A MICROCOMPUTER BASED PLASMA DISPLAY SYSTEM.

Ordale Paul Babin, Jr.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROCOMPUTER BASED PLASMA DISPLAY SYSTEM

by

Ordale Paul Babin, Jr.

and

Ronald Ray Seaman

March 1978

Thesis Advisor: Uno R. Kodres

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microcomputer Based Plasma Display System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1978
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ordale Paul Babin, Jr., LT, USN Ronald Ray Seaman, CAPT, USMC		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1978
		13. NUMBER OF PAGES 300
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, CA 93940		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interface
Plasma
Microcomputer

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An overview of plasma display technology and operation is presented. Advantages and disadvantages of plasma graphics are explored. Some applications that are particularly appropriate for a plasma display

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601 |

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

are listed. Hardware and software developed to interface the AN/UYQ-10 plasma display with an Intellec Microcomputer Development System are discussed.

A MICROCOMPUTER BASED PLASMA DISPLAY SYSTEM

by

Ordale Paul Babin, Jr.
Lieutenant, United States Navy
B.S. Applied Physics, Louisiana Polytechnic University, 1970

and

Ronald Ray Seaman
Captain, United States Marine Corps
B.S. Mech. Eng., University of New Mexico, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
March 1978

ABSTRACT

An overview of plasma display technology and operation is presented. Advantages and disadvantages of plasma graphics are explored. Some applications that are particularly appropriate for a plasma display are listed. Hardware and software developed to interface the AN/UYQ-10 plasma display with an Intellec Microcomputer Development System are discussed.

CONTENTS

LIST OF TABLES

I. INTRODUCTION

Graphic data processing provides a common language of graphics and alphanumerics between man and the computer. A man thinks in terms of sketches, drawings, graphs, letters, characters and numbers. However, the computer relates to bits, bytes and registers. With a graphics display interface, man can spend more time defining a problem in terms he understands best [Pellerin 1977].

Display designers have for years been searching for an alternative to the cathode ray tube (CRT) - for a device that retains the image indefinitely without loss of quality, that can be accommodated in a small space, and that requires only low voltage power supplies. No such device has yet been discovered that retains the high performance and quality of the CRT. One promising alternative to the CRT is the plasma panel. It meets almost all the display designer's needs [Benwill Staff Report 1978].

Included in this thesis is an overview of plasma display technology and plasma display operation. Advantages and disadvantages of plasma graphics are explored. Some applications are listed which utilize the plasma's advantages and appear to be particularly appropriate. The remainder of the thesis discusses hardware and software developed while interfacing the AN/UYQ-10 with the Intellec Microcomputer Development System. Details of the hardware interface are

listed in Appendix A and a brief operations manual for the software developed is included as Appendix B.

II. OVERVIEW

A. DESCRIPTION OF TECHNOLOGIES

Most graphic display systems use refresh or storage technology. Three main types of refresh technologies exist: stroke writing, raster scanning and scan converting. Stroke writing display systems position an electron beam on the tube face much as one would draw on paper with a pencil. In raster scanning systems, the beam sequentially traces the entire face of the tube. When the beam arrives at a point that belongs to the picture under construction, a video signal "brightens" the beam to illuminate the screen. Hybrid scan converters use a storage tube to store the image and then scan the storage tube information onto a raster scanning monitor to display the image. Since the persistence of the phosphor in the tube is low, CRT's using one of these technologies require periodic image refreshing to prevent annoying screen flicker. These CRT's refresh the image at least thirty times each second.

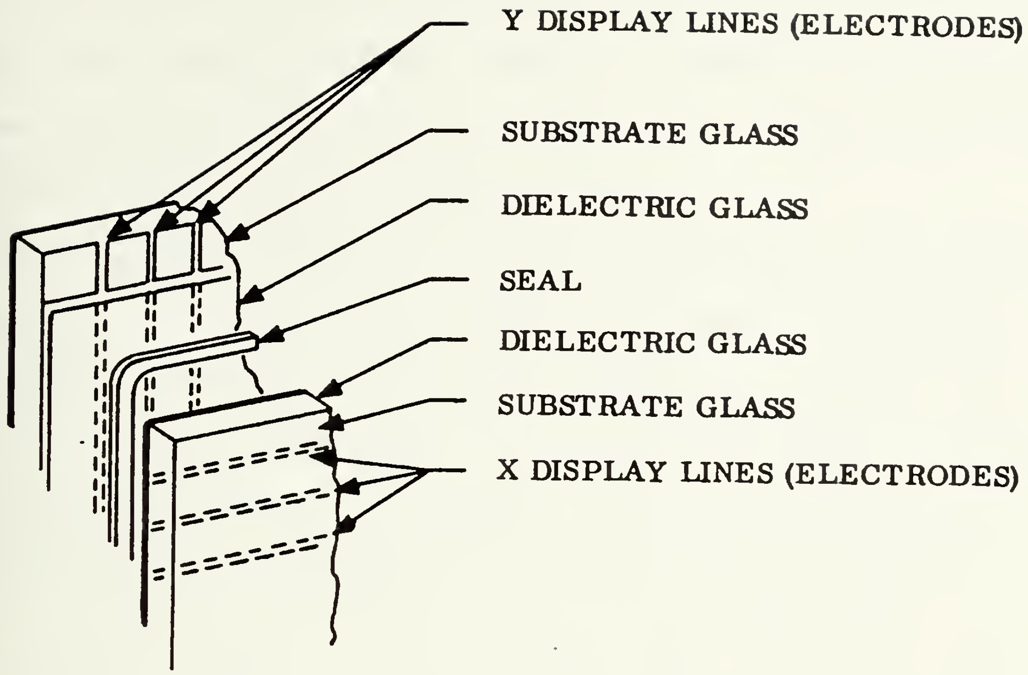
Two storage technologies exist: the storage tube and the plasma panel. With the storage tube, the CRT receives its image in the same way as a stroke writing system.[1] However, the storage tube stores the image on a grid, eliminating periodic refresh. Unlike other graphic display systems, plasma panels do not use CRT's. They substitute etched glass plates separated by a gas which glows when excited by

an electric pulse. The display consists of a series of bright dots that can be formatted into alphanumerics and graphics. Plasma panels do not require refresh and, once a particular point on the display in "turned on," it continues to glow until "turned off" [Pellerin 1977].

B. PLASMA PANEL OPERATION

The construction of a plasma panel is shown in Figure 1. The panel consists of two sheets of glass with thin, closely spaced electrodes attached to the inner faces and covered with a dielectric material. The two sheets of glass are spaced a few thousandths of an inch apart, and the intervening space is filled with a neon-based gas and sealed. By applying voltages between the electrodes, the gas panel is made to behave as if it were divided into tiny cells, each one independent of its neighbors. A cell is made to glow by placing a "firing" voltage across it by means of the electrodes. The gas within the cell begins to discharge, and this developes very rapidly into a glow. The glow can be sustained by maintaining a high frequency alternating voltage across the cell. If the signal amplitude is chosen correctly, cells that have not been "fired" will not be effected, i.e., each cell is bistable.

Cells can be switched on by momentarily increasing the sustaining voltage; this can be done selectively by modifying the signal in the two conductors that intersect at the desired cell. Conversely, if the sustaining voltage is lowered, the glow is removed [Newman 1973].



PLASMA PANEL CONSTRUCTION

Figure 1

C. TECHNICAL ADVANTAGES/DISADVANTAGES OF PLASMA DISPLAYS

The simplicity of construction of the plasma panel suggests that it it can replace the CRT for many computer graphics applications. In the present state of development it compares favorably with the direct-view storage tube.

The main advantage of plasmas over storage tubes is selective erase. The storage tube must erase the entire image and rewrite the modified image - a time consuming process. The plasma display also presents a sharper image that does not deteriorate with time, its power requirements are less stringent, it has a longer life, and it occupies less space. Some disadvantages in comparison to storage tubes are lower resolution and no "real" gray scale. At the

current state of the art, plasma displays use reduced resolution to simulate gray scales. A common resolution for plasma displays is sixty points per inch, which is about half the resolution of a storage tube.

The main advantages of plasma technology over refresh technology is elimination of the requirement to refresh and flatness of the display. The requirement to refresh may be partially offset because of a need for a separate display processor. Raster scan systems offer the advantages of color and higher resolution with the disadvantage of higher memory requirements. Stroke writers offer the advantages of high resolution, limited color and dynamic motion with the disadvantage of limited display (increased amounts of data cause the screen to flicker) [Pellerin 1977].

Plasma technology offers some additional advantages over CRT technology. Plasmas have no pin-cushion and barrel distortion, no focus distortion, no delicately aligned internal components and no digital to analog conversion. Plasmas also have the advantages of high endpoint accuracy, uniform brightness, high screen life and ruggedness.

D. COST COMPARISON OF GRAPHIC SYSTEMS

Storage tube systems by Tektronix range from $4,000 to $15,000. A basic plasma display from Magnavox sells for about $6,000, while raster scan and low cost stroke writers sell from $10,000 to $50,000. Sophisticated stroke writers with the ability to represent three dimensional graphics range in price from $30,000 to $85,000. Further, plasma

displays should decrease in price as development costs are recouped [Pellerin 1977].

E. CURRENT STATUS OF PLASMA DISPLAYS

Common plasma panels consist of 512 by 512 elements with 60 elements per inch, which provides an active viewing surface of about 8.5 inches square. Options such as "touch entry", "rear projection" and "special" gray scales are available. Usable life is advertised to be approximately 10,000 active display hours or about five years.

F. PROBABLE FUTURE

The panel's resolution should be expected to double in the near future. Limited color capability and large scale displays (in excess of three feet square) are theoretically possible. With improved resolution, plasma displays will replace more expensive graphics devices.

Attention has been drawm to the possible effects of radiation hazards from CRT's. Plasma displays operate on different principles and are free from radiation hazards.

G. PROBABLE APPLICATIONS

The plasma display offers advantages in general displays and text editing since it does not need refreshing and since it holds sixty percent more text than most CRT's. In the graphics area, the plasma display offers compactness and durability over storage tube systems. In applications which require ruggedized CRT's or oversized displays, plasma panels are particularly appropriate.

III. DESCRIPTION OF HARDWARE

This chapter presents the equipment that was the basis of this project - the INTELLEC Microcomputer Developement System (MDS) manufactured by INTEL, the Plasma Display Set (AN UYQ-10) manufactured by SCIENCE APPLICATIONS INC. (SAI) and the hardware interface between the MDS and the Plasma Display Set.

The electronic hardware necessary to interface the MDS with the Plasma Display, which was developed and constructed, has been described in detail in Appendix A. This interface was necessary to match the twisted-pair signals of the Plasma Display to the TTL signals of the MDS.

A. MDS INPUT/OUTPUT

The I/O Module of the MDS includes four input and four output ports. Each output port latches eight-bit data words and each input port supports eight bits of data, latched or unlatched. Two input ports were used - one for eight data bits and one for four control bits. Two output ports were used - one for eight data bits and one for two status bits plus two plasma control bits. The two plasma control lines are CONTROL A and CONTROL B. When high, CONTROL A disables all transmissions from the display. When low, CONTROL B allows the display to operate in the local mode where only special actions result in data being transmitted; when high, CONTROL B sets the echo mode and data is transmitted to the

CPU and not operated on by the display. The status lines, OUTBUSY and INBUSY, are discussed later. The I/O of the MDS is TTL logic with data bit seven as the most significant bit.

B. PLASMA DISPLAY SET INPUT/OUTPUT

The Plasma Display uses an eight Bit Differential Interface to communicate with the CPU over eight sets of twisted-pair input lines and eight sets of twisted pair output lines. Each pair has a positive side and a negative side and a logical "1" exists on the pair when the positive line is high and the negative line is low. When the opposite condition exists, a logical "0" is represented. There are four handshaking twisted-pair signal lines provided for control from the CPU. In addition, there are two twisted-pair lines for input control signals and two twisted-pair lines for status output signals. The two status lines provide information to the CPU on the status of the display. When STATUS A is high the display is available for receipt of a character. When low, a character is being processed and the input is unavailable. The STATUS B line is high when the optional parity detector indicates a parity error. The control signals, EXT GATE and INCLK, are discussed later. The data I/O of the Plasma Display consists of eight twisted-pair signals with the most significant bit being data bit zero.

C. TRANSMISSION OF DATA

The transmission of data between the MDS and the plasma display require handshaking signals. The signals used are included in Figure 2.

1. Transmission from CPU to Plasma

To transmit data to the display, the CPU places data on the eight output lines and sets OUTBUSY control line high. When the display receives this signal, it causes EXT GATE handshake signal to be set low. The EXT GATE signal from the display remains low until data is processed, after which the EXT GATE signal is set high. The CPU must maintain data on the data lines for as long as EXT GATE is low. OUTBUSY control line from the CPU may then go low. The OUTBUSY control line must make the transition between high and low for each character transmitted to the display [5].

2. Transmission from Plasma to CPU

To transmit data to the CPU, the display places data on the eight input lines of the CPU. When the CPU is ready to receive data, it will set the INBUSY handshake line high (this may occur before data is placed on the input lines). With INBUSY from the CPU high and data on the input lines, the display sets INCLK high. This signals the CPU that the display has data. The CPU then accepts the data and resets INBUSY line low. When this occurs, the plasma resets INCLK low and the display can continue operation [5].

D. INTERFACE DESCRIPTION

The function of the interface is to change the output of the Plasma Display Set (twisted-pair signals with data bit zero the most significant bit) to an acceptable input for the MDS (TTL signals with data bit seven the most significant bit) and vice versa.

When operating with the MDS system, the interface is as shown in Figure 2. The I/O of the plasma scope is through pin connector J-3 ,while the I/O of the MDS is through the MDS parallel I/O board which was assigned to ports four and five.

```
----------          ---------------------           ----------
         :          :                   :           :
   MDS   :          :     INTERFACE     :           :  PLASMA
         :          :                   :           :
         :          :   -------------   :           :   Out
         :          :   :           :   :           : Signals:
         :  input   :   :   line    :   :  output   : data (8)
         :<-------------:  receivers:<--------------: STATUS A
         :  lines   :   :           :   :  lines    : STATUS B
         :          :   :  AMD 9615 :   :           : INCLK
         :          :   -------------   :           : EXT GATE
  Out    :          :                   :           :
Signals: :          :   -------------   :           :
data (8) :          :   :           :   :           :
CONT. A  : output   :   :   line    :   :  input    :
CONT. B  :------------->:transmitters:-------------->:
INBUSY   : lines    :   :           :   :  lines    :
OUTBUSY  :          :   :  AMD 9614 :   :           :
         :          :   -------------   :           :
         :          :                   :           :
----------          ---------------------           ----------
```

INTERFACE DESCRIPTION

Figure 2

The plasma display may be operated without a CPU connected; i.e. off-line. The following jumpers on connector J-3 are required to operate the plasma for off-line operation: pins 7 to 54, 9 to 53, 14 to 46, 16 to 48, 18 to 49 and 20 to 51.

E. CURRENT SYSTEM CONFIGURATION

The present system configuration is shown in Figure 3.

```
----------      -------------      -------      ---------------
| plasma |      | interface |      | MDS |      | disk drives |
| scope  |<-----|           |<-----|     |<---->|             |
----------      -------------      -------      ---------------
    ↑                                 ↑
    |                                 |
-----------                     ------------
|  plasma  |                    |   crt    |
| keyboard |                    | keyboard |
|(omitted)|                     |          |
-----------                     ------------
```

SYSTEM CONFIGURATION

Figure 3

Because of a design flaw, the handshaking signals required for the plasma to transmit data to the MDS system are present only for four hundred nanoseconds, which is not in accordance with Ref. 5 which states that the signals remain "high" until being reset. Without additional hardware latches and registers, a two-way communication between the systems cannot be established. Effective one-way communication from the MDS to the plasma display was established by using plasma status lines for control.

Current configuration has the plasma scope functioning as a display which is driven from the MDS console keyboard

by the operator. This configuration has the advantages of operator flexibility and minimizing required plasma display hardware. This configuration would allow for the omission of duplicated hardware and firmware - such as memory buffers, hardwired logic and duplicated system firmware.

IV. DESCRIPTION OF SOFTWARE

A. SIMULATION OF PLASMA FUNCTIONS

To drive the plasma display from the MDS system requires the simulation of plasma display control functions. The operator has the capability of duplicating all control functions from the MDS system that could be generated from the plasma display keyboard (for one-way communication.) In the current configuration, two-way communication is not required because the operator has the capability to duplicate in MDS memory all that is written into plasma display memory. Plasma functions are detailed in Appendix B, Operation Manual.

B. TEST AND INTEGRATION

1. Test of the Plasma Display

The testing of the plasma display was done in accordance with System checkout procedures listed in Ref. 4. All functions were tested in the off-line mode of operation with only minor discrepancies noted. When the scope is cleared either in the alphanumeric or vector mode, some plasma cells remained lighted and some that were off originally were lighted. Though not as predominant, some points would fail to light. Multiple clears or multiple writes would usually correct this problem. It was noted that the various voltage levels, significantly the bias voltages, are adjustable. However, this is a factory adjustment.

2. Test of Hardware Interface.

The hardware interface is described in Appendix A. Input signals were applied and output signals were checked. There were no unexpected results.

3. System Integration.

The system was configured as shown in Figure 3. Software primitives were written to simulate plasma scope functions and tested. The plasma responded more consistently than when operated in the off-line mode. During system integration it was found that EXT GATE and INCLK signals were not in accordance with Ref. 5. The STATUS A signal was found to be essentially the complement of EXT GATE and since the EXT GATE signals were erratic, STATUS A was used in the software interface. No substitute signal could be found for the INCLK. However, additional hardware could make the signal consistent with the description in Ref. 5.

4. Operator Interface

Three sets of programs were developed for the plasma display. The first set was a series of independent programs to emulate the thirty-two functions (less those requiring two-way communication) provided by the plasma scope. These programs, operated at the systems level, were used to verify the operability of each function and to clarify their performance. Most functions were found to perform as specified in Ref. 5.

The second set of programs was developed to evaluate the human factors involved with operating the plasma display. These programs provide an operating environment

for interaction between an operator and the plasma display via the CRT console. Application of these programs revealed no serious limitations on operation of the plasma scope in this configuration. However, since the plasma display writes and erases vectors by describing the end points, a program to determine the exact location (within one sixtieth of an inch) of points on the plasma was developed. This program would not be necessary with a functional transmit interface (two-way communication). Also, programs to simulate the plasma display memory in the MDS would not be necessary on a system with two-way communications. However, more efficient utilization of memory can be made by placing it in the MDS system vice the Plasma display system.

The third set of programs was a series of subroutines for use when writing programs for the plasma display. These include necessary primitives for communicating in alphanumeric and vector modes. Additionally, convenient routines to perform common repetitive functions have been provided in a library file. These primitives and routines may be accessed without re-writing or re-compiling them in an application program. Appendix B gives pertinent information on use of these routines.

C. CURRENT EXTENSIONS OF PLASMA FUNCTIONS

The plasma display allows the operator to function in two input modes - foreground and background. The foreground characters may be edited, whereas background characters may not. The disadvantage of not being able to edit background

characters is that the background can never be changed; the operator cannot correct typing errors when entering background characters, and he cannot change background data at a later date if he desires. Operating from the MDS, the operator has use of subroutines which change background data to foreground data, and vice versa.

The operator, from the MDS, also has the ability to call a vector cursor subroutine which will greatly aid in the drawing of vectors. Without this routine, drawing a vector from a point to a line with any accuracy would be practically impossible.

Other capabilities given the operator is the ability to modify the simulated plasma memory in the MDS and to display the modifications of the plasma display.

D. FUTURE EXTENSIONS OF PLASMA FUNCTIONS

One advantage of driving the plasma display from the MDS system is the flexibility of software over hardware - the MDS is programmable, the plasma display is not. Extensions required to make the system complete for storage and retrieval of display data are library, cataloging and file management routines. Possible extensions in the graphics area are coordinate system transformations, vector transformations, window and clipping algorithms, a graphic language and three dimemsional graphics.

E. FUTURE PLASMA DISPLAY SYSTEMS

If the ability of the plasma display to operate off-line is utilized, then a system of two or more displays can be combined with a single MDS. This could best be accomplished by utilizing the interupt mechanisms of the MDS. The system could be designed to operate in a variety of operating modes. For example, one plasma could be operating off-line utilizing the CPU only when transferring data between files or utilizing special function codes. At the same time, another plasma operating on-line could be utilizing the full computing power of the MDS.

Through the use of function codes (operator programmable in the MDS) available on the plasma keyboard and a two-way communication system, an operator on the plasma keyboard can still utilize many of the same programmable functions available to the operator on the MDS keyboard.

V. MILITARY APPLICATIONS

A. ALPHANUMERICS

The nature of plasma displays to have characters in a foreground or background lends itself readily to administrative reports - such as Optical Character Recognition (OCR) forms, standardized reports, executive orders and official correspondence. Libraries of standard forms can be kept on a direct access storage device. The operator could access the required form by Department of Defense (DD) number, display the form in the background mode and fill in the required information. This would reduce the need for many different forms and would reduce the number of copies of each form used by the administration departments. The information could be stored or sent directly to a printer. The printer could fill in a blank form using foreground data or it could print the form along with the data using both foreground and background data.

B. STATIC GRAPHICS

The military organizations make wide use of status or "tote" boards for displaying and keeping current of vital data in Command and Control centers. Present means of display are slow, inaccurate and wasteful. Computerized graphic displays for status boards would do much to improve the timeliness and accuracy of vital information.

Plasma graphics would also be beneficial to the military intelligence community. The ability to project images from the rear of the plasma display gives intelligence officers the capability of storing vast amounts of data for specific geographic locations that cannot be displayed on maps and can still be easily accessible. The capability of rear projection would aid mission commanders in mission planning by making the most current information readily available at a central location.

C. DYNAMIC GRAPHICS

The dynamics of stroke writers would be difficult, if not impossible, to simulate on plasmas because of the time required to move a number of vectors. That is, the writing, erasing and rewriting of vectors can make motion appear disjoint. However, certain dynamic graphic applications that could utilize the advantages of plasmas could more readily lend themselves to dynamic motion on plasma displays.

For example, the plasma would make an excellent display in the Command and Control environment, such as usage in the Navy and Marine tactical data systems (NTDS/MTDS). The dynamics of these tactical data systems are much slower than the continuous motion required for flight simulation (typical updates on TDS are six times a second).

