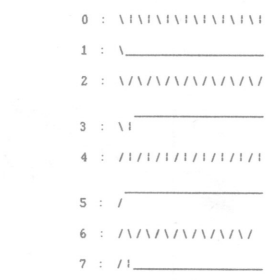
FORTRAN CALL ESHAPE(SHAPE)

C80 eshape(shape);

MBASIC CALL ESHAPE(SHAPE%)

COBOL CALL "ESHAPE" USING SHAPE.

SHAPE is an integer in the range of 0 thru 7. It is used to select one of the following eight shapes:



See pages 25 and 26 of the PSG manual for a complete description of the available envelope shapes.

PSG CONTROL CONSIDERATIONS

It should be noted here that certain timing considerations should be followed when attempting to program the PSG for transient effects, such as explosions. The action of the chip is such that the instant the envelope cycle time is defined through the E_CYCLE procedure, the envelope begins counting through its previously defined shape and cycle combination. Because of the finite speed of the computer language load module, it is important that things be done in the correct order to result in the anticipated sound. The shape of the sound envelope should be set immediatly after the cycle time is set. This is somewhat different than what is shown in the PSG manual. However, tests have shown this to produce more reliable results. For example, the following code extract demonstrates the technique for creating the sound of an explosion:

```
VAR
  I : INTEGER;

NOISE(4000);                    { SET UP 4000 HZ NOISE FREQUENCY }
P_OPTS(0,0,0,1,0,0,0,0);        { ENABLE NOISE ON CHANNEL A }
C_AMP('A', 'V', 15);            { CHAN 'A' = VARIABLE, VOLUME = max }
I := 10;                        { 10 = 1 SECOND IN E_CYCLE}
E_CYCLE(I);                     { DO IT NOW, 1 SECOND DURATION }
E_SHAPE(1);                     { USE AN EXPLOSION SHAPE }
PAUSE(I * 50);                  { 500 = 1 SECOND IN PAUSE}
```

Note that at least one second should elapse before another PSG sound control statement is used. This is so that the explosion sound can run thru the complete amplitude decay profile established by the above statements. Thus it is often necessary to include a 'pause' or 'wait' loop after the E_CYCLE statement. An easy way of keeping the E_CYCLE and PAUSE times in sync is to use a variable to express the the E_CYCLE duration time. The value of this variable can then be multiplied by fifty (50) to obtain the correct value for the PAUSE procedure. This technique is illustrated in the program fragment provided above.

ROUTINES USED ONLY BY THE HA-89-3

VOTRAX SC-01 SPEECH SYNTHESIZER

TEST VOTRAX SC-01

PASCAL PIDLE;

FORTRAN CALL PIDLE

C80 pidle();

MBASIC CALL PIDLE

COBOL CALL "PIDLE".

PIDLE is a boolean function which uses the programmable interrupt controller (PIC) to determine if the SC-01 is idle and therefore ready to receive another phoneme. Since the various phonemes require different amounts of time to be completed, the program generating speech must determine when the current phoneme has been completed. To simplify this task for the programmer the PIDLE function reads the pic interrupt request register, tests the PSS status bit and returns a TRUE (1) if the SC-01 is ready for another phoneme or a FALSE (0) if it is not.

SET SPEECH INFLECTION

PASCAL INFLEC(INFLECTION);

FORTRAN CALL INFLEC(INFLEC)

C80 inflec(inflec);

MBASIC CALL INFLEC(INFLECTION%)

COBOL CALL "INFLEC" USING INFLECTION.

In the procedure INFLEC, the values of 0 thru 7 are used to set the vocal inflection from the lowest to the highest pitch. This procedure has a built-in test using PIDLE to ensure that it does not output the inflection value until the SC-01 is ready for it.

OUTPUT A SPEECH PHONEME

PASCAL PHONEM(PHONEME);

FORTRAN CALL PHONEM(PHONEM)

C80 phonem(phonem);

MBASIC CALL PHONEM(PHONEME%)

COBOL CALL "PHONEM" USING PHONEME.