VI. CONCLUSION

A broad overview of plasma display technology and operation has been presented. The advantages and disadvantages of plasma displays have been discussed and comparisons between different technologies were made. Important, unexpected results were noted in the plasma's resolution and dynamic capability. The resolution of the plasma did not appreciably effect the appearance of the display. For example, graphs of circles and polynomials appeared smooth and continuous. Although plasmas cannot project continuous motion as effectively as stroke writers, they can simulate the slower dynamics of a radar sweep. From information presented, plasma displays appear to be competitive with other graphic devices for most applications.

While there are a variety of graphics systems available, the graphics system selected for an application depends upon the functions the system performs. The advantages offered by plasma displays make them desirable for military use in environments which require rugged, vibration resistent, shockproof functioning. However, the display by SAI has capabilities which are not required under the systems current configuration. When connected to a CPU, a less sophistocated version with reduced buffer capabilities and reduced firmware can provide the same advantages.

The interface design and software tools presented in this thesis provide a foundation for further research and development into microcomputer based plasma display systems. Possible areas for development include transporting existing BASIC programs from other graphics systems to the MDS by implementing BASIC under the ISIS operating system or by writing emulation programs. The usefulness of the plasma display would be extended by developing a file management system, attaching a dot matrix printer for hard copies of graphs and developing a text editor that takes advantage of plasma text processing capabilities.

APPENDIX A

HARDWARE INTERFACE

A. INTERFACE BOARD DESIGN

The board configuration is shown if Figure A-1.

```
                 -------------------
                 ¦ Rx #1      Tx #1 ¦
      ----------¦ Rx #2      Tx #2 ¦----------
      ¦ 26 pin   ¦ Rx #3      Tx #3 ¦  55 pin ¦
      ¦connector¦ Rx #4      Tx #4 ¦connector¦
      ¦ to CPU   ¦ Rx #6      Tx #6 ¦to plasma¦
      ----------¦ Rx #6      Tx #6 ¦----------
                 -------------------
```

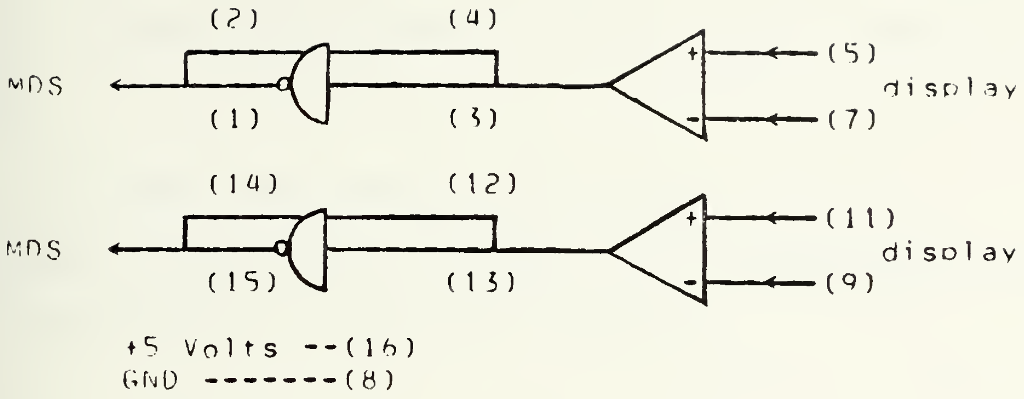
INTERFACE CONFIGURATION DESIGN

Figure A-1

where Rx and Tx are AMD 9615 and 9614 receiver and transmitter "chips" respectively.

B. AMD 9615 RECEIVER

AMD 9615 logic is shown in Figure A-2. Pin numbers are shown in parentheses. The inputs are twisted-pair signals from the plasma display and the outputs are TTL signals to the MDS. The interface board contains six AMD 9615 "chips", each "chip" has two receivers, giving twelve twisted-pair input lines and TTL output lines. These twelve lines include eight data lines, two status lines and two control lines.

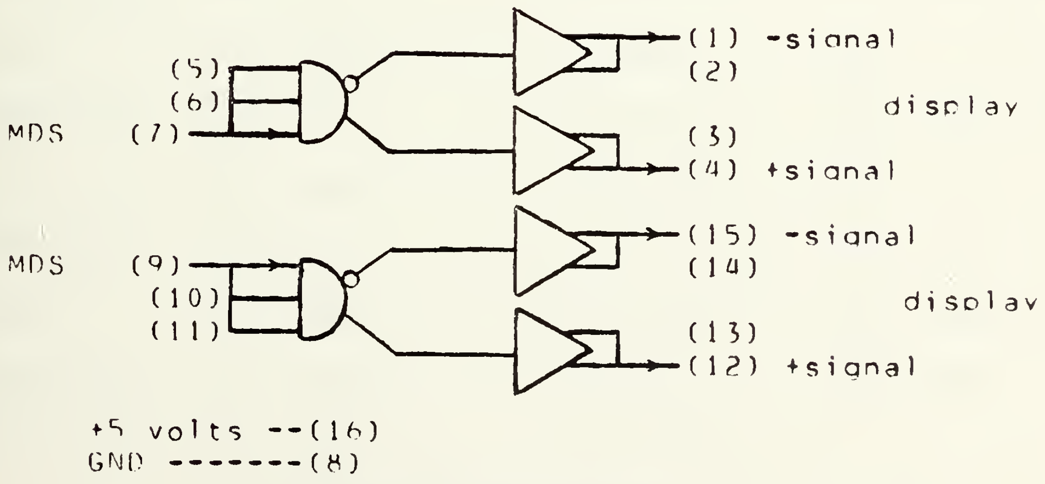


AMD 9615 LOGIC

Figure A-2

C. AMD 9614 TRANSMITTER

AMD 9614 logic is shown in Figure A-3. Pin numbers are shown in parentheses. The inputs are TTL signals from the MDS and the outputs are twisted-pair signals to the plasma display.



AMD 9614 LOGIC

Figure A-3

The interface board contains six AMD 9614 "chips", each chip has two transmitters, giving twelve TTL input lines and twisted-pair output lines. These twelve lines include eight data lines and four control lines.

D. J-3 PLASMA CONNECTOR PIN ASSIGNMENTS

Table A-1 contains pin numbers and corresponding signals for connector J-3 on the plasma display.

pin	signal	pin	signal	pin	signal	pin	signal
1	unused	15	-in 3	29	-in 0	43	+in 7
2	+out 6	16	-control A	30	GND	44	-out 0
3	-out 5	17	+in 3	31	+in 6	45	-out 7
4	-out 6	18	+control B	32	GND	46	-in clock
5	+out 5	19	+in 2	33	-in 6	47	unused
6	+out 4	20	-control B	34	GND	48	+in clock
7	-status B	21	-in 2	35	-in 5	49	+ext gate
8	-out 4	22	-out 2	36	GND	50	-in busy
9	+status B	23	-in 1	37	+in 5	51	-ext gate
10	-out 7	24	+out 2	38	+out 1	52	+in busy
11	+status A	25	+in 1	39	+in 4	53	+out busy
12	+out 7	26	+out 3	40	-out 1	54	-out busy
13	-status A	27	+in 0	41	-in 4	55	GND
14	+control A	28	-out 3	42	+out 0		

J-3 PIN ASSIGNMENTS

.Table A-1

E. INTERFACE WIRE CONNECTIONS

Interface wire connections from plasma display to MDS are included in Table A-2.

signal	J-3 pin #	chip #/ pin #	out	male pin #	female connector	CPU pin #
-in 1	23	1/11	1/15	D	17	55
+in 1	25	1/9				
-in 0	29	1/5	1/1	E	15	56
+in 0	27	1/7				
-in 3	15	2/11	2/15	H	13	57
+in 3	17	2/9				
-in 2	21	2/5	2/1	J	11	58
+in 2	19	2/7				
-in 5	35	3/11	3/15	L	9	60
+in 5	37	3/9				
-in 4	41	3/5	3/1	M	7	59
+in 4	39	3/7				
-in 7	45	4/11	4/15	P	5	61
+in 7	43	4/9				
-in 6	33	4/5	4/1	R	3	62
+in 6	31	4/7				
-ext gate	51	5/11	5/15	T	19	66
+ext gate	49	5/9				
-in clock	46	5/5	5/1	U	21	65
+in clock	48	5/7				
-status B	7	6/11	6/15	V	23	64
+status B	9	6/9				
-status A	13	6/5	6/1	X	24	63
+status A	11	6/7				

PLASMA TO MDS WIRE CONNECTIONS

Table A-2

Interface wire connnections from the MDS to the plasma display are included in Table A-3.

CPU pin #	female connector	male pin #	input	chip #/ pin #	J-3 pin #	signal
7	14	14	1/7	1/1	44	-out 0
				1/4	42	+out 0
6	12	12	1/9	1/15	40	-out 1
				1/12	38	+out 1
8	10	10	2/7	2/1	22	-out 2
				2/4	24	+out 2
5	8	8	2/9	2/15	28	-out 3
				2/12	26	+out 3
11	6	6	3/7	3/1	8	-out 4
				3/4	6	+out 4
10	4	4	3/9	3/15	3	-out 5
				3/12	5	+out 5
3	2	2	4/7	4/1	4	-out 6
				4/4	2	+out 6
9	1	1	4/9	4/15	10	-out 7
				4/12	12	+out 7
19	16	16	5/7	5/1	54	-out busy
				5/4	53	+out busy
18	18	18	5/9	5/15	50	-in busy
				5/12	52	+in busy
20	20	20	6/7	6/1	16	-control A
				6/4	14	+control A
15	22	22	6/9	6/15	20	-control B
				6/12	18	+control B

MDS TO PLASMA WIRE CONNECTIONS

Table A-3

Power and ground (GND) are supplied to the interface board at male pin numbers B and Z respectively. The power input to B is obtained from a separate power supply and the ground input to Z is supplied from female connector pin number 25. A common ground for the MDS, plasma display and interface was supplied.

APPENDIX B

OPERATION MANUAL

for the

AN/UYO-10 PLASMA DISPLAY SET/INTELLEC MICROCOMPUTER SYSTEM

at the

NAVAL POSTGRADUATE SCHOOL

This manual describes the operation of the AN/UYO-10 Plasma Display System (PDS) as configured with the Intellec Microcomputer Development System (MDS) at the Naval Postgraduate School. The specifics about the hardware interface were addressed in Appendix A. This manual assumes knowledge of the MDS System operating under ISIS-II. Information on ISIS-II may be obtained from Ref. 6. The programs for this system were written in PL/M-80 which is a high level language for microcomputers and is described in detail in Ref. 8. For information on the input/output module used on the MDS, attention is directed to Ref. 9.

I. SYSTEM BASICS

The AN/UYQ-10 Plasma Display Set (PDS) was connected to the Intellec Microcomputer Development System (MDS) via a locally developed interface board. The physical connections are reviewed in Chapter 1. This system may be exercised from the MDS using three methods. The most direct method is using the independent programs provided on the diskette labeled "Plasma.sys." These programs were designed to exercise one simple function each. Each program may be executed to clarify the various functions available on the PDS by entering the plasma mnemonic. These functions have been described in Ref. 5. Also, each function may be exercised using control codes generated on the MDS console by selecting the proper control key; i.e. depressing the "CTRL" key and another key. The thirty-two control codes used by the PDS are listed in Chapter 2. The easiest way to get control codes passed to the PDS is by using the "ALPHA" program on the demonstration diskette. Chapter 2 describes the various functions provided on the PDS by the basic programs and the control codes used to generate each function.

A more thorough exercise for the PDS functions may be accomplished using the demonstration routines. These routines were provided to display the capabilities of the PDS. By using the diskette labeled "Plasma.dem," various displays may be generated, information may be stored in the MDS memory, and stored information may be transmitted back

to the PDS. These routines are described in Chapter 3. The demonstration program uses subroutines which are available on the "Plasma.lib" diskette.

The routines on "Plasma.lib" are generally basic functions which may be linked to any appropriate object module. The object modules do not have to be derived from PL/M-80 source code but they do have to follow the parameter passing conventions described in Ref. 8. The external call statements needed to compile PL/M-80 programs without these basic functions have been included with the program source listings. Use of these subroutines have been described in Chapter 4.

One other diskette has been provided for completeness. "Plasma.plm" contains the source code for all the programs developed. Like "Plasma.lib," this is a non-system diskette. These programs were written to provide specifics on at least one way to accomplish various procedures, and to provide an indication of any idiosyncrasies which might limit the usefulness of the plasma display.

II. SYSTEM ORGANIZATION

To properly configure the PDS system, the MDS must have an Input/Output (I/O) Module in addition to the normal modules. This module has two dial type switches which must be set to assign the base address of "0004" to the I/O ports. If there is any doubt about the validity of this physical connection, it should be verified on an oscilloscope. Experience would indicate these switches may not produce the connection indicated by their markings. The interface board must be connected between the PDS and the MDS, and an external power supply must provide five volts to pin B and ground to pin Z of the interface board. If the additional connectors have been provided to complete this interface, the five volt supply should be provided from the MDS. Even though the PDS keyboard is not needed for input to this system, it must be connected unless a jumper is provided. If these connections are properly made, there should be no problems applying power and executing the basic function programs.

The basic function programs are provided on the "Plasma.dem" diskette. This diskette is a systems diskette, and as such may be placed in drive 0. After insuring that power has been turned on to the MDS and its console device, the interface board, and the PDS, the ISIS-II operating system should assure control of the MDS system. Typing "init" carriage return (c/r) should erase any stray dots that may

have been lighted when power was applied and post ON LINE on the plasma display. Entering "etx" c/r should enable the PDS keyboard and display the alphanumeric (A/N) cursor. It should transmit characters to the display screen. Similarly, other basic functions may be called using the mnemonics listed in Ref. 5 and also, in Table B-1. When a function such as "stx, sub, or cg" is selected, the following three bytes of data will be treated as control information.

A demonstration program is also provided on "Plasma.dem." This program provides the basic functions, and a few special functions to manipulate MDS memory, draw vectors, and generate some special displays. The demonstration program is documented in Chapter 3.

plasma mnem-onic	descriptive name	hexa-decimal code	con-trol key	function description
null	no action	00	@ or sp	causes no action on display
ch	cursor home	01	A	cursor counters zeroed (upper left corner)
stx	start text	02	B	sets ASCII input clears edit mode erases alphanumeric (A/N) cursor disables keyboard followed by column and line location and text
etx	enable text	03	C	clears cpu input mode writes A/N cursor enables plasma keyboard
fcl	function code	04	D	special function code not implemented
ca	cursor address	05	E	requests current cursor location from plasma not implemented
fs	forward space	06	F	increments cursor counter
bl	bell	07	G	sound bell 0.5 seconds not implemented
bs	backspace	08	H	decrements cursor counter
tab	to after background	09	I	moves cursor to first location following next background data or end of screen
lf	line feed	0A	J	increments line counter
vt	vertical tab	0B	K	decrements line counter
cs	clear screen	0C	L	erases screen rewrites vectors only
cr	carriage return	0D	M	zeroes column counter

PLASMA CONTROL CODES
Table B-1(a)

plasma mnem-onic	descriptive name	hexa-decimal code	con-trol key	function description
cg	construct graph	0E	N	sets graphics input mode erases cursor disables keyboard followed by 3 bytes of graphics data describes one end point
cv	clear vectors	0F	O	erases screen rewites A/N data only
fc2	function code	10	P	special function code not implemented
bg	background mode	11	Q	sets background mode A/N only etx or fg resets bg
fg	foreground mode	12	R	sets foreground mode A/N only
cb	clear background	13	S	erases background data
cf	clear foreground	14	T	erases foreground data
vr	verify	15	U	sends location and ascii character to cpu for verification not implemented
syn	synchronize	16	V	no action used to synchronize I/O interface
fc3	function code	17	W	special function code not implemented
can	cancel field	18	X	erases foreground data to last background data
fc4	function code	19	Y	special function code not implemented moves following lines up

PLASMA CONTROL CODES
Table B-1(b)

plasma mnem-onic	descriptive name	hexa-decimal code	con-trol key	function description
sub	substitute	1A	Z	substitute character sets ASCII input mode erases A/N cursor disables keyboard followed by column, line and character
fc5	function code	1B	[or ;	special function code not implemented
ir	insert record	1C	back-slant or <	creates blank line at current cursor location
dr	delete record	1D	] or =	erases line at cursor
.ich	insert character	1E	↑ or >	creates blank at cursor moves characters right one column column 80 character lost
dch	delete character	1F	under-score or ?	erases character at cursor location moves following characters left one position column 80 erased

PLASMA CONTROL CODES
Table B-1(c)

III. SYSTEM DEMONSTRATION

A. INVOKING THE DEMONSTRATION PROGRAM

The demonstration program on the system diskette labeled "Plasma.dem" may be placed in drive zero. The program is invoked by typing "demo." Sufficient prompts and information were provided for easy use without extensive knowledge of the program internals. This program provides an easy method by which to exercise the plasma display. It contains commands for all the basic functions which do not require data transmission from the plasma set, plus some extended functions and a few display routines. Data transmission from the plasma display was not implemented because the plasma display has a nonstandard interface. Additional circuitry must be added to the interface board if this capability is desired. The demonstration program uses all the subroutines provided in the plasma library (Plasma.lib) and the demonstration library (Demo.lib). It provides an easy interface for drawing vectors. Further, there is a set of display routines which exercises the plasma display set using various schemes to provide an indication of its capabilities. Example B-1 which uses the demonstration program has been appended to this chapter.

B. DEMONSTRATION PROGRAM COMMANDS

The basic functons may be called using the same mnemonics listed in Table B-1. The extended functions provide a

method of manipulating MDS memory, and of generating vectors. These functions are listed in Table B-2 with a brief explanation of their purpose. The display routines available with the demonstration program will provide some insight into the capabilities of the plasma display and the software interface provided. These functions are listed in Table B-3.

The demonstration program does not automatically perform those functions which may be called within the program. For example, calling individual graphs will not clear the screen prior to their display since clearing alphanumeric (CS) or vector memory (CV) are independent basic functions available within this program. Also, some programs produce different results when the origin is translated. This lack of automation provides more flexibility since graphs may be overlaid or modified using "alpha" or "vector" or another graph. However it would not be practical to require screen clearing for functions such as "dump" or "display all," so these functions clear the display when required.

The functions available in "demo" were placed in an object library called "Plasma.lib," and the display routines are in "Demo.lib." These functions are available without recompiling as explained in Chapter 4. With little effort, one should be able to use these functions with confidence since they may be exercised both as independent programs, and as calls from the demonstration program. Also, the source code has been appended in hopes of easing further development and improvement of the plasma set functions.

mnemonic	descriptive name	function description
btof	background to foreground	changes background characters to foreground characters in the MDS memory by setting high order bit to zero memory must be dumped to reset plasma
ftob	foreground to background	changes foreground characters to background characters in the MDS memory by setting high order bit to one memory must be dumped to reset plasma
dump	dump memory	clears display and writes contents of MDS buffers used to reset plasma when mode changed
vector	draw vectors	provides an interface with the MDS console for drawing vectors. default modes are set and only those items that change need to be entered. Q/NQ stand for query/no query which provides a mechanism to check input before it is passed to display. order of input is not important. duplicate input uses the last entry vectors are stored in MDS memory. escape terminates program. additional comments in chapter 4 under "Graphics One."
cursor	move cursor and report coordinates	provides mechanism to move vector cursor center of diamond reported on console when terminated with an escape control-f = forward one space (1/60 ") control-h = backspace one space control-j = down one space control-k = up one space
file	read/write files of MDS memory	provides basic file handling capabilities to save and recover MDS memory files are dumped uncompressed user must know what types of memory and in what sequence files were dumped multiple graphs and A/N datasets may be dumped to one file escape provides abnormal termination uses ISIS system functions
cmds	commands	produces list of "Demo" commands may be entered when prompt is "%"

EXTENDED PLASMA FUNCTIONS
Table B-2

mnemonic	descriptive name	display description
fsr	fill screen with rows	writes 512 row vectors on plasma
mrcv	move row down screen using clear vector	move row vector down screen erasing screen after each vector erases all vector memory in plasma each time screen is cleared alphanumerics are rewritten each time screen is cleared
fsc	fill screen with columns	writes 512 column vectors on plasma
esc	erase screen by column	erases screen a column at a time
mdc	moving double column	moves a column across screen writes next column before erasing prior column
fsrc	fill screen with rows and columns	writes row vector then column vector 512 times
mrc	moving row and column	erases then writes row vector then erases and writes column vector
tl	translate origin	accepts input from console and sets origin at coordinates given must be from 0 to 511 inclusive
fxg	function of x graph	plots a polynomial with origin of (255, 255) negative points are compliments of points used to plot positive curve
fans	radial lines	draws radial vectors from two origins origin cannot be moved
hw	heatwave	multiple arcs set from two origins plot uses individual points at each degree mark origins cannot be moved

PLASMA DISPLAY ROUTINES
Table B-3(a)

mnemonic	descriptive name	display description
ge	goose eggs	plots circles while they fit on display origin may be moved using translate points plotted are modulo 511 sin and cos uses 2 digit approximation plots each degree 0-45
tb	thunderbird	expanding radial lines from movable origin displays line length on console in hexadecimal expands at 20 dots per lap plots every twentieth vector
rs	radar scan	simulates radar sweep at varying rates rate in dots advanced per vector must be provided origin is movable
da	display all	runs all displays available in "Vdemo" provides mechanism to stop after each display
menu	commands	produces list of "Vdemo" commands may be entered when prompt is "<"

PLASMA DISPLAY ROUTINES
Table B-3(b)

C. DEMONSTRATION PROGRAM EXAMPLE

The following example illustrates a few of the commands and the general command procedure used in the demonstration program. Small letters indicate information keyed on the console device, capital letters indicate responses on the console device, quoted capitals indicate responses on the plasma, and "←" indicates a carriage return must be entered. Parenthesized notes are for explanatory purposes only.

```
(Insure system is configured as specified in Chapter 2, "Sys-
tems Organization," and "Plasma.dem" diskette is in drive 0.)

demo←
COMMAND LIST
NULL, CH, STX, ETX, FS, BS, ....
"ON LINE"        (Plasma screen cleared prior to message.)
%       (Percent sign posted on console as prompt for "Demo.")

abc←
INVALID COMMAND
(No action taken on plasma.)
%

cs←
(Plasma screen cleared.)
%

fg←            (Sets foreground mode.)
%

etx←           (Enables plasma keyboard.)
%              (If large keyboard on plasma, set in A/N mode.)

alpha←
(Line feed, carriage return, but no prompt, passed to console.)
(Plasma uses uppercase letters only, hence press alpha lock.)
abcdef←
ABCDEF
"ABCDEF"
bs,bs,bs       (Backspace)
               (Cursor is under "D" on plasma set.)
escape key
%
```

DEMONSTRATION PROGRAM EXAMPLE
Example B-1(a)

```
bg←             (Sets background mode.)
%

alpha←
xyz
"ABCXYZ"
bs,bs,bs        (Cursor will not backspace under background data.)
←               (Cursor is under "A.")
escape key
%

ftob            (Changes foreground data to background.)
FTOB
%

dump            (Resets plasma data.)
DUMP
%

"ABCXYZ"        (Plasma clears and writes memory in background mode.)
ch
CH
%

"ABCXYZ"        (Plasma cursor follows "Z.")

vdemo
VDEMO
%

USE ISOLATED UPPERCASE LETTERS ....
<       (Less than symbol used as prompt for "Vdemo.")
da
DO YOU WANT TO STOP AFTER EACH GRAPH? (Y/N) ("Yes" is default.)
n
(Numerous displays are produced on the plasma.)
SWEEP RATE?
5←
                (Simulated radar scan appears on plasma set.)
<
exit
%
exit
-       (Program terminated normally.)
```

DEMONSTRATION PROGRAM EXAMPLE
Example B-1(b)

IV. SYSTEM SUBROUTINES

The diskette labeled "Plasma.lib" is a non-system diskette which contains the PL/M-80 compiler,PL/M-80 library, the system library, the plasma library, the demonstration library, all the external files, and some example "submit" (.CSD) files. These files may be used when developing programs for the plasma system to avoid having to recompile any of the subroutines contained therein. The source for the plasma library files is contained on the "Plasma.plm" diskette. A listing of these source files has been appended to this thesis.

A. PROGRAM DEVELOPMENT

The system is predicated on a development scheme that uses a systems diskette in drive 0 which contains the source program being developed, and the "Plasma.lib" diskette in drive 1. If the developing program is qualified with a ".PLM," then the "P.CSD" file may be used directly by typing "submit :f1:p(your-file)" on the console. This will invoke a compile, link, locate, and go process where all the necessary library routines will be linked into the executable module as per the external statements specified in the developing program. The resulting located module will be in a file using the file name without any qualifier. Details for developing a ".CSD" file may be found in Ref. 6. Some compilable programs which were developed with this scheme

are available on "Plasma.plm." Familiarity with the required "include" statements may be gained by copying any of the stand-alone programs onto a working diskette and submitting it to the system. Compiler options for the submit file and specifics about the "include" and external statements are contained in Ref. 8. Additional object modules may be placed into "Plasma.lib" or "Demo.lib" as may be appropriate. These library files could be improved by separating all the contained object modules. The large groupings used on "Plasma.lib" proved to be disadvantageous on many occasions. There is no requirement for modules being linked together to have similar compiler options. For example, a module compiled using "symbols, cross reference, debug, and list" may be linked to this library which used the faster compile available under "C.CSD" without any adverse effects. Of course the symbols and line locations for the library routines will not be displayed in the locate map. Further, programs written in other languages may be linked if they follow the register conventions specified in Ref. 6.

The following paragraphs explain the subroutines available on "Plasma.lib," and "Demo.lib." Additional information is available in the source listings appended hereto.

B. DECLARATION FILES

Three declaration files have been included with the program listings. These files may be included in any program being developed if consistency with the subroutines is desired. All of the subroutines use these declaration files

to establish general abbreviations for recurring compiler tokens. The initial declarations, "Init.dcl," contains common abbreviations for declare, literally, procedure, et cetera, and literals for port assignments. The plasma declarations, "Pscode.dcl," contain literal substitutions for the 32 plasma control mnemonics. The graphics declaration file, "cg.dcl," contain literals for bit masks used to set up the control bytes needed in the graphics mode. These files may be included, as needed, following the first "do" in a program module. However, the plasma and graphics declaration files must be preceded by "Init.dcl."

C. CRT INTERFACE

The crt subroutines furnished are fairly common. These routines may be replaced with ISIS systems calls, if preferred. However, the buffering allocated by the ISIS systems will require about twice as much space. "Read crt" and "Write crt" manipulate only one character, and "Echo crt" combines read and write. "Read crt" strips any parity bit (bit 7) that may be passed by a terminal. "Read line crt" reads characters until a carriage return is detected. The characters read are converted to uppercase and stored at the buffer address provided. This routine appends a double dollar sign to the end of the input string. "Write line crt" places characters on the crt port until a double dollar sign is detected. These routines expect correct parameters and ample space for buffering, no checks are made to insure accurate calls.

D. PLASMA SET INTERFACE

The plasma set subroutines parallel the crt routines and provide similar functions to store all I/O information in the MDS memory. "Set status ps" inverts and sets the plasma set control lines as requested. "Write p" passes a single byte to the plasma set, while "Write ps" passes a byte and stores it in the CPU memory. "Write line ps" posts characters to the plasma port until a double dollar sign is detected. This routine is not safe for passing other than ASCII characters, since multiple occurances of a hexadecimal 24 will terminate the string. "Write vector" posts four bytes to the plasma port. It was designed to pass the "cg" control (0E hex) and three control bytes which include the x and y location plus write/erase, solid/dashed, start/end bits. "Write vs" extends "Write vector" by storing the information in memory. "Initialize ps" has been included with these functions as an easy way to insure the plasma set is on line and ready to receive data. All of the plasma functions invert the output as required by the plasma set, hence calling routines should use positive logic. The read functions have not been implemented since the necessary hardware has not been implemented to provide for data transmission from the plasma set.

E. MISCELLANEOUS SUBROUTINES

The miscellaneous subroutines provided are for number conversions and text handling. There are routines to convert from ASCII characters to binary and back. Subroutines

to convert to printable decimal, hexadecimal and binary formats were provided. Also, subroutines to convert to printable formats and write the results on the crt are available. The text handling routines "detrash" and "find blank" return the number of bytes from a given address to an alphanumeric character (eliminating blanks, commas, semicolons, and tabs) and provide the number of alphanumeric characters to the next blank or carriage return.

"Search" is a sequential search routine which may be used to search a table of variable length tokens. The token number returned indicates the first match without regard to remaining characters in the token passed. Hence if an exact match is desired, tokens and token table entries must be isolated with blanks. Also, tokens which are composites of other tokens should be arranged with the longest entries first. For example, if the search table is to contain both "add" and "addition," the "addition" should appear first in the table since the token "addition" would match "add" in the table. Note, however, that "addition" would not match "add " due to the blank. One further extension of this search routine is possible. If abbreviations are allowable, passing a token terminated with a dollar sign will return the token number of the first table entry that matches through that dollar sign. To aid in understanding and testing applications of "search," a program called "find" has been provided. This program allows a table of 256 bytes or less, terminated with a double dollar sign to be entered from the console, and then search for a token. The

hexadecimal position of the token found is displayed. If no token is found, the number displayed will be one more than the number of entries in the table.

F. CURSOR SUBROUTINES

Since the vector cursor cannot be enabled from an external cpu, a set of subroutines have been provided to display and move a vector cursor. This set of subroutines may be used by calling "move cursor" to drive the cursor around the plasma display and return the location when the routine is exited (use escape). Individual procedures may be accessed to draw a dot or a set of dots (diamond) or erase them, also. The cursor location returned is the address of the center dot.

G. VECTOR SUBROUTINES

The subroutines which operate the plasma in graphics mode have been provided in two sets. The graphics one package was designed to interface with the MDS console, while graphics two was designed for internally generated vectors.

The graphics one package provides for interaction with the MDS console using tokens "X=, Y=, WRITE, ERASE, SOLID, DASHED, START, END, Q, NQ." All tokens may be abbreviated by the first two or more characters. Unqualified numbers are accepted as first "X" and then "Y", however a warning message is generated and the output must be verified. Multiple occurances of the same token are accepted and set to the last value received. "Q/NQ" stand for query or no query. In the query mode, the vector control word is displayed for

verification. A yes (Y) response passes the control word to the plasma set. Any other response is taken as "no" and waits for more input. Tokens may be input in any sequence separated by commas, semicolons, blanks, or tabs. The vector control word is initially set to "0, 0, START, SOLID, WRITE, Q." All attributes are carried forward unless changed, hence a carriage return when first invoked would set a starting point at position (0,0). If this were followed by a "X=511, Y=511, END," a solid vector would be drawn from corner to corner, since "WRITE" and "SOLID" were carried forward. Multiple end points build chained vectors. The individual routines within this package may be used to set X or Y, or to move character strings in memory. The "display vector attributes" procedure is a handy debug tool (see translate procedure for an example).

The graphics two package provides routines for handling internally generated vectors. It provides subroutines for setting X or Y without disturbing other attributes which may have already been set plus some extended functions. These extended functions draw row or column vectors, and translate the origin. The translate procedure is designed to accept two coordinates and a third variable to indicate whether to set a new origin or to translate the coordinates to the last origin. This function returns false if the requested origin cannot be set. However, it does not check limits when translating a vector.

H. MEMORY SUBROUTINES

The memory management subroutines have been divided into two sets of procedures. The plasma scope memory, "Psmem," routines control the MDS memory. While the plasma scope control, "Pscont" routines call the proper "Psmem" routines to emulate plasma memory. These routines use 8,804 bytes of memory to store information which may be used to reconstruct the current plasma set display. Further, facilities have been provided for filing this information on diskette for subsequent recall. The extensive set of procedures may be used independently if off-line development of displays is desired.

I. DEMONSTRATION SUBROUTINES

The demonstration subroutines and programs have been provided as a means of learning the affects of various plasma scope fuctions. These routines make extensive use of all the available procedures developed on this project. Additionally, included with the demonstration programs is an independent module to initialize the plasma set. This may prove necessary when operating the plasma set in an offline mode without disconnecting the plasma and connecting a jumper plug. Once the plasma has been turned on, entering "Init" will clear the transmission lines of extraneous signals and enable the keyboard.

J. TEST PROGRAMS

A set of test programs developed throughout this project have been included on the "Plasma.plm" diskette, and with the program listings. These programs proved invaluable in developing some of routines on this project. They have been included, as a vehicle for testing modifications prior to implementing a change to the basic system.