The PHONEM procedure is used to output the sixty-four phonemes listed on pages 3 and 4 of the PHONETIC SPEECH DICTIONARY for the SC-01 SPEECH SYNTHESIZER. The phoneme codes (0 thru 64 decimal, or 0 thru 3F hex) may be output by the program without regard to phoneme durations inasmuch as this procedure also uses the PIC to determine if the SC-01 is ready for another phoneme.

DIGITAL-TO-ANALOG CONVERSION (DAC)

DAC OUTPUT

PASCAL	DAC0(VALUE);	DAC1(VALUE);
FORTRAN	CALL DAC0(VALUE)	CALL DAC1(VALUE)
C80	dac0(value);	dac1(value);
MBASIC	CALL DAC0(VALUE%)	CALL DAC1(VALUE%)
COBOL	CAL "DAC0" USING VALUE.	CALL "DAC1" USING VALUE.

The DAC0 and DAC1 procedures are used to output digital values in the range of 0 thru 4095 to the two twelve-bit digital-to-analog converters. Although the values are passed as sixteen-bit arguments the DAC0 and DAC1 procedures only output the low order twelve bits the DACs. The rapidity with which new values can be sent to the DACs will depend on the response times of the analog devices attached to the DACs. Thus, the output timing requirements are application dependent and are the responsibility of the application programmer.

APPENDIX A

SETTING THE INPUT AND OUTPUT PORT ADDRESSES

One of the new features added to the graphics support package is the ability to configure your programs to talk to graphics boards using any port addresses. On each distribution disk there are two programs which allow you to do this. They are, for CP/M, SETCPM8.COM and SETCPM89.COM. For HDOS they are SETHOS8.ABS and SETHOS89.ABS.

One of the first things you should do after studying this manual is to run these programs. The purpose of running these programs is to generate special files which contain port addressing information which will be used by the linker to resolve symbolic port addresses which are contained in many of the graphics routines. By using this approach you have the ability to compile your programs such that they can be run at any port addresses. In most situations the defaults built into the programs will be the ones you will want to use as they are the standard port addresses used by the HA-8-3 and HA-89-3 color graphics boards. Once you have run these programs you should not have to run them again unless you change the port addresses of your graphics board, which is not recommended, or you want to prepare programs for someone who has their baord at different addresses.

To prepare the necessary input and output files follow these steps.

1. Copy the above mentioned programs to a working disk and then remove the distribution disk.

2. Run SETCPM8.COM or SETHOS8.ABS as appropriate. The program will issue the following messages:

Recommended port address is 184 (270 OCTAL).

Enter base port address in decimal (184):

If you need to use other than the standard port addresses then enter the base address you are using. Otherwise simply depress the RETURN key. When the program finishes you will have a new file called IOADDR.REL on drive "A" or "SY0:" Therefore make sure that you have a few K of free space on that drive.

3. Rename this file to IOH8.REL or IOH8.ERL if you are using Pascal MT+.

4. Run SETCPM89.COM or SETHOS89.ABS as appropriate. The program will issue the following messages:

Recommended port address is 208 (320 octal).

Enter base port address in decimal (208):

As above, enter the port address you are using or simply RETURN if you are using the standard configuration.

5. Rename this file to IOH89.REL or IOH89.ERL if you are using Pascal MT+.

Well, that's all there is to it. Exactly how you will use these files is explained in the language supplement which is included with this manual. It might be a good idea to make some backup copies of these files on some of your working disks.

APPENDIX B

Rewiring the JVC-40 Joy Stick
To Work With The Heath HA-8-3/HA-89-3 Graphics Boards

The following steps are specifically related to the HA-8-3 model of the graphics board. The internal joystick connections/rewiring steps are identical for both the H8 and H89 models. Read the note at the bottom of the page before doing the rewiring with the standard 40K pots.

1. Clip off the white connector at the end of the cable.

2. For use with the H8 model, solder Heath spring clips onto each of the five wires and install them into a 10 position socket (such as are used with the 4-port serial board) as shown at the bottom of the diagram. For use with the H89 model, simply tin the ends of the wires.