P R O G R A M   L I S T I N G S

CONTENTS

CONTENTS

```
/*DECLARATIONS: */

DECLARE LIT LITERALLY 'LITERALLY',
        DCL       LIT     'DECLARE',
        PROC      LIT     'PROCEDURE';

DCL     CR        LIT     '0DH',
        LF        LIT     '0AH',
        EOL       LIT     '24H';         /* "$$" MARKS END OF LINE BUFFER */

DCL     TRUE      LIT     '01H',
        FALSE     LIT     '00H';

DCL     FOREVER LIT       'WHILE TRUE';

DCL     CRT$DATA          LIT     '0F6H',     /* CRT DATA ON PORT 246 */
        CRT$STATUS        LIT     '0F7H',     /* CRT STATUS ON PORT 247 */
        PS$DATA           LIT     '04H',      /* PLASMA SCOPE DATA ON PORT 4 */
        PS$STATUS         LIT     '05H';      /* PLASMA SCOPE STATUS ON PORT 5 */

DCL     RECEIVE$MASK      LIT     '06H',
        TRANSMIT$MASK     LIT     '05H';

DCL     CTL$X             LIT     '18H',
        CTL$R             LIT     '12H';

DCL     RUBOUT            LIT     '7FH',
        ESCAPE            LIT     '1BH';

DCL     BS                LIT     '08H',
        COMMA             LIT     '2CH',
        SEMI$COLON        LIT     '3BH',
        TAB              LIT      '09H';
```

```
/*INITIALIZATION                                                              /*DECLARATIONS

    DCL     MASK$6          LIT         '0010$0000B',
            MASK$7          LIT         '0100$0000B';

    DCL     X$VECTOR        ADDRESS             INITIAL (1),
            Y$VECTOR        ADDRESS                     INITIAL (1);
```

```
/*DECLARATIONS: */

     DCL     NULL     LIT      '00H',
             CH       LIT      '01H',
             STX      LIT      '02H',
             ETX      LIT      '03H';

     DCL    FC1                LIT      '04H',
             CA                 LIT      '05H',
             FS                 LIT      '06H',
             BL                 LIT      '07H';

     DCL     VT                 LIT      '0BH',
             CS                 LIT      '0CH',
             CG                 LIT      '0EH',
             CV                 LIT      '0FH';

     DCL     FC2                LIT      '10H',
             BG                 LIT      '11H',
             FG                 LIT      '12H',
             CB                 LIT      '13H';

     DCL     CF                 LIT      '14H',
             VR                LIT      '15H',
             SYN                LIT      '16H',
             FC3                LIT      '17H';

     DCL     CAN                LIT      '18H',
             FC4                LIT      '19H',
             SUB                LIT      '1AH',
             FC5                LIT      '1BH';

     DCL     IR                 LIT      '1CH',
```

```
/*PLASMA$SCOPE$CODES                                        /*DECLARATIONS

           DR                  LIT       '1DH',
           ICH                 LIT       '1EH',
           DCH                 LIT       '1FH';

   DCL     IN$CLOCK            LIT       '01H',
           EXTERNAL$GATE     LIT       '02H',
           STATUS$A            LIT       '04H',
           STATUS$B            LIT       '08H',
           OUT$BUSY            LIT       '01H',
           IN$BUSY             LIT       '02H',
           CONTROL$A           LIT       '04H',
           CONTROL$B           LIT       '08H',
           RESET$ALL           LIT       '00H';

   DCL     COLUMN$0            LIT       '00H',
           LINE$2              LIT       '02H';

   DCL     NEUTRAL             LIT       '00H',
          ALP$WS             LIT       '01H',
           VEC$WS              LIT       '02H';
```

```
/*DECLARATIONS: */

      DCL     SET$ERASE         LIT      '0100$0000B',
              SET$DASHED        LIT      '0001$0000B',
              SET$END           LIT      '0010$0000B',
              X$MASK            LIT      '0000$0011B',
              Y$MASK            LIT      '0000$1100B',
              Q$MASK            LIT      '1000$0000B';

      DCL     ( CHAR, CONTINUE, TOKEN$NUMBER, INDEX ) BYTE;
      DCL     ( BPTR, Q, DIGITS, NO$X )         BYTE;

      DCL     ( X, Y )          ADDRESS;

      DCL     BUFFER( 124 )     BYTE;
      DCL     BUFF$SIZE        BYTE     INITIAL( 120 );
      DCL     CGS$FIRST         BYTE     INITIAL( TRUE );
      DCL     XLATE$FIRST       BYTE     INITIAL( TRUE );

      DCL     TOKEN$TABLE( 52 )       BYTE
              DATA( 'X=$', 'Y=$', 'WRITE$', 'ERASE$', 'SOLID $', 'DASHED$',
                      'START$', 'END  $', 'Q$', 'NQ$', '$$$' );

      DCL     TOKEN( 16 ) BYTE;
      DCL     VECTOR( 5 )       BYTE;
```

```
/*EXTERNALS: */

 WRITE$CRT:
      PROC( CHARACTER ) EXTERNAL;
              DCL      CHARACTER        BYTE;
      END WRITE$CRT;

 CRLF$CRT:
      PROC EXTERNAL;
      END CRLF$CRT;

 WRITE$LINE$CRT:                /* "A" IS THE ADDRESS OF AN 80 POSITION BUFFER */
      PROC( A ) EXTERNAL;
              DCL      A        ADDRESS;
      END WRITE$LINE$CRT;

 READ$CRT:
      PROC BYTE EXTERNAL;
      END READ$CRT;

 ECHO$CRT:
      PROC BYTE EXTERNAL;
      END ECHO$CRT;

 READ$LINE$CRT:
      PROC( BUFFER$ADDRESS ) EXTERNAL;
              DCL      BUFFER$ADDRESS   ADDRESS;
      END READ$LINE$CRT;
```

```
/*WRITE$CRT: */

DO;

$ INCLUDE( :F1:INIT.DCL )

 WRITE$CRT:  /* WRITE ONE CHARACTER TO CRT */
      PROC( CHARACTER ) PUBLIC;
               DCL      CHARACTER        BYTE;

      DO WHILE ( INPUT( CRT$STATUS ) AND TRANSMIT$MASK ) <> TRANSMIT$MASK;
               /*       WAIT     */
     END;

      OUTPUT( CRT$DATA ) = CHARACTER;

      END WRITE$CRT;
```

```
CRLF$CRT:  /* CARRIAGE RETURN AND LINE FEED TO CRT */
      PROC PUBLIC;

      CALL WRITE$CRT( CR );
      CALL WRITE$CRT( LF );

      END CRLF$CRT;
```

```
WRITE$LINE$CRT:  /* WRITES A LINE TO CRT BEGINNING AT ADDRESS IN CALL AND */
               /* ENDING WITH FIRST OCCURENCE OF DOUBLE DOLLAR SIGNS */
      PROC( A ) PUBLIC;
              DCL      ( POINTER, A )  ADDRESS;
              DCL      (BUFFER BASED A) ( 80 ) BYTE;

      POINTER = 0;

      DO WHILE ( BUFFER( POINTER ) <> EOL ) OR
              ( BUFFER( POINTER + 1 ) <> EOL );
              CALL WRITE$CRT( BUFFER( POINTER ) );
              POINTER = POINTER + 1;
      END;

      END WRITE$LINE$CRT;
```

```
READ$CRT:  /* READS ONE CHARACTER FROM KEYBOARD AND CLEARS PARITY BIT */
      PROC BYTE PUBLIC;

              DCL      ( CHARACTER, LAST$CHAR )        BYTE;
              DCL     PARITY$MASK      LIT      '0111$1111B';

      LAST$CHAR = CHARACTER;

      DO WHILE ( INPUT( CRT$STATUS ) AND RECEIVE$MASK ) <> RECEIVE$MASK;
              /*       WAIT    */
      END;

      CHARACTER = INPUT( CRT$DATA ) AND PARITY$MASK;
      RETURN CHARACTER;

      END READ$CRT;
```

```
ECHO$CRT:  /* READS CHARACTER FROM KEYBOARD, WRITES IT BACK TO CRT AND */
          /* PASSES IT BACK IN ECHO$CRT */
       PROC BYTE PUBLIC;
                DCL      CHARACTER       BYTE;

       CHARACTER = READ$CRT;
       CALL WRITE$CRT( CHARACTER );
       RETURN CHARACTER;

       END ECHO$CRT;
```

```
READ$LINE$CRT:
                    /* READ LINE FROM CRT AND STORE IN BUFFER */
                     /* CONVERT LOWER CASE TO UPPER CASE      */
                     /* REPEATS CHARACTERS RUBBED OUT */
                     /* CTL-R REPEATS LINE MINUS RUBOUTS */
                     /* CTL-X RESTARTS CURRENT LINE */

      PROC( BUFFER$ADDRESS ) PUBLIC;

              DCL     (LBP,CHAR)                   BYTE;
              DCL      BUFFER$ADDRESS               ADDRESS;
              DCL      (LINE$BUFFER BASED BUFFER$ADDRESS)(123) BYTE;

LBP = 0;
CHAR = ' ';
LINE$BUFFER( 121 ), LINE$BUFFER( 122 ) = '$';

DO WHILE ((CHAR <> CR) AND (LBP < 120));
      CHAR = READ$CRT;
      IF (((CHAR = BS) OR (CHAR = RUBOUT)) AND (LBP > 0))
              THEN DO;
                     LBP = LBP - 1;
                     IF CHAR = BS THEN CALL WRITE$CRT( CHAR );
                             ELSE CALL WRITE$CRT (LINE$BUFFER( LBP ));
                 END;
              ELSE IF (CHAR = CTL$R)
                     THEN DO;
                             LINE$BUFFER(LBP) = '$';
                             LINE$BUFFER(LBP + 1) = '$';
                             CALL CRLF$CRT;
                             CALL WRITE$LINE$CRT( BUFFER$ADDRESS );
                         END;
                     ELSE IF (CHAR = CTL$X)
```

```
                                  THEN DO;
                                          LBP = 0;
                                          CALL CRLF$CRT;
                                       END;
            ELSE DO;
                    CHAR = ((((CHAR) AND (MASK$6)) AND (SHR(((CHAR) AND (MASK$7))
                          ,1))) XOR (CHAR));
                    LINE$BUFFER( LBP ) = CHAR;
                    LBP = LBP + 1;
                    CALL WRITE$CRT( CHAR );
                 END;

END;  /* DO WHILE */

LINE$BUFFER( LBP ), LINE$BUFFER( LBP + 1 ) = '$';

END READ$LINE$CRT;

END CRT$FUNCTIONS;
```

```
/*EXTERNALS: */
 SET$STATUS$PS:
      PROC( MASK ) EXTERNAL;
               DCL      MASK     BYTE;
      END SET$STATUS$PS;

 WRITE$P:
      PROC( CHARACTER ) EXTERNAL;
               DCL      CHARACTER        BYTE;
      END WRITE$P;

 WRITE$PS:
      PROC( CHARACTER ) EXTERNAL;
               DCL      CHARACTER       BYTE;
      END WRITE$PS;

 WRITE$LINE$PS:
      PROC( LOCATION ) EXTERNAL;
               DCL      LOCATION         ADDRESS;
      END WRITE$LINE$PS;

 WRITE$VECTOR:
      PROC( VECTOR$ADDRESS ) EXTERNAL;
               DCL      VECTOR$ADDRESS   ADDRESS;
      END WRITE$VECTOR;

 WRITE$VS:
      PROC( VECTOR$ADDRESS ) EXTERNAL;
               DCL      VECTOR$ADDRESS   ADDRESS;
      END WRITE$VS;

 INITIALIZE$PS:
      PROC EXTERNAL;
```

```
          END INITIALIZE$PS;

;
```

```
/*SET$STATUS$PS: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )

 STORE$IN$MEMORY:
      PROCEDURE( CHARACTER ) EXTERNAL;
              DCL      CHARACTER        BYTE;
      END STORE$IN$MEMORY;

/*    PLASMA SCOPE ROUTINES             */

 SET$STATUS$PS:  /* SETS PLASMA SCOPE STATUS CONTROL LINES */
              /* USE INVERTED SIGNALS */

      PROCEDURE( MASK ) PUBLIC;
              DECLARE MASK   BYTE;

      OUTPUT( PS$STATUS ) = NOT MASK;

      END SET$STATUS$PS;
```

```
WRITE$P:
            /* WRITE CHARACTER TO THE PLASMA WITHOUT WRITING IN MEMORY */

     PROCEDURE( CHARACTER ) PUBLIC;

            DCL     CHARACTER       BYTE;

     DO WHILE ( ( NOT INPUT( PS$STATUS ) ) AND STATUS$A ) <> STATUS$A;
            /* WAIT FOR PLASMA TO READY */
     END;

     OUTPUT( PS$STATUS ) = NOT RESET$ALL;
     OUTPUT( PS$DATA ) = NOT CHARACTER;
     OUTPUT( PS$STATUS ) = NOT OUT$BUSY;

     END WRITE$P;
```

```
WRITE$PS:  /* PASS ONE CHARACTER TO PLASMA SCOPE AND STORE IN MEMORY.  */
      PROCEDURE( CHARACTER ) PUBLIC;

              DECLARE CHARACTER       BYTE;

      DO WHILE ( ( NOT INPUT( PS$STATUS ) ) AND STATUS$A ) <> STATUS$A;
              /* WAIT FOR PLASMA TO READY */
      END;

      OUTPUT( PS$STATUS ) = NOT RESET$ALL;
      OUTPUT( PS$DATA ) = NOT CHARACTER;                /* OUTPUT MUST BE INVERTED */
      OUTPUT( PS$STATUS ) = NOT OUT$BUSY;       /* SET OUT BUSY LINE */

      CALL STORE$IN$MEMORY( CHARACTER );

      END WRITE$PS;
```

```
WRITE$LINE$PS:

            /* WRITE A LINE FROM THE GIVEN ADDRESS TO A DOUBLE DOLLAR SIGN.
            DOES NOT STORE LINE IN MEMORY
            */
        PROCEDURE( A ) PUBLIC;

            DECLARE A        ADDRESS;
            DECLARE COUNT    BYTE;
            DECLARE ( LINE BASED A ) ( 160 )        BYTE;

        COUNT = 0;

        DO WHILE (( COUNT < 160 ) AND (( LINE( COUNT ) <> '$' )
                    OR ( LINE( COUNT + 1 ) <> '$' )));

            /* "$$" TERMINATES INPUT LINE */

            CALL WRITE$P( LINE( COUNT ) );
            COUNT = COUNT + 1;
        END;

        END WRITE$LINE$PS;
```

```
WRITE$VECTOR:
                /* WRITE CG, STX, SUB TO PLASMA SCOPE WITH THE FOLLOWING
                THREE BYTES */

      PROCEDURE( VECTOR$ADDRESS ) PUBLIC;

                DCL     VECTOR$ADDRESS  ADDRESS;
                DCL     (VECTOR BASED VECTOR$ADDRESS ) ( 4 )    BYTE;
                DCL     VPTR    BYTE;

      DO VPTR = 0 TO 3;
                CALL WRITE$P( VECTOR( VPTR ) );
      END;

      END WRITE$VECTOR;
```

```
WRITE#VS:
        /* WRITE CG, STX, OR SUB AND THE FOLLOWING THREE BYTES TO THE PLASMA
        SCOPE AND STORE IN MEMORY.
        */
PROCEDURE( VECTOR#ADDRESS ) PUBLIC;

    DCL     VECTOR#ADDRESS  ADDRESS;
    DCL     ( VECTOR BASED VECTOR#ADDRESS ) ( 4 )   BYTE;
    DCL     VPTR    BYTE;

DO VPTR = 0 TO 3;
    CALL WRITE#PS( VECTOR( VPTR ) );
END;

END WRITE#VS;
```

```
INITIALIZE$PS:

                /* CLEARS PLASMA DISPLAY AND POSTS "ON LINE" */

       PROCEDURE PUBLIC;

                DECLARE         BUFFER( * )      BYTE
                         DATA    ( CS, STX, COLUMN$0, LINE$2, 'ON LINE',
                                 STX, COLUMN$0, COLUMN$0, ETX, '$$' );

       CALL SET$STATUS$PS( IN$BUSY );           /* INSURE PLASMA IS NOT IN TRANSMIT MODE*/
       CALL SET$STATUS$PS( RESET$ALL );
       CALL WRITE$LINE$PS( .BUFFER );
       CALL SET$STATUS$PS( RESET$ALL );

       END INITIALIZE$PS;

END PLASMA$SCOPE$FUNCTIONS;
```

```
/*EXTERNALS: */

      OPEN:
               PROC (AFT, FILE, ACCESS, MODE, STATUS ) EXTERNAL;
                    DCL ( AFT, FILE, ACCESS, MODE, STATUS ) ADDRESS;
               END OPEN;

      CLOSE:
               PROC ( AFT, STATUS ) EXTERNAL;
                    DCL (AFT, STATUS ) ADDRESS;
               END CLOSE;

      READ:
               PROC ( AFT, BUFFER, COUNT, ACTUAL, STATUS ) EXTERNAL;
                    DCL ( AFT, BUFFER, COUNT, ACTUAL, STATUS ) ADDRESS;
               END READ;

      WRITE:
               PROC ( AFT, BUFFER, COUNT, STATUS ) EXTERNAL;
                    DCL ( AFT, BUFFER, COUNT, STATUS ) ADDRESS;
               END WRITE;

      EXIT:
               PROC EXTERNAL;
                    DCL STATUS ADDRESS;
               END EXIT;

      ERROR:
               PROC ( ERRNUM ) EXTERNAL;
                    DCL ( ERRNUM, STATUS ) ADDRESS;
               END ERROR;
```

```
/*EXTERNALS: */

 DISPLAY$BINARY:
      PROC( CHARACTER ) ADDRESS EXTERNAL;
               DCL      CHARACTER        BYTE;
      END DISPLAY$BINARY;

 DISPLAY$HEXADECIMAL:
      PROC( CHARACTER ) ADDRESS EXTERNAL;
               DCL      CHARACTER        BYTE;
      END DISPLAY$HEXADECIMAL;

 WRITE$BINARY:
      PROC( CHARACTER ) EXTERNAL;
               DCL      CHARACTER        BYTE;
      END WRITE$BINARY;

 WRITE$HEXADECIMAL:
      PROC( CHARACTER ) EXTERNAL;
               DCL      CHARACTER        BYTE;
      END WRITE$HEXADECIMAL;

 DISPLAY$DECIMAL:
      PROC( NUMBER, BUFFER$ADDRESS ) EXTERNAL;
              DCL      ( NUMBER, BUFFER$ADDRESS )       ADDRESS;
      END DISPLAY$DECIMAL;

 CONVERT$HEXADECIMAL:
      PROC( ASCII$ADDRESS ) ADDRESS EXTERNAL;
               DCL      ASCII$ADDRESS   ADDRESS;
      END CONVERT$HEXADECIMAL;

 DETRASH:
```

```
/*MISCELLANEOUS                                              /*EXTERNALS

     PROC( BUFFER$ADDRESS ) BYTE EXTERNAL;
             DCL      BUFFER$ADDRESS ADDRESS;
     END DETRASH;

FIND$BLANK:
     PROC( BUFFER$ADDRESS ) BYTE      EXTERNAL;
             DCL      BUFFER$ADDRESS  ADDRESS;
     END FIND$BLANK;

SEARCH:
     PROC( TOKEN$ADDRESS, TABLE$ADDRESS ) BYTE EXTERNAL;
             DCL      ( TOKEN$ADDRESS, TABLE$ADDRESS )         ADDRESS;
     END SEARCH;
```

```
/*DISPLAY$BINARY: */

DO;

$ INCLUDE( :F1:INIT.DCL )

 WRITE$LINE$CRT:                   /* USED BY WRITE CONVERSIONS */
      PROC( ADDR ) EXTERNAL;
              DCL      ADDR    ADDRESS;
      END WRITE$LINE$CRT;

 DISPLAY$BINARY:                   /* CONVERT CHARACTER TO DISPLAY BINARY FORMAT */
                                   /* ACCEPTS ONE BYTE, RETURNS ADDRESS OF ELEVEN */
                                   /* ELEMENT ARRAY WITH FORM "XXXXXXXX $$"      */
                                   /* SAMPLE CALL:
                                           A$ADDRESS = DISPLAY$BINARY( CHAR )    */

      PROC( CHARACTER ) ADDRESS PUBLIC;
              DCL      ( CHARACTER, BIT, DIGIT )       BYTE;
              DCL      BUFFER( 11 ) BYTE;

      BUFFER( 8 ) = ' ';
      BUFFER( 9 ), BUFFER( 10 ) = '$';
      BIT = -1;

      DO WHILE BIT <> 7;
             BUFFER( BIT := BIT + 1 ) = (( CHARACTER := ROL( CHARACTER, 1 ) )
                      AND 0000$0001B ) OR 30H;
      END;

      RETURN .BUFFER;
      END DISPLAY$BINARY;
```

```
DISPLAY$HEXADECIMAL:  /* CONVERT CHARACTER TO DISPLAY HEXADECIMAL FORMAT */
                      /* ACCEPTS ONE BYTE VARIABLE, RETURNS ADDRESS OF
                      FIVE ELEMENT ARRAY OF FORM "XX $$"                  */
                      /* SAMPLE CALL:
                           A$ADDRESS = DISPLAY$HEXADECIMAL( CHAR )       */

    PROC( CHARACTER ) ADDRESS PUBLIC;
         DCL      ( CHARACTER, NIBBLE )    BYTE;
         DCL      BUFFER( 5 )      BYTE;

    BUFFER( 2 ) = ' ';
    BUFFER( 3 ), BUFFER( 4 ) = '$';
    NIBBLE = -1;

    DO WHILE NIBBLE <> 1;
         BUFFER( NIBBLE := NIBBLE + 1 ) =
                 ( ( CHARACTER := ROL( CHARACTER, 4 ) ) AND 0FH );
         IF BUFFER( NIBBLE ) < 10 THEN
                 BUFFER( NIBBLE ) = BUFFER( NIBBLE ) OR 30H;
         ELSE
                 BUFFER( NIBBLE ) = ( BUFFER( NIBBLE ) - 9 ) OR 40H;
    END;

    RETURN . BUFFER;

    END DISPLAY$HEXADECIMAL;
```

```
WRITE$BINARY:         /* CONVERT AND WRITE VARIABLE IN BINARY FORMAT */
                                      /* ACCEPTS BYTE VARIABLE CONVERTS TO BINARY
                                      FORMAT AND OUTPUTS RESULTS ON CONSOLE DEVICE   */
                                      /*SAMPLE CALL:
                                              CALL WRITE$BINARY( CHAR )              */

      PROC( CHARACTER ) PUBLIC;
              DCL     CHARACTER       BYTE;
              DCL     A$ADDRESS       ADDRESS;

      A$ADDRESS = DISPLAY$BINARY( CHARACTER );
      CALL WRITE$LINE$CRT( A$ADDRESS );

      END WRITE$BINARY;
```

```
WRITE$HEXADECIMAL:      /* CONVERT AND WRITE VARIABLE IN HEXADECIMAL FORMAT */
                        /* ACCEPTS BYTE VARIABLE, CONVERTS TO HEXADECIMAL
                        FORMAT AND OUTPUTS RESULTS TO CONSOLE DEVICE        */
                        /* SAMPLE CALL:
                                CALL WRITE$HEXADECIMAL( CHAR )               */

      PROC( CHARACTER ) PUBLIC;
              DCL     CHARACTER        BYTE;
              DCL     A$ADDRESS        ADDRESS;

     A$ADDRESS = DISPLAY$HEXADECIMAL( CHARACTER );
      CALL WRITE$LINE$CRT( A$ADDRESS );

      END WRITE$HEXADECIMAL;
```

```
DISPLAY$DECIMAL:     /* CONVERTS TWO BYTE NUMBER TO DECIMAL DISPLAY FORMAT */

        PROC( NUMBER, BUFFER$ADDRESS ) PUBLIC;
                DCL ( NUMBER, BUFFER$ADDRESS ) ADDRESS;
                DCL ( ASCII$NUMBER BASED BUFFER$ADDRESS ) ( 10 )          BYTE;
                DCL     APTR     BYTE;
                DCL     TEN      ADDRESS DATA( 10 ),
                        ZIP     ADDRESS DATA( 0 );

        APTR = 0;

        DO WHILE APTR < 9;
                ASCII$NUMBER( APTR ) = ' ';
                APTR = APTR + 1;
        END;    /* DO WHILE */

        APTR = 9;

        CONVERT:  DO WHILE ( NUMBER <> ZIP ) AND ( APTR > 0 );
                ASCII$NUMBER( APTR ) = ( LOW( NUMBER MOD TEN ) ) OR 30H;
                NUMBER = NUMBER / 10;
                APTR = APTR - 1;
        END CONVERT;

END DISPLAY$DECIMAL;
```

```
DETRASH:        /* RETURNS NUMBER OF BYTES TO THE NEXT ELEMENT
                THAT IS NOT A BLANK, TAB, COMMA, OR SEMICOLON           */
                /* SAMPLE CALL:
                     POINTER = POINTER + DETRASH( BUFFER( POINTER )     */

    PROC ( BUFFER$ADDRESS ) BYTE PUBLIC;

                DCL     POINTER BYTE;
                DCL     BUFFER$ADDRESS  ADDRESS;
                DCL     ( BUFFER BASED BUFFER$ADDRESS) ( 123 )  BYTE;

    POINTER = 0;

    DO WHILE (POINTER < 120) AND ((BUFFER(POINTER) = COMMA) OR
                (BUFFER(POINTER) = ' ') OR (BUFFER(POINTER) = TAB) OR
                ( BUFFER( POINTER ) = SEMI$COLON ) );
                POINTER = POINTER + 1;
    END; /* DO WHILE */

    RETURN POINTER;

    END DETRASH;
```

```
CONVERT$HEXADECIMAL:  /* CONVERTS DISPLAY DECIMAL FORMATTED NUMBERS TO A
                      TWO BYTE HEXIDECIMAL ( BINARY ) NUMBER                 */
                      /* SAMPLE CALL:
                              HEX = CONVERT$HEXADECIMAL( ADDRESS$DECIMAL )   */

                      PROC( N$ADDRESS ) ADDRESS PUBLIC;
                              DCL DIGITS  BYTE;
                              DCL N$ADDRESS ADDRESS;
                              DCL (NUMBER BASED N$ADDRESS) (5) BYTE;
                              DCL HEX ADDRESS;
                              DCL     TEN     ADDRESS DATA( 10 );

      HEX = 0;
      DIGITS = 0;

      DO WHILE  ( NUMBER( DIGITS ) >= '0' ) AND ( NUMBER( DIGITS ) <= '9' );
              HEX = ( HEX * TEN ) + ( DOUBLE( NUMBER( DIGITS ) AND  0FH  ) );
              DIGITS = DIGITS + 1;
              IF DIGITS > 5
                      THEN DO;
                              CALL WRITE$LINE$CRT( .('NUMBER TOO LARGE',
                                      CR, LF, '$$' ) );
                              RETURN HEX;
                      END;    /* THEN DO */
      END;    /* DO WHILE */

      RETURN HEX;
END CONVERT$HEXADECIMAL;
```

```
SEARCH:

            /*      SEQUENTIALLY SEARCH A TABLE OF VARIABLE LENGTH
            TOKENS OF THE FORM "TOKEN$" UNTIL A MATCH IS
            FOUND OR THE END OF TABLE "$$".   RETURN THE TOKEN NUMBER
            RELATIVE TO THE START OF THE TABLE OR NUMBER OF TOKENS PLUS ONE.
            MATCH IS INDICATED WHEN CHARACTERS ARE EQUAL THROUGH THE
            FIRST "$" ON EITHER STRING.   */

        PROC( TOKEN$ADDRESS, TABLE$ADDRESS )    BYTE    PUBLIC;

            DCL     (TOKEN$ADDRESS, TABLE$ADDRESS )         ADDRESS;
            DCL     ( TOKEN BASED TOKEN$ADDRESS ) ( 16 )     BYTE;
            DCL     ( TABLE BASED TABLE$ADDRESS ) ( 256 )    BYTE;
            DCL    ( TOKEN$NUMBER, TAPTR, TOPTR )  BYTE;

        TOKEN$NUMBER, TOPTR, TAPTR = 0;

        CHECK:  DO WHILE ( ( TABLE( TAPTR ) <> '$' ) OR
               ( TABLE( TAPTR + 1 ) <> '$' ) ) AND
                ( TOPTR < 16 );
            MATCH:  DO WHILE TOKEN( TOPTR ) = TABLE(TAPTR );
                    IF ( ( TOKEN( TOPTR := TOPTR + 1 ) = '$' ) OR
                            ( TABLE( TAPTR := TAPTR + 1 ) = '$' ) )
                                  THEN
                                  RETURN TOKEN$NUMBER;

            END MATCH;

            TOPTR = 0;
            DO WHILE TABLE( TAPTR := TAPTR + 1 ) <> '$';
            END; /* FIND END OF CURRENT ENTRY IN TABLE */
```

```
            IF TABLE( TAPTR + 1 ) <> '$'
                    THEN
                    TAPTR = TAPTR + 1; /* FIRST OF NEXT ENTRY */
            TOKEN$NUMBER = TOKEN$NUMBER + 1;
    END CHECK;

    RETURN TOKEN$NUMBER;

    END SEARCH;
```

```
FIND$BLANK: '

    PROC( BUFFER$ADDRESS)   BYTE    PUBLIC;

          DCL     BUFFER$ADDRESS                               ADDRESS;
          DCL     COUNT                                        BYTE;
          DCL     (BUFFER BASED BUFFER$ADDRESS)(160)           BYTE;

    COUNT = -1;
    DO WHILE (BUFFER(COUNT:=COUNT + 1) <> ' ') AND (BUFFER(COUNT) <> CR);
    END; /* DO WHILE */
    RETURN COUNT;

END FIND$BLANK;
END MISCELLANEOUS;
```

```
FILES:
                /* SAVES MEMORY SPECIFIED ON FILE SPECIFIED */
DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSMEM.EXT )

      DCL      ALPHA$WS( 2562 )        BYTE    EXTERNAL;
      DCL      VECTOR$WS( 6146 )       BYTE     EXTERNAL;
```

```
GET$FILE:
                /* GET FILENAME AND OPEN FILE */

        PROCEDURE( ACCESS ) ADDRESS PUBLIC;

                DCL     BUFFER( 128 )   BYTE;
                DCL     BPTR    BYTE;
                DCL     ( AFT, ACCESS, STATUS ) ADDRESS;

        AFT = 0;
        STATUS = 1;

       DO WHILE STATUS <> 0;

                DO BPTR = 0 TO 125;
                        BUFFER( BPTR ) = ' ';
                END;    /* DO BPTR */

                BUFFER( 126 ), BUFFER( 127 ) = '$';

                CALL WRITE$LINE$CRT( .( CR, LF, 'FILENAME? $$') );

                CALL READ$LINE$CRT( .BUFFER );
                IF BUFFER( 0 ) = ESCAPE THEN RETURN 0;
                CALL OPEN( .AFT, .BUFFER, ACCESS, 0, .STATUS );
                IF STATUS <> 0 THEN
                        DO;
                        CALL ERROR( STATUS );
                        CALL WRITE$LINE$CRT( .('UNABLE TO OPEN FILE', CR, LF, '$$' ) );
                END;    /* IF STATUS */
        END;    /* WHILE 0 */

        RETURN AFT;
```

```
    END GET$FILE;
```

```
SAVE$VM:
               /* WRITE VECTOR MEMORY TO FILE */

       PROC( AFTN ) PUBLIC;

               DCL       ( AFTN, COUNT, STATUS ) ADDRESS;

       COUNT = LAST( VECTOR$WS );

       CALL WRITE( AFTN, .VECTOR$WS, COUNT, .STATUS );
       IF STATUS <> 0 THEN
               DO;
               CALL ERROR( STATUS );
               CALL WRITE$LINE$CRT( .('WRITE ERROR ON FILE', CR, LF, '$$' ) );
               END;     /* IF STATUS */

       END SAVE$VM;
```

```
SAVE$AM:
             /* WRITE ALPHA MEMORY TO FILE */

        PROCEDURE( AFTN ) PUBLIC;

             DCL      ( AFTN, COUNT, STATUS ) ADDRESS;

        COUNT = LAST( ALPHA$WS );

        CALL WRITE( AFTN, .ALPHA$WS, COUNT, .STATUS );
        IF STATUS <> 0 THEN
             DO;
             CALL ERROR( STATUS );
             CALL WRITE$LINE$CRT( .('WRITE ERROR ON FILE', CR, LF, '$$' ) );
             END;     /* IF STATUS */

        END SAVE$AM;
```

```
SAVE$M:
                /* SELECTS PROPER MEMORY FOR LOADING */

        PROC( AFTN ) PUBLIC;

                DCL     AFTN    ADDRESS;
                DCL     BPTR    BYTE;
                DCL     BUFFER( 128 )   BYTE;

        DO BPTR = 0 TO 125;
                BUFFER( BPTR ) = ' ';
        END;    /* CLEAR BUFFER */

        BUFFER( 126 ), BUFFER( 127 ) = '$';

        CALL WRITE$LINE$CRT( .( CR, LF, 'VECTOR MEMORY? $$' ));
        CALL READ$LINE$CRT( .BUFFER );
        IF BUFFER( 0 ) = ESCAPE THEN RETURN;
        BPTR = DETRASH( .BUFFER );
        IF BUFFER( BPTR ) = 'Y' THEN
                CALL SAVE$VM( AFTN );
        ELSE CALL SAVE$AM( AFTN );

        END SAVE$M;
```

```
LOAD$M:
             /* LOAD FILE TO MEMORY */

      PROC( AFTN ) PUBLIC;

             DCL      ( AFTN, COUNT, ACTUAL, STATUS ) ADDRESS;
             DCL       BUFFER( 128 )    BYTE;
             DCL      BPTR      BYTE;

      DO BPTR = 0 TO 125;
             BUFFER( BPTR ) = ' ';
      END;   /* CLEAR BUFFER */

      BUFFER( 126 ), BUFFER( 127 ) = '$';

      CALL WRITE$LINE$CRT( .('VECTOR MEMORY? $$' ) );
      CALL READ$LINE$CRT( .BUFFER );
      IF BUFFER( 0 ) = ESCAPE THEN RETURN;
      BPTR = DETRASH( .BUFFER );
      IF BUFFER( BPTR ) = 'Y' THEN
             DO;
             COUNT = LAST( VECTOR$WS );
             CALL READ( AFTN, .VECTOR$WS, COUNT, .ACTUAL, .STATUS );
             IF STATUS <> 0 THEN CALL ERROR( STATUS );
             RETURN;
             END;    /* LOAD VECTOR MEMORY */

      COUNT = LAST( ALPHA$WS );
      CALL READ( AFTN, .ALPHA$WS, COUNT, .ACTUAL, .STATUS );
      IF STATUS <> 0 THEN CALL ERROR( STATUS );

      END LOAD$M;
```

```
FILE$ACCESS:
             /* SET ACCESS MODE FOR ISIS */
     PROC ADDRESS PUBLIC;

             DCL      ( BPTR, TOKEN ) BYTE;
             DCL      BUFFER( 128 )  BYTE;
             DCL      ACCESS$TABLE( * )        BYTE
                      DATA( 'READ$', 'WRITE$', 'UPDATE$', '$$' );

     TOKEN = 255;

     DO WHILE TOKEN > 2;

             DO BPTR = 0 TO 125;
                      BUFFER( BPTR ) = ' ';
             END;     /* CLEAR BUFFER */

             BUFFER( 126 ), BUFFER( 127 ) = '$';

             CALL WRITE$LINE$CRT( .('READ, WRITE, OR UPDATE? $$' ) );
             CALL READ$LINE$CRT( .BUFFER );
             IF BUFFER( 0 ) = ESCAPE THEN RETURN 0;
             BPTR = DETRASH( .BUFFER );

             TOKEN = SEARCH( .BUFFER( BPTR ), .ACCESS$TABLE );
     END;    /* TOKEN > 2 */

     RETURN TOKEN + 1;

     END FILE$ACCESS;
```

```
UPDATE$FILE:
                /* ALLOWS IN PLACE UPDATE OF FILE */

      PROC( AFTN ) PUBLIC;

                DCL      AFTN     ADDRESS;

      CALL WRITE$LINE$CRT( .( CR, LF, 'UPDATE NOT FULLY IMPLEMENTED' ) );

      CALL SAVE$M( AFTN );

      END UPDATE$FILE;
```

```
FILE$HANDLER:
          /* ALLOWS MULTIPLE SETS OF DATA ON ONE FILE */

    PROC( MORE ) PUBLIC;

          DCL      ( MORE, BPTR )   BYTE;
          DCL      ( ACCESS, FILE$NUMBER, STATUS )        ADDRESS;
          DCL      BUFFER( 128 )    BYTE;

    CALL WRITE$LINE$CRT( .( CR, LF, 'MULTIPLE ESCAPE/CRS TERMINATE $$' ) );

    IF NOT MORE THEN
          DO;
          ACCESS = FILE$ACCESS;
          FILE$NUMBER = GET$FILE( ACCESS );
          MORE = TRUE;
          END;     /* NOT MORE */

    DO WHILE MORE;

    IF ACCESS = 1 THEN CALL LOAD$M( FILE$NUMBER );
    ELSE IF ACCESS = 2 THEN CALL SAVE$M( FILE$NUMBER );
          ELSE IF ACCESS = 3 THEN CALL UPDATE$FILE( FILE$NUMBER );
    ELSE CALL WRITE$LINE$CRT( .( CR, LF, 'ILLEGAL ACCESS $$' ) );

    CALL WRITE$LINE$CRT( .( CR, LF, 'MORE? $$' ) );
    CALL READ$LINE$CRT( .BUFFER );
    BPTR = DETRASH( .BUFFER );
    IF BUFFER( BPTR ) = 'Y' THEN
          MORE = TRUE;
    ELSE MORE = FALSE;
    END;     /* WHILE MORE */
```

```
      CALL CLOSE( FILE$NUMBER, .STATUS );

      END FILE$HANDLER;

END FILES;
```

```
/*DETRASH: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )

      DCL      BUFFER(123)     BYTE;
      DCL      BPTR             BYTE;

 DETRASH:
                      /* TRASH REMOVING PROCEDURE */
      PROC ( BUFFER$ADDRESS ) BYTE;

              DCL      ( BUFFER$ADDRESS, POINTER )     ADDRESS;
              DCL      ( BUFFER BASED BUFFER$ADDRESS) ( 123 )   BYTE;

      POINTER = 0;

      DO WHILE (POINTER < 120) AND ((BUFFER(POINTER) = COMMA) OR
              (BUFFER(POINTER) = ' ') OR (BUFFER(POINTER) = TAB) OR
              ( BUFFER( POINTER ) = SEMI$COLON ) );
              POINTER = POINTER + 1;
              POINTER = POINTER + 1;
      END; /* DO WHILE */

      RETURN POINTER;

      END DETRASH;

BUFFER(121), BUFFER(122) = '$';
DO FOREVER;
BPTR = 0;
```

```
CALL READ$LINE$CRT( .BUFFER );
DO WHILE BPTR < 120;
BPTR =BPTR + DETRASH( .BUFFER(BPTR));
CALL WRITE$LINE$CRT( .BUFFER(BPTR));
BPTR=BPTR + 10;
END;
END;

END;   /* DETRASH  */
```

```
FIND:          /* TEST PROGRAM FOR SEARCH ROUTINE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISS.EXT )

        DCL      TABLE( 256 )     BYTE;
        DCL      TPTR     BYTE;

        DCL HEX ADDRESS;
        DCL BUFFER( 123 ) BYTE;
        DCL BPTR BYTE;
        DCL NUMBER( 5 ) BYTE;
        DCL NPTR BYTE;
        DCL      TN       BYTE;
```

```
SEARCH:
      PROC( TOKEN$ADDRESS, TABLE$ADDRESS ) BYTE PUBLIC;

      /*      SEQUENTIALLY SEARCH A TABLE OF VARIABLE LENGTH
      TOKENS OF THE FORM "TOKEN$" UNTIL A MATCH IS
      FOUND OR THE END OF TABLE "$$".  RETURN THE TOKEN NUMBER
      RELATIVE TO THE START OF THE TABLE OR NUMBER OF TOKENS PLUS ONE.
      MATCH IS INDICATED WHEN CHARACTERS ARE EQUAL THROUGH THE
      FIRST "$" ON EITHER STRING.  */

              DCL     (TOKEN$ADDRESS, TABLE$ADDRESS ) ADDRESS;
              DCL     ( TOKEN BASED TOKEN$ADDRESS ) ( 16 )   BYTE;
              DCL     ( TABLE BASED TABLE$ADDRESS ) ( 256 )   BYTE;
              DCL     ( TOKEN$NUMBER, TAPTR, TOPTR )  BYTE;

      TOKEN$NUMBER, TOPTR, TAPTR = 0;

      CHECK:  DO WHILE ( ( TABLE( TAPTR ) <> '$' ) OR
              ( TABLE( TAPTR + 1 ) <> '$' ) ) AND
              ( TOPTR < 16 );
             MATCH:  DO WHILE TOKEN( TOPTR ) = TABLE(TAPTR );
                       IF ( ( TABLE( TAPTR := TAPTR + 1 ) = '$' )
                              OR ( TOKEN( TOPTR := TOPTR + 1 ) = '$' ) )
                              THEN
                                     RETURN TOKEN$NUMBER;

                       TAPTR = TAPTR + 1;
                       TOPTR = TOPTR + 1;

              END MATCH;

              TOPTR = 0;
              DO WHILE TABLE( TAPTR := TAPTR + 1 ) <> '$';
```

```
                END; /* FIND END OF CURRENT ENTRY IN TABLE */

                IF TABLE( TAPTR + 1 ) <> '$'
                        THEN
                        TAPTR = TAPTR + 1; /* FIRST OF NEXT ENTRY */
                TOKEN$NUMBER = TOKEN$NUMBER + 1;
        END CHECK;

        RETURN TOKEN$NUMBER;

        END SEARCH;

DO FOREVER;
CALL READ$LINE$CRT( .TABLE );
TN=0;
CALL READ$LINE$CRT( .BUFFER );
TN = SEARCH( .BUFFER, .TABLE );
HEX= DISPLAY$HEXADECIMAL( TN );
CALL WRITE$LINE$CRT( HEX );
CALL WRITE$LINE$CRT( .TABLE );
END;
END FIND;
```

```
CONVERT$HEXADECIMAL:
           PROC( CONVERT ) ADDRESS;
                 DCL DIGITS  BYTE;
                 DCL CONVERT ADDRESS;
                 DCL (NUMBER BASED CONVERT) (5) BYTE;
                 DCL HEX ADDRESS;

     HEX = 0;
     DIGITS = 0;

     DO WHILE  ( NUMBER( DIGITS ) >= '0' ) AND ( NUMBER( DIGITS ) <= '9' );
           HEX = ( HEX * 10 ) + ( NUMBER( DIGITS ) AND  0FH  );
           DIGITS = DIGITS + 1;
           IF DIGITS > 5
                 THEN DO;
                       CALL WRITE$LINE$CRT( .('NUMBER TOO LARGE',
                             CR, LF, '$$' ) );
                       RETURN HEX;
                 END;  /* THEN DO */
     END;     /* DO WHILE */

     RETURN HEX;
END CONVERT$HEXADECIMAL;
```

```
/*EXTERNALS: */

 WRITE$DOT:

      PROCEDURE(X, Y) EXTERNAL;
              DCL       (X, Y)    ADDRESS;
      END;   /* WRITE$DOT */


 ERASE$DOT:

      PROCEDURE(X, Y) EXTERNAL;
              DCL       (X, Y)    ADDRESS;
      END;   /* ERASE$DOT */


 WRITE$CURSOR:

      PROCEDURE(X, Y) EXTERNAL;
              DCL       (X, Y)    ADDRESS;
      END;   /* WRITE$CURSOR */


 ERASE$CURSOR:

      PROCEDURE(X, Y) EXTERNAL;
              DCL       (X, Y)    ADDRESS;
      END;   /* ERASE$CURSOR */


 WRITE$CURSOR$LOCATION:

      PROCEDURE(POSITION) EXTERNAL;
```

```
             DCL     POSITION         ADDRESS;
        END;   /* WRITE$CURSOR$LOCATION */
```

```
/*WRITE$DOT: */
          /* GENERATE VECTOR CURSOR AND DRIVE CURSOR FROM CRT CONSOLE. */
                 /* EXIT FROM CURSOR WITH A 'CONTROL X.'  CURSOR COORDINATES  */
                 /* APPEAR ON CRT ON EXIT FROM ROUTINE.                       */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )


 WRITE$DOT:   /* WRITE DOT TO PLASMA */


PROCEDURE(X,Y) PUBLIC;

      DCL      X        ADDRESS;
      DCL      Y        ADDRESS;

CALL WRITE$P( CG );
CALL WRITE$P( LOW( X ) AND 7FH );
CALL WRITE$P( LOW( Y ) AND 7FH );
CALL WRITE$P( HIGH(SHL(( Y AND 0180H), 3 ) )
      OR ( HIGH ( SHL(( X AND 0180H ), 1) ) ) );
CALL WRITE$P(ETX);

END;  /* WRITE$DOT */
```

```
ERASE$DOT:                /* ERASE DOT ON PLASMA */

PROCEDURE(X,Y) PUBLIC;

        DCL      X        ADDRESS;
        DCL      Y        ADDRESS;

CALL WRITE$P( CG );
CALL WRITE$P( LOW( X ) AND 7FH );
CALL WRITE$P( LOW( Y ) AND 7FH );
CALL WRITE$P( HIGH(SHL(( Y AND 0180H), 3 )  )
        OR ( HIGH ( SHL(( X AND 0180H), 1) ) ) OR 40H );
CALL WRITE$P(ETX);

END;  /* ERASE$DOT */
```

```
WRITE$CURSOR:           /* WRITE VECTOR CURSOR ON PLASMA */

PROCEDURE (X$COORDINATE, Y$COORDINATE) PUBLIC;

        DCL      (X$COORDINATE, Y$COORDINATE)    ADDRESS;

CALL WRITE$DOT(X$COORDINATE - 1, Y$COORDINATE);
CALL WRITE$DOT(X$COORDINATE + 1, Y$COORDINATE);
CALL WRITE$DOT(X$COORDINATE, Y$COORDINATE - 1);
CALL WRITE$DOT(X$COORDINATE, Y$COORDINATE + 1);

END;   /* WRITE$CURSOR */
```

```
ERASE$CURSOR:                    /* ERASE LAST WRITTEN VECTOR CURSOR */

PROCEDURE (X$COORDINATE, Y$COORDINATE) PUBLIC;

      DCL      (X$COORDINATE, Y$COORDINATE)    ADDRESS;

CALL ERASE$DOT(X$COORDINATE - 1, Y$COORDINATE);
CALL ERASE$DOT(X$COORDINATE + 1, Y$COORDINATE);
CALL ERASE$DOT(X$COORDINATE, Y$COORDINATE - 1);
CALL ERASE$DOT(X$COORDINATE, Y$COORDINATE + 1);

END;   /* ERASE$CURSOR */
```

```
WRITE$CURSOR$LOCATION:  /* WRITE CURSOR LOCATION TO CRT */

PROCEDURE(POSITION) PUBLIC;

      DCL     POSITION        ADDRESS;
      DCL     LOCATION(3)    BYTE;
      DCL     INDEX           BYTE;

LOCATION(0) = ((POSITION/100) OR 30H);
LOCATION(1) = (((POSITION MOD 100)/10) OR 30H);
LOCATION(2) = (((POSITION MOD 100) MOD 10) OR 30H);

DO INDEX = 0 TO 2;
      CALL WRITE$CRT(LOCATION(INDEX));
END;  /* DO */

END;  /* WRITE$CURSOR$POSITION */

END;  /* CURSOR */
```

```
/*MOVE$CURSOR:                  MOVE COMPUTER GENERATED CURSOR */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CURSOR.EXT )

 MOVE$CURSOR:

      PROCEDURE PUBLIC;

      DCL     CHAR            BYTE;

CALL WRITE$LINE$CRT( .(CR,LF,'FORWARD SPACE IS CTL-F',
                       CR,LF,'BACKSPACE IS CTL-H',
                       CR,LF,'LINE FEED IS CTL-J',
                       CR,LF,'VERTICAL TAB IS CTL-K','$$') );
CHAR = ' ';
CALL WRITE$CURSOR( X$VECTOR, Y$VECTOR );
DO WHILE (CHAR <> ESCAPE);
      CHAR = READ$CRT;
      CALL ERASE$CURSOR( X$VECTOR, Y$VECTOR);
      IF ((CHAR = FS) AND (X$VECTOR < 510))
              THEN X$VECTOR = X$VECTOR + 1;
      IF ((CHAR = BS) AND (X$VECTOR > 1))
              THEN X$VECTOR = X$VECTOR - 1;
      IF ((CHAR = VT) AND (Y$VECTOR > 1))
              THEN Y$VECTOR = Y$VECTOR - 1;
      IF ((CHAR = LF) AND (Y$VECTOR < 510))
              THEN Y$VECTOR = Y$VECTOR + 1;
```

```
      IF (CHAR = CH) THEN X$VECTOR, Y$VECTOR = 1;
      CALL WRITE$CURSOR( X$VECTOR, Y$VECTOR);
END;  /* DO WHILE */

CALL WRITE$LINE$CRT( .(CR,LF,'X = ','$$') );
CALL WRITE$CURSOR$LOCATION(X$VECTOR);
CALL WRITE$LINE$CRT( .(CR,LF,'Y = ','$$') );
CALL WRITE$CURSOR$LOCATION(Y$VECTOR);
CALL WRITE$LINE$CRT( .(CR,LF,'$$') );