3. Carefully remove the bottom of the joy stick housing. This can be done by squeezing the sides of the bottom and pulling the two halves apart.

4. Disconnect the blue wire from the push button switch and connect it to lug 1 of the Y-axis pot.

5. Disconnect the yellow wire from lug 3 of the X-axis pot and connect it to lug 2 of the X-axis pot. As you do this, remove the red wire which is connected to lug 2 of X-axis pot. Now disconnect the other end of this red wire from lug 2 of the Y-axis pot.

6. Connect the red wire removed in Step 5 to Y-axis lug 1 and X-axis lug 1.

7. Connect a jumper from X-axis lug 3 to lug 1 on the push button.

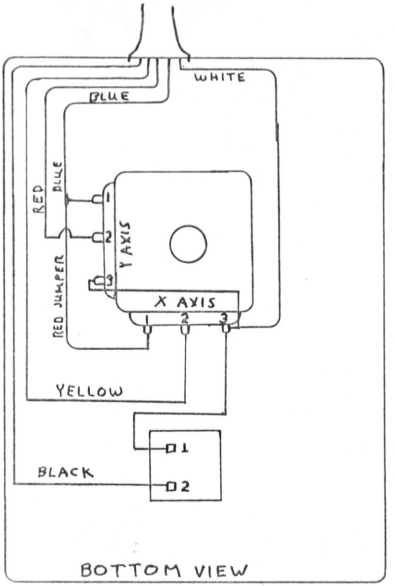
Connect a jumper from X-axis lug 3 to Y-axis lug 3.

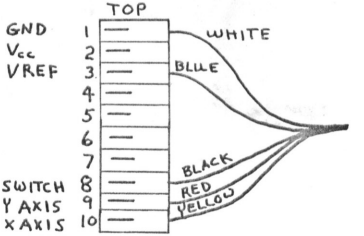
9. After carefully positioning the joy stick, cable stress relief, and internal wires, snap on the bottom of the case. A few simple resistance checks will verify the correctness of your wiring.

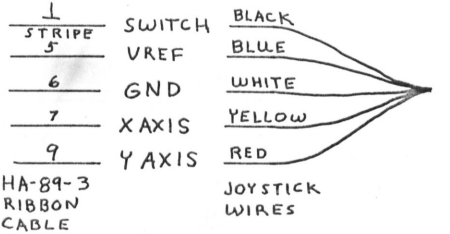
10. For the H8 model, slide the spring clips into the housing as shown in the diagram. For the H89, connect the joystick wires to the flat ribbon cable as shown in the diagram.

NOTE: Using the pots which are standard in these joysticks will result in the sprites moving over only the center 2/3 to 3/4 of the screen. This is because the motion of the stick limits the travel of the wipers. This is normal and can be compensated for in your software. The JAMCO 100K pots appear to go the whole resistance range. These pots are exact physical replacements

for the 40K pots which come with the joysticks. The connections described above put the origin of the screen (X=0, Y=0) in the lower left corner. Reversing the wires on Y-axis lugs 1 and 3 will put the origin at the upper left corner. The software in this support package assumes the joy sticks are wired as outlined above.


WHITE
BLUE
RED
BLUE
1
2
3
Y AXIS
RED JUMPER
X AXIS
1
2
3
YELLOW
1
2
BLACK
BOTTOM VIEW


TOP
GND 1
Vcc 2
VREF 3
4
5
6
7
SWITCH 8
Y AXIS 9
X AXIS 10
WHITE
BLUE
BLACK
RED
YELLOW
HA-8-3


1
STRIPE
5
6
7
9
SWITCH
VREF
GND
X AXIS
Y AXIS
HA-89-3
RIBBON
CABLE
BLACK
BLUE
WHITE
YELLOW
RED
JOYSTICK
WIRES

```
W_P_REG, 46
X, 16
X coordinate, 20, 21, 23, 36, 38, 40
Y, 16
Y coordinate, 20, 21, 23, 36, 38, 40
Z80, 1, 2
```