CALL ERASE$CURSOR( X$VECTOR,Y$VECTOR);

END MOVE$CURSOR;

END MVCUR;
```

```
/*EXTERNALS: */
      MOVE$WORD:
            PROC( FROM, TOO ) EXTERNAL;
                  DCL     ( FROM, TOO )   ADDRESS;
      END MOVE$WORD;

      DISPLAY$VECTOR$ATTRIBUTES:
            PROC( VECTOR$ADDRESS ) EXTERNAL;
                  DCL     VECTOR$ADDRESS  ADDRESS;
      END DISPLAY$VECTOR$ATTRIBUTES;

      GET$TOKEN:
            PROC( BUFFER$ADDRESS, BPTR$ADDRESS, TABLE$ADDRESS ) BYTE EXTERNAL;
                  DCL     ( BUFFER$ADDRESS, BPTR$ADDRESS )        ADDRESS;
                  DCL     TABLE$ADDRESS   ADDRESS;
      END GET$TOKEN;

      GET$X:
            PROC( ASCII$ADDRESS, VECTOR$ADDRESS ) BYTE EXTERNAL;
                  DCL     ( ASCII$ADDRESS, VECTOR$ADDRESS )       ADDRESS;
      END GET$X;

      GET$Y:
            PROC( ASCII$ADDRESS, VECTOR$ADDRESS ) BYTE EXTERNAL;
                  DCL     ( ASCII$ADDRESS, VECTOR$ADDRESS )       ADDRESS;
      END GET$Y;

      CGS:
            PROC EXTERNAL;
      END CGS;

      MOVE$CURSOR:
           PROCEDURE EXTERNAL;
```

```
        END MOVE$CURSOR;
```

```
/*MOVE$WORD: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CG.DCL )

$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

 MOVE$WORD:
      /* MOVES BYTES FROM ADDRESS GIVEN TO LOCATION AT "TOO" UNTIL A DOLLAR SIGN
      IS ENCOUNTER IN EITHER SOURCE OR DESTINATION FIELD.  */

      PROC( FROM, TOO ) PUBLIC;
              DCL       ( FROM, TOO )    ADDRESS;
              DCL       ( SOURCE BASED FROM ) ( 123 )    BYTE;
              DCL       ( DESTINATION BASED TOO ) ( 123 )       BYTE;
              DCL       POINTER BYTE;

      POINTER = 0;

      DO WHILE ( SOURCE( POINTER ) <> '$' ) AND
              ( DESTINATION( POINTER ) ) <> '$';
              DESTINATION( POINTER ) = SOURCE( POINTER );
              POINTER = POINTER + 1;
      END;    /* DO WHILE */

      END MOVE$WORD;
```

```
DISPLAY$VECTOR$ATTRIBUTES:
      /******* U S E S   A   G L O B A L   T O K E N   T A B L E *******/
      /* DUMPS A LINE ON THE CRT REVEALING THE VECTOR ATTRIBUTES REQUESTED BY
      THE THREE BYTE FIELD LOCATED AT VECTOR ADDRESS. */

      PROC( VECTOR$ADDRESS ) PUBLIC;
              DCL     VECTOR$ADDRESS  ADDRESS;
              DCL      ( VECTOR BASED VECTOR$ADDRESS ) ( 5 )    BYTE;
              DCL      QUERY( 40 )      BYTE;
              DCL      ( DIGITS, POINTER )     BYTE;
              DCL      NUMBER  ADDRESS;
              DCL      ASCII$NUMBER( 10 )       BYTE;

      DO POINTER = 0 TO LAST( QUERY );
              QUERY( POINTER ) = ' ';
      END;

      QUERY( LAST( QUERY ) - 1 ), QUERY( LAST( QUERY ) ) = '$';

      CALL MOVE$WORD( .TOKEN$TABLE( 0 ), .QUERY( 0 ) );          /* X= */
      NUMBER = VECTOR( 2 );
      NUMBER = SHL( ( NUMBER AND X$MASK ), 7 ) + VECTOR( 0 );
      CALL DISPLAY$DECIMAL( NUMBER, .ASCII$NUMBER );
      DIGITS = 0;

      DO WHILE DIGITS < 4;
              QUERY( 6 - DIGITS ) = ASCII$NUMBER( 9 - DIGITS );
              DIGITS = DIGITS + 1;
      END;    /* DO WHILE */

      IF NUMBER = 0 THEN QUERY( 6 ) = '0';
      CALL MOVE$WORD( .TOKEN$TABLE( 3 ), .QUERY( 8 ) );          /* Y= */
      NUMBER = VECTOR( 2 );
```

```
        NUMBER = SHL( ( NUMBER AND Y$MASK ), 5 ) + VECTOR( 1 );
        CALL DISPLAY$DECIMAL( NUMBER, .ASCII$NUMBER );
        DIGITS = 0;

        DO WHILE DIGITS < 4;
                QUERY( 14 - DIGITS ) = ASCII$NUMBER( 9 - DIGITS );
                DIGITS = DIGITS + 1;
        END;    /* DO WHILE */

        IF NUMBER = 0 THEN QUERY( 14 ) = '0';
        IF ( VECTOR( 2 ) AND SET$ERASE ) = SET$ERASE
                THEN POINTER = 12;              /* ERASE */
               ELSE POINTER =  6;              /* WRITE */
        CALL MOVE$WORD( .TOKEN$TABLE( POINTER ), .QUERY( 16 ) );
        IF ( VECTOR( 2 ) AND SET$DASHED ) = SET$DASHED
                THEN POINTER = 25;              /* DASHED */
                ELSE POINTER = 18;              /* SOLID */
        CALL MOVE$WORD( .TOKEN$TABLE( POINTER ), .QUERY( 22 ) );
        IF ( VECTOR( 2 ) AND SET$END ) = SET$END
                THEN POINTER = 38;              /* END */
                ELSE POINTER = 32;              /* START */
       CALL MOVE$WORD( .TOKEN$TABLE( POINTER ), .QUERY( 29 ) );
        IF ( VECTOR( 2 ) AND Q$MASK ) = Q$MASK
                THEN POINTER = 44;              /* Q */
                ELSE POINTER = 46;              /* NQ */
        CALL MOVE$WORD( .TOKEN$TABLE( POINTER ), .QUERY( 35 ) );

        CALL WRITE$LINE$CRT( .QUERY );

        END DISPLAY$VECTOR$ATTRIBUTES;
```

```
GET$TOKEN:    /* FIND TOKEN NUMBER IN GLOBAL TOKEN TABLE */
    PROC( BUFFER$ADDRESS, BPTR$ADDRESS, TOKEN$TABLE$ADDRESS ) BYTE PUBLIC;
        DCL     ( BUFFER$ADDRESS, BPTR$ADDRESS, TOKEN$TABLE$ADDRESS )   ADDRESS;
        DCL     ( BUFFER BASED BUFFER$ADDRESS ) ( 100 )         BYTE;
        DCL     ( TOKEN$TABLE BASED TOKEN$TABLE$ADDRESS ) ( 52 )         BYTE;
        DCL     BPTR BASED BPTR$ADDRESS         BYTE;
        DCL     ( TPTR, TOKEN$NUMBER )           BYTE;

    DO TPTR = 0 TO LAST( TOKEN );
        TOKEN( TPTR ) = '$';
    END; /* TPTR */

    TPTR = 0;
    DO WHILE ( ( BUFFER( BPTR ) <> ';' )
        AND ( BUFFER( BPTR ) <> ',' )
        AND ( BUFFER( BPTR ) <> ' ' )
        AND ( BUFFER( BPTR ) <> CR  )
        AND ( BUFFER( BPTR ) <> '$' )
        AND ( TPTR < ( LAST( TOKEN ) - 1 ) ) );

        TOKEN( TPTR ) = BUFFER( BPTR );
        TPTR = TPTR + 1;
        BPTR = BPTR + 1;
    END;    /* DO WHILE NOT DELIMITER */

    TOKEN$NUMBER = SEARCH( .TOKEN, .TOKEN$TABLE );
    RETURN TOKEN$NUMBER;

    END GET$TOKEN;
```

```
GET$X:
     PROC( TOKEN$ADDRESS, VECTOR$ADDRESS ) BYTE PUBLIC;
          DCL     ( TOKEN$ADDRESS, VECTOR$ADDRESS )       ADDRESS;
          DCL     ( NUMBER BASED TOKEN$ADDRESS ) ( 16 )    BYTE;
          DCL     ( VECTOR BASED VECTOR$ADDRESS ) ( 3 )    BYTE;
          DCL     NPTR     BYTE;
          DCL     X        ADDRESS;

     NPTR = 0;

     DO WHILE ( ( NUMBER( NPTR ) < '0' ) OR ( NUMBER( NPTR ) > '9' ))
          AND ( NPTR < 15 );
          NPTR = NPTR + 1;
     END;

     VECTOR( 2 ) = VECTOR( 2 ) AND NOT X$MASK;
     VECTOR( 0 ) = 0;
     X = CONVERT$HEXADECIMAL( .NUMBER( NPTR ) );
     IF X < 512
          THEN DO;
               VECTOR( 0 ) = LOW( X ) AND 0111$1111B;
               VECTOR( 2 ) = VECTOR( 2 ) OR ( ( HIGH( SHL( X, 1 ) ) )
                                   AND X$MASK );
               NO$X = FALSE;
               RETURN TRUE;
          END;     /* THEN DO */
          ELSE
               RETURN FALSE;

     END GET$X;
```

```
GET$Y:
      PROC( TOKEN$ADDRESS, VECTOR$ADDRESS ) BYTE PUBLIC;
              DCL      ( TOKEN$ADDRESS, VECTOR$ADDRESS )        ADDRESS;
             DCL      ( NUMBER BASED TOKEN$ADDRESS) ( 16 )     BYTE;
              DCL      ( VECTOR BASED VECTOR$ADDRESS ) ( 3 )    BYTE;
              DCL      NPTR     BYTE;
              DCL      Y        ADDRESS;

      NPTR = 0;

      DO WHILE ( ( NUMBER( NPTR ) < '0' ) OR ( NUMBER( NPTR ) > '9' ) )
              AND ( NPTR < 15 );
              NPTR = NPTR + 1;
      END;

      VECTOR( 2 ) = VECTOR( 2 ) AND NOT Y$MASK;
      VECTOR( 1 ) = 0;
      Y = CONVERT$HEXADECIMAL( TOKEN$ADDRESS + NPTR );
      IF Y < 512
              THEN DO;
                      VECTOR( 1 ) = LOW( Y ) AND 0111$1111B;
                      VECTOR( 2 ) = VECTOR( 2 ) OR ( ( HIGH( SHL( Y, 3 ) ) )
                                      AND Y$MASK );
                      NO$X = TRUE;
                      RETURN TRUE;
              END;    /* THEN DO */
              ELSE
                      RETURN FALSE;

      END GET$Y;
```

```
CGS:
                /* CGS ACCEPTS INPUT FROM THE CONSOLE AND PASSES TO THE
                 PLASMA DISPLAY.  INITIAL DEFAULTS ARE CARRIED FORWARD UNLESS
                 CHANGED.  AN ESCAPE TERMINATES THIS PROGRAM.           */

        PROC PUBLIC;

/*      CONSTRUCT GRAPHS (CG) MAIN PROGRAM       */
/* SET DEFAULTS */

        IF CGS$FIRST THEN
                DO;
                CGS$FIRST = FALSE;
                X, Y = 0;
                VECTOR( 0 ) = CG;
                VECTOR( 1 ), VECTOR( 2 ), VECTOR( 3 ), VECTOR( 4 ) = 00H;
                BUFFER( 121 ), BUFFER( 122 ), BUFFER( 123 ) = '$';
                CHAR = ' ';
                END;     /* CGS$FIRST */

CALL WRITE$LINE$CRT( .(CR, LF, '           CONSTRUCT GRAPHS', CR, LF, '$$' ) );

 VECTORS: DO WHILE ( CHAR <> 'N' ) AND ( CHAR <> 'N' );

        CALL WRITE$LINE$CRT( .('X=    , Y=    , WR/ER, SO/DA, ST/EN, Q/NQ', '$$' ) );
        NO$X = TRUE;
        CONTINUE = FALSE;

        BUILD$VECTOR: DO WHILE CONTINUE = FALSE;

                CALL WRITE$LINE$CRT( .( CR, LF, '$$' ) );
                DO BPTR = 0 TO BUFF$SIZE;
                        BUFFER( BPTR ) = ' ';
```

```
          END;

          CALL READ$LINE$CRT( .BUFFER );
          BPTR = DETRASH( .BUFFER( 0 ) );
                  /* SET QUERY MODE IF NO INPUT */
          IF ( BUFFER( BPTR ) = CR ) OR ( BUFFER( BPTR ) = LF ) THEN
                  VECTOR( 3 ) = VECTOR( 3 ) OR Q$MASK;

                  GET$PARMS: DO WHILE ( ( BPTR := BPTR +
                                 DETRASH( .BUFFER( BPTR ) ) ) < 120 )
                          AND ( ( BUFFER( BPTR ) <> '$' )
                          OR ( BUFFER( BPTR + 1 ) <> '$' ) )
                          AND ( BUFFER( BPTR ) <> CR );

                          TOKEN$NUMBER = GET$TOKEN( .BUFFER, .BPTR, .TOKEN$TABLE );

                          FIND$TOKEN: DO CASE TOKEN$NUMBER;

      /* CASE 0:  GET X */    IF ( NOT GET$X( .TOKEN, .VECTOR( 1 ) ) ) THEN
                                 DO;
                                 CALL WRITE$LINE$CRT( .('X TOO LARGE',
                                         CR, LF, '$$' ) );
                                 VECTOR( 3 ) = VECTOR( 3 ) OR Q$MASK;
                                 END;

      /* CASE 1:  GET Y */    IF ( NOT GET$Y( .TOKEN, .VECTOR( 1 ) ) ) THEN
                                 DO;
                                 CALL WRITE$LINE$CRT( .('Y TOO LARGE',
                                         CR, LF, '$$' ) );
                                 VECTOR( 3 ) = VECTOR( 3 ) OR Q$MASK;
                                 END;

      /* CASE 2:  WRITE */    VECTOR( 3 ) = VECTOR( 3 ) AND NOT SET$ERASE;
```

```
/* CASE 3:  ERASE */   VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
/* CASE 4:  SOLID */    VECTOR( 3 ) = VECTOR( 3 ) AND NOT SET$DASHED;
/* CASE 5:  DASHED */  VECTOR( 3 ) = VECTOR( 3 ) OR SET$DASHED;
/* CASE 6:  START */    VECTOR( 3 ) = VECTOR( 3 ) AND NOT SET$END;
/* CASE 7:  END */     VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;
/* CASE 8:  QUERY */    VECTOR( 3 ) = VECTOR( 3 ) OR Q$MASK;
/* CASE 9:  NO QUERY */        VECTOR( 3 ) = VECTOR( 3 ) AND NOT Q$MASK;
/* CASE 10:  INVALID */ DO;
                        IF ( ( TOKEN( 0 ) >= '0' ) AND
                                ( TOKEN( 0 ) <= '9' ) ) THEN
                                DO;
                                IF NO$X THEN
                                        DO;
                                        IF GET$X( .TOKEN, .VECTOR( 1 ) ) THEN
                                                NO$X = FALSE;
                                        END;
                                ELSE IF GET$Y( .TOKEN, .VECTOR( 1 ) ) THEN NO$X = TRUE;
                                END;

                        IF TOKEN( 0 ) = ESCAPE THEN RETURN;
                        CALL WRITE$LINE$CRT( .TOKEN );
                        CALL WRITE$LINE$CRT( .(' NOT TOKEN', CR, LF, '$$' ) );
                        DO WHILE ( (BUFFER( BPTR := BPTR + 1 ) > '0' )
                                AND ( BUFFER( BPTR ) < '9' ) );
                        END;    /* DO WHILE */

                        END; /* INVALID */

                        END FIND$TOKEN;

                END GET$PARMS;

        CONTINUE = TRUE;
```

```
        CALL DISPLAY$VECTOR$ATTRIBUTES( .VECTOR( 1 ) );
        IF ( VECTOR( 3 ) AND Q$MASK ) = Q$MASK
                THEN DO;
                        CALL WRITE$LINE$CRT( .('?  ( Y/N ) $$' ) );
                        CHAR = READ$CRT;
                        IF CHAR = ESCAPE THEN RETURN;
                        IF ( ( CHAR <> 'Y' ) AND ( CHAR <> 'Y' ) )
                                THEN CONTINUE = FALSE;
                END;    /* THEN DO */
        CALL WRITE$LINE$CRT( .( CR, LF, '$$' ) );

        END BUILD$VECTOR;

        CALL WRITE$VS( .VECTOR );
    END VECTORS;

    END CGS;

END CG1;
```

```
/*EXTERNALS: */

        SET$X:
                PROC( X$HEX, VECTOR$ADDRESS ) EXTERNAL;
                        DCL     ( X$HEX, VECTOR$ADDRESS )       ADDRESS;
                END SET$X;

        SET$Y:
                PROC( Y$HEX, VECTOR$ADDRESS ) EXTERNAL;
                        DCL     ( Y$HEX, VECTOR$ADDRESS )        ADDRESS;
                END SET$Y;

        TRANSLATE:
                PROC( X$ADDRESS, Y$ADDRESS, SET$ORIGIN ) BYTE EXTERNAL;
                        DCL     ( X$ADDRESS, Y$ADDRESS )         ADDRESS;
                        DCL    SET$ORIGIN       BYTE;
                END TRANSLATE;

        ROW:
                PROC( ROW$NUMBER ) EXTERNAL;
                        DCL ROW$NUMBER  ADDRESS;
                END ROW;

        COL:
                PROC( COLUMN$NUMBER ) EXTERNAL;
                        DCL     COLUMN$NUMBER   ADDRESS;
                END COL;
```

```
/*SET$X: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CG.DCL )

$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

 SET$X:
      /* PLACES THE VALUE RECEIVED IN "X" IN PROPER FORMAT FOR
      TRANSMITTING TO THE PLASMA SCOPE AT THE ADDRESS SPECIFIED BY "VECTOR$ADDRESS".
      */

      PROC( X, VECTOR$ADDRESS ) PUBLIC;
              DCL       ( X, VECTOR$ADDRESS )    ADDRESS;
              DCL       ( VECTOR BASED VECTOR$ADDRESS ) ( 3 )    BYTE;

      VECTOR( 2 ) = VECTOR( 2 ) AND NOT X$MASK;
      VECTOR( 0 ) = 0;
      IF X < 512
              THEN DO;
                      VECTOR( 0 ) = LOW( X ) AND 0111$1111B;
                      VECTOR( 2 ) = ( VECTOR(2) AND 1111$1100B) OR ((HIGH(SHL(X,1)))
                                              AND X$MASK );
              END;    /* THEN DO */

      ELSE CALL WRITE$LINE$CRT( .(' X TOO LARGE', CR, LF, '$$' ) );
```

```
    END 'SET$X;
```

```
SET$Y:
    /* PLACES THE VALUE RECEIVED IN "Y" IN PROPER FORMAT FOR THE PLASMA
    SCOPE.
    */

    PROC( Y, VECTOR$ADDRESS ) PUBLIC;
            DCL      ( Y, VECTOR$ADDRESS )    ADDRESS;
            DCL      ( VECTOR BASED VECTOR$ADDRESS ) ( 3 )    BYTE;

    VECTOR( 2 ) = VECTOR( 2 ) AND NOT Y$MASK;
    VECTOR( 1 ) = 0;

    IF Y < 512
            THEN DO;
                    VECTOR( 1 ) = LOW( Y ) AND 0111$1111B;
                    VECTOR( 2 ) = ( VECTOR(2) AND 1111$0011B ) OR ((HIGH(SHL(Y,3)))
                                    AND Y$MASK );
            END;     /* THEN DO */

    ELSE CALL WRITE$LINE$CRT( .(' Y TOO LARGE', CR, LF, '$$' ) );

    END SET$Y;
```

```
TRANSLATE:
    /* RETURNS TRUE IF THE ORIGIN HAS BEEN MOVED TO A LEGITIMATE VALUE,
    OTHERWISE FALSE IS RETURNED.
    THE VALUES RECIEVED AT THE X AND Y ADDRESS ARE ADJUSTED TO THE NEW COORDINATES,
    IF IT EXISTS.
    */

    PROC( X$ADDRESS, Y$ADDRESS, SET ) BYTE PUBLIC;
            DCL     ( X$ADDRESS, Y$ADDRESS )         ADDRESS;
            DCL     SET     BYTE;
            DCL     ( X BASED X$ADDRESS )   ADDRESS;
            DCL     ( Y BASED Y$ADDRESS )   ADDRESS;
            DCL     ( X$OFFSET, Y$OFFSET )  ADDRESS;

    IF XLATE$FIRST THEN
            DO;
            XLATE$FIRST = FALSE;
            X$OFFSET = 0;
            Y$OFFSET = 0;
            END;    /* FIRST */

    IF NOT SET THEN
            DO;
            IF ( X$OFFSET = 0 ) AND ( Y$OFFSET = 0 ) THEN
                    RETURN FALSE;
            X = X + X$OFFSET;
            Y = Y$OFFSET - Y;
            RETURN TRUE;
            END;    /*NOT SET */

    IF ( X > 511 ) OR ( Y > 511 ) THEN
            RETURN FALSE;
    X$OFFSET = X;
```

```
        Y$OFFSET = Y;
        RETURN TRUE;

        END TRANSLATE;
```

```
ROW:
     /* DRAWS A SOLID VECTOR ON THE ROW ( 0 - 511 ) SPECIFIED.  */
     /* ROW NUMBERS OVER 511 ARE REDUCED TO MODULO 511 AND ERASED.  */

     PROC( ROW$NO ) PUBLIC;
             DCL      ROW$NO  ADDRESS;
             DCL      VECTOR( 6 )      BYTE;

     VECTOR( 4 ), VECTOR( 5 ) = '$';
     VECTOR( 0 ) = CG;
     VECTOR( 1 ), VECTOR( 2 ) = 0;
     VECTOR( 3 ) = 80H;          /* WRITE, SOLID, START, 0 */

     IF ROW$NO > 511 THEN
             DO;
             VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
             ROW$NO = ROW$NO AND 01FFH;
             END;
     CALL SET$Y( ROW$NO, .VECTOR( 1 ) );
     CALL WRITE$LINE$PS( .VECTOR );

     VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;
     CALL SET$X( 511, .VECTOR( 1 ) );
     CALL WRITE$LINE$PS( .VECTOR );

     END ROW;
```

```
COL:
      /* DRAWS A SOLID VECTOR IN COLUMN SPECIFIED.  */
      /* VECTORS MAY BE ERASED BY SPECIFYING NUMBERS GREATER THAN 511 */

      PROC( COL$NO ) PUBLIC;
              DCL      COL$NO  ADDRESS;
              DCL      VECTOR( 6 )     BYTE;

      VECTOR( 4 ), VECTOR( 5 ) = '$';
      VECTOR( 0 ) = CG;
      VECTOR( 1 ), VECTOR( 2 ) = 0;
      VECTOR( 3 ) = 80H;        /* WRITE,SOLID,START,0 */

      IF COL$NO > 511 THEN
              DO;
              VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
              COL$NO = COL$NO AND 01FFH;
              END;
      CALL SET$X( COL$NO, .VECTOR( 1 ) );
      CALL WRITE$LINE$PS( .VECTOR );

      VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;
      CALL SET$Y( 511, .VECTOR( 1 ) );
     CALL WRITE$LINE$PS( .VECTOR );

      END COL;

END CG2;
```

```
/*EXTERNALS: */

 INITIALIZE$MEMORY:

         PROCEDURE EXTERNAL;
         END;   /* INITIALIZE$MEMORY */



 ASCHII$PTR:

         PROCEDURE ADDRESS EXTERNAL;
         END;   /* ASCHII$PTR */


 NULL$CHECK:

         PROCEDURE BYTE EXTERNAL;
         END;   /* NULL$CHECK */


 LINE$CHECK:

         PROCEDURE BYTE EXTERNAL;
         END;   /* LINE$CHECK */


 PAGE$CHECK:

         PROCEDURE BYTE EXTERNAL;
         END;   /* PAGE$CHECK */
```

```
INCREMENT$POINTER:

      PROCEDURE BYTE EXTERNAL;
      END;   /* INCREMENT$POINTER */


DECREMENT$POINTER:

      PROCEDURE BYTE EXTERNAL;
      END;   /* DECREMENT$POINTER */


BACKGROUND$STATUS:

      PROCEDURE BYTE EXTERNAL;
      END;   /* BACKGROUND$STATUS */


FIND$FOREGROUND:

      PROCEDURE EXTERNAL;
      END;   /* FIND$EXTERNAL */


FIND$BACKGROUND:

      PROCEDURE BYTE EXTERNAL;
      END;   /* FIND$BACKGROUND */


CURSOR$HOME:

      PROCEDURE EXTERNAL;
```

```
    END;   /* CURSOR$HOME */

FORESPACE:

    PROCEDURE EXTERNAL;
    END;   /* FORESPACE */

BACKSPACE:

    PROCEDURE EXTERNAL;
    END;   /* BACKSPACE */

CURSOR$TAB:

    PROCEDURE EXTERNAL;
    END;   /* CURSOR$TAB */

LINE$FEED:

    PROCEDURE EXTERNAL;
    END;   /* LINE$FEED */

VERTICAL$FEED:

    PROCEDURE EXTERNAL;
    END;   /* VERTICAL$FEED */
```

```
/*PLASMA$SCOPE$MEMORY                              /*EXTERNALS

CARRAGE$RETURN:

      PROCEDURE EXTERNAL;
      END;   /* CARRAGE$RETURN */


CURSOR$ROUTINE:

      PROCEDURE(CHAR) BYTE EXTERNAL;
               DCL CHAR BYTE;
      END;   /* CURSOR$ROUTINE */


STORE$ASCHII$MEMORY:

      PROCEDURE(CHAR) EXTERNAL;
               DCL CHAR BYTE;
      END;   /* STORE$ASCHII$MEMORY */


MEMORY$DUMP:

      PROCEDURE EXTERNAL;
      END;   /* MEMORY$DUMP */
```

```
/*EXTERNALS: */

 VECTOR$DUMP:
      PROCEDURE EXTERNAL;
      END VECTOR$DUMP;


 CHANGE$FOREGROUND$TO$BACKGROUND:

      PROCEDURE EXTERNAL;
      END;   /* CHANGE$FOREGROUND$TO$BACKGROUND */


 CHANGE$BACKGROUND$TO$FOREGROUND:

      PROCEDURE EXTERNAL;
      END;   /* CHANGE$BACKGROUND$TO$FOREGROUND */
```

```
PSMEM:          /* THIS PROCEDURE IS PART OF THE STORE$IN$MEMORY ROUTINE WHICH    */
        /* WRITES INTO MEMORY DATA AS IT IS WRITTEN ON THE PLASMA FROM  */
        /* THE CRT.                                                  */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PS.EXT )

            /* GLOBAL VARIABLES FOR PSMEM */

            DCL     EDIT$MODE                  BYTE     PUBLIC,
                    CURRENT$MODE               BYTE     PUBLIC,
                    BG$FG$MODE                 BYTE     PUBLIC,
                    EXPECTED$BYTES            BYTE     PUBLIC,
                    LINE                       BYTE     PUBLIC,
                    COLUMN                     BYTE     PUBLIC,
                    ASCII$PTR                  ADDRESS  PUBLIC,
                    VECTOR$POINTER             ADDRESS  PUBLIC,
                    ALPHA$WS(2562)             BYTE     PUBLIC,
                    VECTOR$WS(6146)            BYTE     PUBLIC;
```

```
INITIALIZE$MEMORY:  /* CLEARS 2660 BYTES OF ALPHANUMERIC MEMORY AND */
                     /* 6144 BYTES OF VECTOR MEMORY AND SETS INITIAL PARAMETERS */

      PROCEDURE      PUBLIC;

              DCL     INDEX   ADDRESS;

      EDIT$MODE = FALSE;
      CURRENT$MODE = NEUTRAL;
      BG$FG$MODE = FALSE;
      EXPECTED$BYTES = 0;
      LINE = 1;
      COLUMN = 1;
      ASCII$PTR = 0;
      VECTOR$POINTER = 1;

      DO INDEX = 0 TO 2559;
              ALPHA$WS(INDEX) = 00H;
      END;

      ALPHA$WS(2560), ALPHA$WS(2561) = '$';

      DO INDEX = 0 TO 6143;
              VECTOR$WS(INDEX) = 00H;
      END;

      VECTOR$WS(0),VECTOR$WS(6144),VECTOR$WS(6145) = '$';

END;  /* INITIALIZE$MEMORY */
```

```
ASCHII$PTR:  /* RETURNS POSITION OF POINTER IN ALPHANUMERIC MEMORY */

      PROCEDURE ADDRESS PUBLIC;
      RETURN ( ASCII$PTR := (LINE-1) * 80 + COLUMN - 1);

END;  /* ASCHII$PTR */
```

```
NULL$CHECK:  /* CHECKS FOR NULL CHARACTER AT CURRENT POINTER POSITION */

      PROCEDURE BYTE PUBLIC;

      IF(ALPHA$WS(ASCHII$PTR) = NULL) THEN RETURN (TRUE);
              ELSE RETURN (FALSE);

END;  /* NULL$CHECK */
```

```
LINE#CHECK:  /* RETURNS TRUE WHEN THERE IS MORE ROOM AVAILABLE ON CURRENT LINE */

       PROCEDURE BYTE PUBLIC;

       IF( COLUMN > 80 )
               THEN DO;
                       COLUMN = 1;
                       LINE = LINE + 1;
                       RETURN (FALSE);
               END;
       RETURN (TRUE);
END;   /* LINE#CHECK */
```

```
PAGE$CHECK:  /* RETURNS TRUE AT END OF PAGE */

        PROCEDURE BYTE PUBLIC;

        IF( LINE >= 33) THEN RETURN (FALSE);
                ELSE RETURN (TRUE);

END;    /* PAGE$CHECK */
```

```
INCREMENT$POINTER:  /* MOVES MEMORY POINTER FORWARD ONE SPACE AND ACCOUNTS FOR */
                     /* END OF LINE OR PAGE.   RETURNS TRUE IF DISPLAY */
                     /* IS NOT FULL */

      PROCEDURE BYTE PUBLIC;

      COLUMN = COLUMN + 1;
      IF( LINE$CHECK AND PAGE$CHECK ) THEN RETURN (TRUE);
      IF( PAGE$CHECK = FALSE ) THEN RETURN (FALSE);
              ELSE RETURN (TRUE);

END;  /* INCREMENT$POINTER */
```

```
DECREMENT$POINTER:  /* DECREMENTS POINTER INTO ASCII MEMORY */
                       /* RETURNS TRUE WHEN POINTER WAS DECREMENTED */

      PROCEDURE BYTE PUBLIC;

      IF( COLUMN > 1) THEN COLUMN = COLUMN - 1;
              ELSE IF ( LINE > 1)
                      THEN DO;
                              LINE = LINE - 1;
                              COLUMN = 80;
                           END;
                      ELSE RETURN (FALSE);
      RETURN (TRUE);

END;  /* DECREMENT$POINTER */
```

```
BACKGROUND$STATUS:  /* TRUE INDICATES CHARACTER IS IN BACKGROUND NOT FOREGROUND */

      PROCEDURE BYTE PUBLIC;

      IF(( ALPHA$WS(ASCHII$PTR) AND 80H) = 80H ) THEN RETURN (TRUE);
              ELSE RETURN (FALSE);

END;  /* BACKGROUND$STATUS */
```

```
FIND$FOREGROUND:  /* INCREMENTS POINTER UNTIL FOREGROUND OR END OF PAGE */

      PROCEDURE PUBLIC;

      DO WHILE(BACKGROUND$STATUS);
              IF(INCREMENT$POINTER = FALSE) THEN RETURN;
      END;

END;  /* FIND FOREGROUND */
```

```
FIND$BACKGROUND: /* INCREMENTS POINTER UNTIL BACKGROUND OR END OF PAGE */

        PROCEDURE BYTE PUBLIC;

        DO WHILE(NOT(BACKGROUND$STATUS));
                IF(INCREMENT$POINTER = FALSE) THEN RETURN (FALSE);
        END;
        RETURN (TRUE);

END;    /* FIND$BACKGROUND */

/* BEGIN CURSOR ROUTINES */
```

```
CURSOR$HOME:  /* RETURNS POINTER TO FIRST FOREGROUND CHARACTER ON THE PAGE */

        PROCEDURE PUBLIC;

        LINE, COLUMN = 1;
        CALL FIND$FOREGROUND;

END;    /* CURSOR$HOME */
```

```
FORESPACE:  /* WRITE A NULL, THEN INCREMENT POINTER UNTIL FOREGROUND */
                /* DATA OR END OF PAGE IS FOUND */

        PROCEDURE PUBLIC;

        IF( INCREMENT$POINTER ) THEN CALL FIND$FOREGROUND;

END;    /* FORESPACE */
```

```
BACKSPACE:  /* MORE POINTER BACK ONE SPACE IF RESULT DOES NOT PLACE THE */
               /* POINTER UNDER A BACKGROUND CHARACTER OR PAST BEGINNING OF LINE */

      PROCEDURE PUBLIC;

      IF( DECREMENT$POINTER AND ( BACKGROUND$STATUS = FALSE ))
              THEN RETURN;
      IF( INCREMENT$POINTER ) THEN RETURN;

END;  /* BACKSPACE */
```

```
CURSOR$TAB:  /* MOVE POINTER TO FIRST FOREGROUND CHARACTER FOLLOWING */
              /* THE NEXT BACKGROUND FIELD, IF THERE IS ONE */

      PROCEDURE PUBLIC;

      IF( FIND$BACKGROUND ) THEN CALL FIND$FOREGROUND;

END;  /* CURSOR$TAB */
```

```
LINE$FEED:  /* MOVE POINTER TO NEXT LINE, SAME COLUMN.  LEAVE POINTER AT */
              /* FIRST FOREGROUND CHARACTER FOUND IN LINE BETWEEN COLUMN */
              /* AND END OF LINE, IF ONE EXISTS.  IF ONE DOES NOT EXIST, */
              /* INCREMENT LINE NUMBER AND REPEAT PROCESS UNTIL ONE IS */
              /* FOUND OR END OF PAGE OCCURS */

      PROCEDURE PUBLIC;

              DCL     TEMP    BYTE;

      LINE = LINE + 1;
      IF( PAGE$CHECK = FALSE)
              THEN DO;
                      LINE = LINE - 1;
                      RETURN;
              END;
      TEMP = COLUMN;
 LOOP:        IF( BACKGROUND$STATUS = FALSE ) THEN RETURN;
      COLUMN = COLUMN + 1;
      IF( LINE$CHECK = FALSE)
              THEN DO;
                      IF( PAGE$CHECK = TRUE)
                              THEN DO;
                                      COLUMN = TEMP;
                                      GO TO LOOP;
                                   END;
                              ELSE RETURN;
             END;
              GO TO LOOP;

END;  /* LINE$FEED */
```

```
VERTICAL$FEED:  /* SAME PROCESS AS LINE$FEED, EXCEPT WORKS BACKWARDS */

   PROCEDURE PUBLIC;

       DCL     TEMP     BYTE;

   IF( LINE > 1 )
       THEN DO;
           LINE = LINE - 1;
           TEMP = COLUMN;
       LOOP:   IF( BACKGROUND$STATUS = FALSE) THEN RETURN;
           COLUMN = COLUMN + 1;
           IF( COLUMN > 80)
               THEN DO;
                   IF(( LINE := LINE -1 ) > 0)
                       THEN DO;
                           COLUMN = TEMP;
                           GO TO LOOP;
                         END;
                       ELSE RETURN;
                 END;
               ELSE GO TO LOOP;
       END;
       ELSE RETURN;

END;  /* VERTICAL$FEED */
```

```
CARRAGE$RETURN:  /* PLACES POINTER AT BEGINNING OF FIRST FOREGROUND CHARACTER */
                 /* IN LINE, IF ONE EXISTS */

    PROCEDURE PUBLIC;

    COLUMN = 1;
    DO WHILE(BACKGROUND$STATUS AND (COLUMN < 80) );
         COLUMN = COLUMN + 1;
    END;

END;  /* CARRAGE$RETURN */
```

```
CURSOR$ROUTINE:  /* MAIN ROUTINE TO CONTROL POINTER INTO ASCII MEMORY */

      PROCEDURE (CHAR) BYTE PUBLIC;

      DCL     CURSOR$TABLE(*) BYTE    DATA(CH,'$',FS,'$',BS,'$',LF,'$',VT,'$',
              TAB,'$',CR,'$',0FFH,'$',0FFH,'$','$',);
      DCL     (CHAR, TOKEN)            BYTE;

      TOKEN = SEARCH(.CHAR,.CURSOR$TABLE);
      DO CASE TOKEN;
              CALL CURSOR$HOME;                 /* CASE 0 */
              CALL FORESPACE;                    /* CASE 1 */
              CALL BACKSPACE;                    /* CASE 2 */
              CALL LINE$FEED;                    /* CASE 3 */
              CALL VERTICAL$FEED;                /* CASE 4 */
              CALL CURSOR$TAB;                   /* CASE 5 */
              CALL CARRAGE$RETURN;               /* CASE 6 */
              RETURN (TOKEN);                           /* CASE 7 */
              RETURN (TOKEN);                    /* CASE 8 */
              RETURN (TOKEN);                    /* CASE 9 */
      END;  /* CASE */
      RETURN (TOKEN);

END;  /* CURSOR$ROUTINE */
```

```
STORE$ASCHII$MEMORY:  /* MAIN ROUTINE THAT CONTROLS WRITING CHARACTERS IN MEMORY */

      PROCEDURE (CHAR) PUBLIC;

              DCL     CHAR    BYTE;

 WRITE:  IF(EDIT$MODE) OR (NULL$CHECK)
              THEN DO;
              IF( ASCHII$PTR > 2559 ) THEN RETURN;
              IF( BACKGROUND$STATUS = FALSE ) THEN IF ( BG$FG$MODE )
                              THEN ALPHA$WS(ASCHII$PTR) = (CHAR OR 80H);
                              ELSE ALPHA$WS(ASCHII$PTR) = CHAR;
                      ELSE DO;
                              CALL FIND$FOREGROUND;
                              GO TO WRITE;
                      END;  /* DO */
              IF(INCREMENT$POINTER = FALSE) THEN RETURN;
              END;  /* THEN DO */
              ELSE IF(BACKGROUND$STATUS = FALSE)
                              THEN IF(INCREMENT$POINTER = FALSE) THEN RETURN;
                                                                 ELSE RETURN;
                              ELSE DO;
                                      DO WHILE(ALPHA$WS(ASCHII$PTR) <> NULL);
                                              IF(INCREMENT$POINTER = FALSE)
                                                      THEN RETURN;
                                      END;  /* DO WHILE */
                                      GO TO WRITE;
                              END;  /* ELSE */


END;  /* STORE$ASCHII$MEMORY */
```

```
MEMORY$DUMP:  /* DUMPS SIMULATED ALPHA MEMORY ON THE PLASMA */

        PROCEDURE PUBLIC;

                DCL     A$M     BYTE;

        CALL WRITE$P(CS);
        A$M = TRUE;
        LINE, COLUMN = 1;
        DO WHILE A$M;
                IF(BACKGROUND$STATUS) THEN CALL WRITE$P(BG);
                DO WHILE BACKGROUND$STATUS AND A$M;
                        CALL WRITE$P(ALPHA$WS(ASCII$PTR));
                        IF(INCREMENT$POINTER = FALSE) THEN A$M = FALSE;
                END;  /* DO WHILE BACKGROUND STATUS */

                IF(BACKGROUND$STATUS = FALSE) THEN CALL WRITE$P(FG);
                DO WHILE ( ( BACKGROUND$STATUS = FALSE ) AND A$M );
                        IF(ALPHA$WS(ASCII$PTR) = NULL)
                                THEN CALL WRITE$P(20H);
                                ELSE CALL WRITE$P(ALPHA$WS(ASCII$PTR));
                        IF(INCREMENT$POINTER = FALSE) THEN A$M = FALSE;
                END;  /* WHILE FOREGROUND */

        END;  /* DO WHILE A$M */

END MEMORY$DUMP;
```

```
VECTOR$DUMP:
            /* DUMPS VECTOR MEMORY TO PLASMA */
      PROC PUBLIC;

              DCL      ( I, J, LAST$VECTOR )    ADDRESS;

      I = 0;
      LAST$VECTOR = 6144;
      CALL WRITE$P( CV );

      DO WHILE ( ( LAST$VECTOR := LAST$VECTOR - 1 ) > 0 ) AND
                      ( VECTOR$WS( LAST$VECTOR ) <> '$' );
              /*      FIND LAST VECTOR ENTERED         */
      END;

      DO WHILE I < LAST$VECTOR;
              CALL WRITE$P(CG);
              DO J = 1 TO 3;
                      CALL WRITE$P(VECTOR$WS(I + J));
              END;   /* DO */
              I = I + 3;
      END;   /* DO */
      CALL WRITE$P(ETX);

END VECTOR$DUMP;
```

```
CHANGE$FOREGROUND$TO$BACKGROUND:
    /* ALLOWS CHANGING FOREGROUND TO BACKGROUND TO PREVENT EDITING OF TEXT */

    PROCEDURE PUBLIC;

    LINE = 1;
    COLUMN = 0;
    DO WHILE( INCREMENT$POINTER = TRUE );
            IF(BACKGROUND$STATUS = FALSE) AND (ALPHA$WS(ASCHII$PTR) <> NULL)
                    THEN ALPHA$WS(ASCHII$PTR) = ALPHA$WS(ASCHII$PTR) OR 80H;
    END;   /* DO WHILE */

END;   /* CHANGE$FOREGROUND$TO$BACKGROUND */
```

```
CHANGE$BACKGROUND$TO$FOREGROUND:
      /* ALLOWS CHANGING BACKGROUND TO FOREGROUND FOR EDITING */

      PROCEDURE PUBLIC;

      LINE = 1;
      COLUMN = 0;
      DO WHILE( INCREMENT$POINTER = TRUE );
              ALPHA$WS(ASCHII$PTR) = (ALPHA$WS(ASCHII$PTR) AND 7FH);
      END;   /* DO WHILE */

END;   /* CHANGE$BACKGROUND$TO$FOREGROUND */

END;   /* PSMEM */
```

```
/*EXTERNALS: */

 CLEAR$SCREEN:

      PROCEDURE EXTERNAL;
      END;   /* CLEAR$SCREEN */


 CLEAR$VECTORS:

      PROCEDURE EXTERNAL;
      END;   /* CLEAR$VECTORS */


 CLEAR$BACKGROUND:

      PROCEDURE EXTERNAL;
      END;   /* CLEAR BACKGROUND */


 CLEAR$FOREGROUND:

      PROCEDURE EXTERNAL;
      END;   /* CLEAR$FOREGROUND */


 CANCEL$FOREGROUND:

      PROCEDURE EXTERNAL;
      END;   /* CANCEL$FOREGROUND */


 INSERT$RECORD:
```

```
        PROCEDURE EXTERNAL;
        END;   /* INSERT$RECORD */


DELETE$RECORD:

        PROCEDURE EXTERNAL;
        END;   /* DELETE$RECORD */


DELETE$CHARACTER:

        PROCEDURE EXTERNAL;
        END;   /* DELETE$CHARACTER */


INSERT$CHARACTER:

        PROCEDURE EXTERNAL;
        END;   /* INSERT$CHARACTER */


MEMORY$CONTROL$ROUTINE:

        PROCEDURE(CHAR) BYTE EXTERNAL;
                DCL       CHAR     BYTE;
        END;   /* MEMORY$CONTROL$ROUTINE */


STORE$TEXT:

        PROCEDURE EXTERNAL;
```

```
      END;   /* STORE$TEXT */


END$TEXT:

      PROCEDURE EXTERNAL;
      END;   /* END$TEXT */


SUBSTITUTE$TEXT:

      PROCEDURE EXTERNAL;
      END;   /* SUBSTITUTE$TEXT */


BACKGROUND$MODE:

      PROCEDURE EXTERNAL;
      END;   /* BACKGROUND$MODE */


FOREGROUND$MODE:

      PROCEDURE EXTERNAL;
      END;   /* FOREGROUND$MODE */


GRAPHICS$MODE:

      PROCEDURE EXTERNAL;
      END;   /* GRAPHICS$MODE */
```

```
/*PLASMA$SCOPE$CONTROL                                  /*EXTERNALS

CONTROL$CODE$ROUTINE:

      PROCEDURE(CHAR) BYTE EXTERNAL;
                DCL       CHAR     BYTE;
      END;   /* CONTROL$CODE$ROUTINE */


UNUSED$ROUTINE:

      PROCEDURE(CHAR) BYTE EXTERNAL;
                DCL       CHAR     BYTE;
      END;   /* UNUSED$ROUTINE */


WS$MEMORY$CONTROLLER:

      PROCEDURE(CHAR) EXTERNAL;
                DCL       CHAR     BYTE;
      END;   /* WS$MEMORY$CONTROLLER */


STORE$IN$MEMORY:

      PROCEDURE(CHAR) EXTERNAL;
                DCL       CHAR     BYTE;
      END;   /* STORE$IN$MEMORY */
```

```
PSCONT:  /* CONTROL PROGRAMS FOR STORE$IN$MEMORY ROUTINES */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PSMEM.EXT )

            DCL     EDIT$MODE                 BYTE     EXTERNAL,
                    CURRENT$MODE             BYTE     EXTERNAL,
                    BG$FG$MODE                BYTE     EXTERNAL;

            DCL     EXPECTED$BYTES            BYTE     EXTERNAL,
                    LINE                      BYTE     EXTERNAL,
                    COLUMN                   BYTE     EXTERNAL;

            DCL     VECTOR$POINTER            BYTE     EXTERNAL,
                    ALPHA$WS(2562)            BYTE     EXTERNAL,
                    VECTOR$WS(6146)           BYTE     EXTERNAL;

            DCL     FIRST$PASS                BYTE     INITIAL (TRUE);


        /*  MEMORY$CONTROL ROUTINES */
```

```
CLEAR$SCREEN:    /* MOVE NULL CHARACTERS TO ALPHANUMERIC MEMORY AND LEAVE */
                /* SYSTEM READY FOR FOREGROUND.                          */

        PROCEDURE PUBLIC;

                DCL      INDEX   ADDRESS;

        DO INDEX = 0 TO 2559;
                ALPHA$WS( INDEX ) = 00H;
        END;
        COLUMN, LINE = 1;
        BG$FG$MODE = FALSE;

END;    /* CLEAR$SCREEN */
```

```
CLEAR$VECTORS:  /* MOVE NULL CHARACTERS TO VECTOR MEMORY */

      PROCEDURE PUBLIC;

                DCL      INDEX   ADDRESS;

      DO INDEX = 1 TO 6143;
                VECTOR$WS( INDEX ) = 00H;
      END;
      VECTOR$POINTER = 1;

END;  /* CLEAR$VECTORS */
```

```
CLEAR$BACKGROUND:  /* MOVE NULLS INTO BACKGROUND OF ASCII MEMORY */

        PROCEDURE PUBLIC;

        LINE = 1;
        COLUMN = 0;
        DO WHILE(INCREMENT$POINTER);
                IF(BACKGROUND$STATUS) THEN ALPHA$WS(ASCHII$PTR) = 00H;
        END;
        COLUMN, LINE = 1;
        CALL FIND$FOREGROUND;

END;   /* CLEAR$BACKGROUND */
```

```
CLEAR$FOREGROUND:  /* MOVE NULLS INTO FOREGROUND OF ASCII MEMORY */

      PROCEDURE PUBLIC;

      LINE = 1;
     COLUMN = 0;
      DO WHILE(INCREMENT$POINTER);
              IF(BACKGROUND$STATUS = FALSE) THEN ALPHA$WS(ASCHII$PTR) = 00H;
      END;
      COLUMN, LINE = 1;
      CALL FIND$FOREGROUND;

END; /* CLEAR$FOREGROUND */
```

```
CANCEL$FOREGROUND:  /* MOVE NULLS INTO LAST ENTERED FOREGROUND */

        PROCEDURE PUBLIC;

        DO WHILE(BACKGROUND$STATUS = FALSE);
                ALPHA$WS(ASCHII$PTR) = 00H;
                IF(DECREMENT$POINTER = FALSE) THEN RETURN;
        END;

END;    /* CANCEL$FOREGROUND */
```

```
INSERT$RECORD:  /* INSERT BLANK LINE AT CURSOR LOCATION */

        PROCEDURE PUBLIC;

                DCL      (I,TEMP)          BYTE;


        TEMP = LINE;
        DO I = (TEMP + 1) TO 32;
                LINE = 32 - I + TEMP + 1;
                DO COLUMN = 1 TO 80;
                        ALPHA$WS(ASCHII$PTR) = ALPHA$WS(ASCHII$PTR - 80);
                END;
        END;
        LINE = TEMP;
        DO COLUMN = 1 TO 80;
                ALPHA$WS(ASCHII$PTR) = 00H;
        END;
        COLUMN = 1;

END;   /* INSERT RECORD */
```

```
DELETE$RECORD:  /* DELETE CURRENT LINE */

        PROCEDURE PUBLIC;

                DCL     (I,TEMP)        BYTE;

        TEMP = LINE;
        DO I = TEMP TO 31;
                LINE = I;
                DO COLUMN = 1 TO 80;
                        ALPHA$WS(ASCHII$PTR) = ALPHA$WS(ASCHII$PTR + 80);
                END;
        END;
        LINE = 32;
        DO COLUMN = 1 TO 80;
                ALPHA$WS(ASCHII$PTR) = 00H;
        END;
        LINE=TEMP;
        COLUMN = 1;

END;    /* DELETE$RECORD */
```

```
DELETE$CHARACTER:  /* DELETE CHARACTER OF CURRENT POINTER POSITION */
                    /* AND MOVE ALL CHARACTERS TO THE RIGHT ONE SPACE LEFT */

    PROCEDURE PUBLIC;

         DCL     (I,TEMP)        BYTE;

    TEMP = COLUMN;
    DO COLUMN = TEMP TO 79;
         ALPHA$WS(ASCHII$PTR) = ALPHA$WS(ASCHII$PTR + 1);
    END;
    COLUMN = 80;
    ALPHA$WS(ASCHII$PTR) = 00H;
    COLUMN = TEMP;

END;  /* DELETE$CHARACTER */
```

```
INSERT$CHARACTER:  /* INSERT BLANK CHARACTER A CURRENT POSITION AND MOVE */
                     /* ALL CHARACTERS TO THE RIGHT ONE SPACE TO THE RIGHT */

    PROCEDURE PUBLIC;

            DCL      (I,TEMP)          BYTE;

    TEMP = COLUMN;
    DO I = (TEMP + 1) TO 80;
            COLUMN = 80 - I + TEMP + 1;
            ALPHA$WS(ASCHII$PTR) = ALPHA$WS(ASCHII$PTR - 1);
    END;
    COLUMN = TEMP;
    ALPHA$WS(ASCHII$PTR) = 00H;

END;  /* INSERT$CHARACTER */
```

```
MEMORY$CONTROL$ROUTINE:   /* MAIN MEMORY CONTROL ROUTINE */

      PROCEDURE(CHAR) BYTE PUBLIC;

              DCL      (CHAR, TOKEN)              BYTE;
              DCL      MEMORY$CONT$TABLE(*)       BYTE     DATA (CS,'$',CV,'$',CB,
                       '$',CF,'$',CAN,'$',IR,'$',DR,'$',ICH,'$',DCH,'$','$');

      TOKEN = SEARCH(.CHAR, .MEMORY$CONT$TABLE);
      DO CASE TOKEN;
              CALL CLEAR$SCREEN;                  /* CASE 0 */
              CALL CLEAR$VECTORS;                 /* CASE 1 */
              CALL CLEAR$BACKGROUND;              /* CASE 2 */
              CALL CLEAR$FOREGROUND;             /* CASE 3 */
              CALL CANCEL$FOREGROUND;             /* CASE 4 */
              CALL INSERT$RECORD;                 /* CASE 5 */
              CALL DELETE$RECORD;                 /* CASE 6 */
              CALL INSERT$CHARACTER;              /* CASE 7 */
              CALL DELETE$CHARACTER;              /* CASE 8 */
              RETURN (TOKEN);                     /* CASE 9 */
      END;  /* CASE */
      RETURN (TOKEN);

END;  /* MEMORY$CONTROL$ROUTINE */

/*  END OF MEMORY CONTROL ROUTINES */


/* BEGIN CONTROL CODE ROUTINES */
```

```
STORE$TEXT:  /* STORE CHARACTER IN POSITION INDICATED BY NEXT 2 BYTES */
              /* DISABLES EDIT MODE */

        PROCEDURE PUBLIC;

        EDIT$MODE = FALSE;
        EXPECTED$BYTES = 3;
        CURRENT$MODE = ALP$WS;

END;    /* STORE$TEXT */
```

```
END$TEXT:  /* ENABLES EDIT MODE */

      PROCEDURE PUBLIC;

      BG$FG$MODE = FALSE;
      EXPECTED$BYTES = 0;
      CURRENT$MODE = NEUTRAL;
      EDIT$MODE = TRUE;

END;  /* END$TEXT */
```

```
SUBSTITUTE$TEXT:  /* SUBSTITUDE FOREGROUND CHARACTER AT LOCATION OF NEXT 2 BYTES WITH */
                     /* CHARACTER IN THE THIRD BYTE */

      PROCEDURE PUBLIC;

      EDIT$MODE = TRUE;
      EXPECTED$BYTES = 3;
      CURRENT$MODE = ALP$WS;

END;  /* SUBSTITUTE$TEXT */
```

```
BACKGROUND$MODE:  /* SET BACKGROUND MODE IN ASCII MEMORY */

      PROCEDURE PUBLIC;

      BG$FG$MODE = TRUE;        /* BG SET */
      CURRENT$MODE = ALP$WS;

END;  /* BACKGROUND$MODE */
```

```
FOREGROUND$MODE:  /* SET FOREGROUND MODE IN ASCII MEMORY */

      PROCEDURE PUBLIC;

      BG$FG$MODE = FALSE;
      CURRENT$MODE = ALP$WS;

END;  /* FOREGROUND$MODE */
```

```
GRAPHICS$MODE:  /* SET GRAPHICS MODE OF OPERATION IN MDS VECTOR MEMORY */

      PROCEDURE PUBLIC;

      CURRENT$MODE = VEC$WS;
      EXPECTED$BYTES = 3;

END;  /* GRAPHICS$MODE */
```

```
CONTROL$CODE$ROUTINE:  /* SECOND MAIN MEMORY CONTROL ROUTINE */

      PROCEDURE(CHAR) BYTE PUBLIC;

              DCL     CONTROL$TABLE(*)          BYTE    DATA (STX,'$',ETX,'$',
                      SUB,'$',BG,'$',FG,'$',CG,'$','$');
              DCL     (CHAR,TOKEN)             BYTE;

      TOKEN = SEARCH(.CHAR, .CONTROL$TABLE);
      DO CASE TOKEN;
              CALL STORE$TEXT;                  /* CASE 0 */
              CALL END$TEXT;                   /* CASE 1 */
              CALL SUBSTITUTE$TEXT;             /* CASE 2 */
              CALL BACKGROUND$MODE;             /* CASE 3 */
              CALL FOREGROUND$MODE;             /* CASE 4 */
              CALL GRAPHICS$MODE;               /* CASE 5 */
              RETURN (TOKEN);                   /* CASE 6 */
      END;  /* CASE */
      RETURN (TOKEN);

END;  /* CONTROL$CODE$ROUTINE */

/* END CONTROL CODE ROUTINES */
/* BEGIN UNUSED ROUTINES */
```

```
UNUSED$ROUTINE:   /* PERFORM NO ACTION ON UNUSED OF CODES.   THESE CODES */
                     /* ARE NOT USED WITH ONE-WAY COMMUNICATION */

      PROCEDURE(CHAR) BYTE PUBLIC;

            DCL      (CHAR,TOKEN)    BYTE;
            DCL      UNUSED$TABLE(*) BYTE     DATA (FC1,'$',CA,'$',BL,'$',FC2,
                     '$',VR,'$',SYN,'$',FC3,'$',FC4,'$',FC5,'$','$');

      TOKEN = SEARCH(.CHAR, .UNUSED$TABLE);
      DO CASE TOKEN;
            RETURN (TOKEN);          /* CASE 0 */
            RETURN (TOKEN);          /* CASE 1 */
            RETURN (TOKEN);          /* CASE 2 */
            RETURN (TOKEN);          /* CASE 3 */
            RETURN (TOKEN);          /* CASE 4 */
            RETURN (TOKEN);          /* CASE 5 */
            RETURN (TOKEN);          /* CASE 6 */
            RETURN (TOKEN);          /* CASE 7 */
            RETURN (TOKEN);          /* CASE 8 */
            RETURN (TOKEN);          /* CASE 9 */
      END;  /* CASE */

END;  /* UNUSED$ROUTINE */


/* BEGIN WS$MEMORY$CONTROLLER */
```

```
WS$MEMORY$CONTROLLER:  /* CONTROLS IF AND WHAT IS WRITTEN INTO MDS MEMORY */

      PROCEDURE(CHAR) PUBLIC;

            DCL      CHAR     BYTE;

      IF( CURRENT$MODE = ALF$WS)
            THEN DO;
                  IF( EXPECTED$BYTES = 3) THEN COLUMN = (CHAR + 01H);
                  IF( EXPECTED$BYTES = 2) THEN LINE = (CHAR + 01H);
                  IF( EXPECTED$BYTES = 1)
                        THEN CALL STORE$ASCHII$MEMORY(CHAR);
                  EXPECTED$BYTES = EXPECTED$BYTES - 1;
            END;

      IF( CURRENT$MODE = VEC$WS)
            THEN DO;
                  VECTOR$WS(VECTOR$POINTER) = CHAR;
                  EXPECTED$BYTES = EXPECTED$BYTES - 1;
                  VECTOR$POINTER = VECTOR$POINTER + 1;
                  IF( EXPECTED$BYTES = 0 )
                        THEN VECTOR$WS(VECTOR$POINTER) = '$';
            END;

END;  /* WS$MEMORY$CONTROLLER */
```

```
STORE$IN$MEMORY:   /* MAIN PROGRAM TO SIMULATE PLASMA MEMORY */
                      /* CALLS SECONDARY CONTROL PROGRAMS */

      PROCEDURE(CHAR) PUBLIC;

              DCL       CHAR      BYTE;

      IF(FIRST$PASS)
              THEN DO;
                      CALL INITIALIZE$MEMORY;
                      FIRST$PASS = FALSE;
              END;   /* DO */
      IF( EXPECTED$BYTES > 0)
              THEN DO;
                      CALL WS$MEMORY$CONTROLLER(CHAR);
                      RETURN;
              END;   /* DO */
      IF( CURSOR$ROUTINE(CHAR) < 7 ) THEN RETURN;
      IF( MEMORY$CONTROL$ROUTINE(CHAR) < 9) THEN RETURN;
      IF( CONTROL$CODE$ROUTINE(CHAR) < 6 ) THEN RETURN;
      IF( UNUSED$ROUTINE(CHAR) < 9 ) THEN RETURN;
      CALL STORE$ASCHII$MEMORY(CHAR);

END STORE$IN$MEMORY;


END PSCONT;
```

```
/*EXTERNALS: */


 ALPHA:
      PROCEDURE        EXTERNAL;
      END ALPHA;

 STXS:
      PROCEDURE        EXTERNAL;
      END STXS;

 SUBS:
      PROCEDURE        EXTERNAL;
      END SUBS;
```

```
/*ALPHA: */

                /* CONTAINS ALPHA, STXS, SUBS SUBROUTINES */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:SYS.EXT )

 ALPHA:
                /* PASSES CHARACTERS FROM THE MDS KEYBOARD TO PLASMA */

      PROCEDURE PUBLIC;

      DCL     ( CHAR, STATUS )         BYTE;

CALL WRITE$LINE$CRT( .('INPUT:', CR, LF, '$$' ) );

DO FOREVER;
      CHAR = ECHO$CRT;
      CALL WRITE$PS( CHAR );
      IF CHAR = ESCAPE THEN RETURN;
END; /* FOREVER */
END ALPHA;
```

```
STXS:          /* SET ASCII INPUT MODE AND CLEAR EDIT MODE */

       PROCEDURE        PUBLIC;

       DCL      BPTR               BYTE;
      DCL      BUFF(80)           BYTE;
       DCL      (COLUMN, ROW)              ADDRESS;

COLUMN,ROW = 99;
DO WHILE ((COLUMN > 79 ) OR (ROW > 31 ));
       BPTR=0;
       CALL WRITE$LINE$CRT( .(CR,LF,'COLUMN, ROW: ',CR,LF,'$$') );
       CALL READ$LINE$CRT( .BUFF );
       BPTR = BPTR + DETRASH( .BUFF );
       COLUMN = CONVERT$HEXADECIMAL( .BUFF( BPTR ) );
       BPTR = BPTR + FIND$BLANK( .BUFF( BPTR ) );
       BPTR = BPTR + DETRASH( .BUFF( BPTR ) );
       ROW = CONVERT$HEXADECIMAL( .BUFF( BPTR ) );
       BPTR = BPTR + FIND$BLANK( .BUFF( BPTR ) );
END;

CALL WRITE$PS( STX );
CALL WRITE$PS( LOW( COLUMN ) );
CALL WRITE$PS( LOW( ROW ) );

END  STXS;
```

```
SUBS:           /* SUBSTITUTE TEXT AND SET EDIT MODE */

     PROCEDURE       PUBLIC;

      DCL      BPTR             BYTE;
      DCL      BUFF(80)         BYTE;
      DCL      (COLUMN, ROW)            ADDRESS;

COLUMN,ROW = 99;
DO WHILE (( COLUMN > 79 ) OR ( ROW > 31 ));
      BPTR=0;
      CALL WRITE$LINE$CRT( .(CR,LF,'COLUMN, ROW: ',CR,LF,'$$') );
      CALL READ$LINE$CRT( .BUFF );
      BPTR = BPTR + DETRASH( .BUFF );
      COLUMN = CONVERT$HEXADECIMAL( .BUFF( BPTR ) );
      BPTR = BPTR + FIND$BLANK( .BUFF( BPTR ) );
      BPTR = BPTR + DETRASH( .BUFF( BPTR ) );
      ROW = CONVERT$HEXADECIMAL( .BUFF( BPTR ) );
      BPTR = BPTR + FIND$BLANK( .BUFF( BPTR ) );
END;

CALL WRITE$PS( SUB );
CALL WRITE$PS( LOW( COLUMN ) );
CALL WRITE$PS( LOW( ROW ) );

END SUBS;

END CTEXT;
```

```
DEMO:  /* PROGRAM TO DEMONSTRATE AND EXERCISE PLASMA FUNCTIONS */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PSMEM.EXT )
$ INCLUDE( :F1:CG1.EXT )
$ INCLUDE( :F1:VDEMO.EXT )
$ INCLUDE( :F1:CTEXT.EXT )
$ INCLUDE( :F1:FILE.EXT )

      DCL     PERIOD                   LIT     '2EH';
      DCL     (TOKEN$NUMBER,BPTR)      BYTE;
      DCL     COMMAND$TABLE(*)         BYTE    DATA('NULL$','CH$','STX$',
             'ETX$','FS$','BS$','TAB$','LF$','VT$','CS$','CR$','CG$','CV$',
              'BG$','FG$','CB$','CF$','CAN$','SUB$','IR$','DR$','ICH$','DCH$',
              'ALPHA$','DUMPA$', 'DUMPV$', 'DUMP$','FTOB$','BTOF$','CURSOR$',
              'VECTOR$', 'CMDS$', 'FILE$', 'VDEMO$', 'EXIT$', '$$');
      DCL     BUFFER(128)              BYTE;
      DCL     CHAR                     BYTE;

/* ALPHA EXERCISES ALL PLASMA CODES */
/* DUMPA AND DUMPV DUMPS ALPHA OR VECTOR MEMORY ONLY */
/* DUMP DUMPS ALL SIMULATED MEMORY PLASMA */
/* FTOB CHANGES FOREGROUND TO BACKGROUND */
/* BTOF CHANGES BACKGROUND TO FOREGROUND */
```

```
/*DEMONSTRATION                                                    DEMO

/* CURSOR DRIVES A VECTOR CURSOR ON PLASMA */
/* VECTOR ALLOWS EASY WRITING AND ERASING OF VECTORS ON PLASMA */
/* VDEMO IS A VECTOR DEMONSTRATION */
/* FILE PROVIDES FACILITIES FOR SAVING OR RECOVERY MEMORY ON DISK */
/* EXIT CAUSES TERMINATION OF PROGRAM AND ENTRY INTO ISIS */
```

```
/*MESSAGES: */

 ERROR$MESSAGE$1:

        PROCEDURE;
        CALL WRITE$LINE$CRT( .(CR,LF,'INVALID COMMAND SYNTAX',CR,LF,'$$') );
        END ERROR$MESSAGE$1;

 ERROR$MESSAGE$2:

        PROCEDURE;
        CALL WRITE$LINE$CRT( .(CR,LF,'INVALID COMMAND',CR,LF,'$$') );
        END ERROR$MESSAGE$2;

 COMMAND$MESSAGE:

        PROCEDURE;

        CALL WRITE$LINE$CRT( .(CR,LF,'           COMMAND LIST',CR,LF,
        'NULL, CH, STX, ETX, FS, BS, TAB, LF, VT, CS, CR, CG, CV, BG', CR, LF,
         'FG, CB, CF, CAN, SUB, IR, DR, ICH, DCH, ALPHA, DUMP, DUMPA, DUMPV', CR, LF,
         'FTOB, BTOF, CURSOR, VECTOR, VDEMO, FILE, CMDS, EXIT', CR, LF, '$$') );

        END COMMAND$MESSAGE;
```

```
/*MAIN$DEMONSTRATION$PROGRAM: */

BUFFER(126), BUFFER(127) = '$';
CALL INITIALIZE$PS;
CALL COMMAND$MESSAGE;
DO FOREVER;

 BEGIN:       CALL WRITE$LINE$CRT( .(CR,LF, '% $$') );
        CALL READ$LINE$CRT(.BUFFER);
        BPTR = DETRASH(.BUFFER);

        TOKEN$NUMBER = SEARCH(.BUFFER(BPTR), .COMMAND$TABLE);

        DO CASE TOKEN$NUMBER;
                CALL WRITE$PS(NULL);                      /* CASE 0 */
                CALL WRITE$PS(CH);                       /* CASE 1 */
                DO;
                CALL STXS;                                /* CASE 2 */
                CALL WRITE$PS( ETX );
                CALL ALPHA;
                END;
                CALL WRITE$PS(ETX);                       /* CASE 3 */
                CALL WRITE$PS(FS);                        /* CASE 4 */
                CALL WRITE$PS(BS);                        /* CASE 5 */
                CALL WRITE$PS(TAB);                       /* CASE 6 */
                CALL WRITE$PS(LF);                       /* CASE 7 */
                CALL WRITE$PS(VT);                        /* CASE 8 */
                CALL WRITE$PS(CS);                        /* CASE 9 */
                CALL WRITE$PS(CR);                       /* CASE 10 */
                CALL WRITE$PS(CG);                        /* CASE 11 */
                CALL WRITE$PS(CV);                        /* CASE 12 */
                CALL WRITE$PS(BG);                        /* CASE 13 */
                CALL WRITE$PS(FG);                        /* CASE 14 */
```

```
                CALL WRITE$PS(CB);                              /* CASE 15 */
                CALL WRITE$PS(CF);                              /* CASE 16 */
                CALL WRITE$PS(CAN);                             /* CASE 17 */
                DO;
                CALL SUBS;                                      /* CASE 18 */
                CALL WRITE$PS( ETX );
                CALL ALPHA;
                END;
                CALL WRITE$PS(IR);                              /* CASE 19 */
                CALL WRITE$PS(DR);                              /* CASE 20 */
                CALL WRITE$PS(ICH);                             /* CASE 21 */
                CALL WRITE$PS(DCH);                            /* CASE 22 */
                CALL ALPHA;                                     /* CASE 23 */
                CALL MEMORY$DUMP;                               /* CASE 24 */
                CALL VECTOR$DUMP;                               /* CASE 25 */
                DO;
                CALL MEMORY$DUMP;
                CALL VECTOR$DUMP;                               /* CASE 26 */
                END;
                CALL CHANGE$FOREGROUND$TO$BACKGROUND;           /* CASE 26 */
                CALL CHANGE$BACKGROUND$TO$FOREGROUND;           /* CASE 27 */
                CALL MOVE$CURSOR;                               /* CASE 28 */
                CALL CGS;                                       /* CASE 29 */
               CALL COMMAND$MESSAGE;                           /* CASE 30 */
                CALL FILE$HANDLER( FALSE );                     /* CASE 31 */
                CALL VDEMO;                                     /* CASE 32 */
                CALL EXIT;                                     /* CASE 33 */
                CALL ERROR$MESSAGE$2;                           /* CASE 34 */
        END;  /* CASE */

END;  /* DO FOREVER */

END DEMO;
```

```
/*EXTERNALS: */

 DISPLAY$MENU:
       PROC EXTERNAL;
       END DISPLAY$MENU;

 CALL$ROUTINES:
       PROC( TOKEN$ADDRESS ) BYTE EXTERNAL;
                DCL      TOKEN$ADDRESS   ADDRESS;
       END CALL$ROUTINES;

 DISPLAY$ALL:
       PROC EXTERNAL;
       END DISPLAY$ALL;

 VDEMO:
       PROC EXTERNAL;
       END VDEMO;
```

```
/*WAIT: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CG.DCL )

$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:CG1.EXT )
$ INCLUDE( :F1:CG2.EXT )

      DCL      ( X$ORIGIN, Y$ORIGIN )  ADDRESS PUBLIC;

 WAIT:
               /*  POSTS "CONTINUE?" ON CRT AND WAITS FOR A REPLY FROM KEYBOARD.  */
               /*  RETURNS TRUE IF REPLY IS "N" OR "N" */

      PROC BYTE PUBLIC;
               DCL     CHAR     BYTE;

      CALL WRITE$LINE$CRT( .('CONTINUE? (Y/N) $$' ) );
      IF ( ( CHAR := READ$CRT ) = 'N' ) OR ( CHAR = 'N' ) THEN
               RETURN TRUE;
      RETURN FALSE;

      END WAIT;
```

```
SET$ORIGIN:
                /* TRANSLATE TO THE POINT SPECIFIED.
                ALL POINTS ARE TAKEN FROM THE HARDWARE ORIGIN IN THE
               UPPER LEFT HAND CORNER.                                    */

      PROC PUBLIC;

                DCL       BUFFER( 128 )    BYTE;
                /* USES GLOBAL X$ORIGIN AND Y$ORIGIN */
                DCL       ( BPTR, SET )    BYTE;

      DO BPTR = 0 TO 125;
                BUFFER( BPTR ) = ' ';
      END;      /* CLEAR BUFFER */

      BUFFER( 126 ), BUFFER( 127 ) = '$';
      SET = TRUE;

      CALL WRITE$LINE$CRT( .( CR, LF, 'X ORIGIN, Y ORIGIN =', CR, LF, '$$' ));
      CALL READ$LINE$CRT( .BUFFER );
      BPTR = DETRASH( .BUFFER );
      X$ORIGIN = CONVERT$HEXADECIMAL( .BUFFER( BPTR ) );
      BPTR = BPTR + FIND$BLANK( .BUFFER( BPTR ) );
      BPTR = BPTR + DETRASH( .BUFFER( BPTR ) );
      Y$ORIGIN = CONVERT$HEXADECIMAL( .BUFFER( BPTR ) );
      IF NOT TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
                CALL WRITE$LINE$CRT( .(CR, LF, 'CANNOT TRANSLATE ORIGIN', CR, LF, '$$' ) );
                SET = FALSE;

      END SET$ORIGIN;
```

```
SET$VECTOR:
                /* SET A VECTOR POINT  */

      PROC( X, Y, VECTOR$ADDRESS ) PUBLIC;
                DCL     ( X, Y )          ADDRESS;
                DCL      VECTOR$ADDRESS  ADDRESS;
                DCL      ( VECTOR BASED VECTOR$ADDRESS ) ( 4 )    BYTE;

      VECTOR( 0 ) = CG;
      VECTOR(1), VECTOR(2), VECTOR(3) = 0;
      VECTOR( 1 ) = LOW( X ) AND 7FH;
      VECTOR( 2 ) = LOW( Y ) AND 7FH;
      VECTOR( 3 ) = HIGH( SHL( Y AND 0180H, 3 ) )
                OR HIGH( SHL( X AND 0180H, 1 ) );

      END SET$VECTOR;
```

```
START:
                /* WRITE A START POINT FOR A VECTOR */
      PROC( X, Y ) PUBLIC;
                DCL       ( X, Y )          ADDRESS;
                DCL       VECTOR( 4 )       BYTE;

      CALL SET$VECTOR( X, Y, .VECTOR );
      CALL WRITE$VECTOR( .VECTOR );

      END START;
```

```
DRAW:
            /* DRAWS A VECTOR FROM ANY ORIGIN EXTABLISHED BY TRANSLATE TO THE
            POINTS GIVEN */

    PROC( X, Y ) PUBLIC;
            DCL     ( X, Y )          ADDRESS;
            DCL     VECTOR( 6 )       BYTE;
            DCL     ( X0, Y0 )        ADDRESS;

    X0, Y0 = 0;
    VECTOR( 0 ) = CG;
    VECTOR( 1 ), VECTOR( 2 ), VECTOR( 3 ) = 0;
    VECTOR( 4 ), VECTOR( 5 ) = '$';

    IF TRANSLATE( .X0, .Y0, FALSE ) THEN
            DO;
            CALL SET$X( X0, .VECTOR( 1 ) );
            CALL SET$Y( Y0, .VECTOR( 1 ) );
            CALL WRITE$VECTOR( .VECTOR );
            END;    /* IF TRANSLATE */
    ELSE
            DO;
            CALL DISPLAY$VECTOR$ATTRIBUTES( .VECTOR( 1 ) );
            CALL WRITE$LINE$CRT( .('TRANSLATE CANNOT SET ORIGIN', CR, LF, '$$' ) );
            END;    /* ELSE DO */

    VECTOR( 3 ) = SET$END;

    IF TRANSLATE( .X, .Y, FALSE ) THEN
            DO;
            CALL SET$X( X, .VECTOR( 1 ) );
            CALL SET$Y( Y, .VECTOR( 1 ) );
            CALL WRITE$VECTOR( .VECTOR );
```

```
            END;     /* IF TRANSLATE */
      ELSE
            DO;
            CALL DISPLAY$VECTOR$ATTRIBUTES( .VECTOR( 1 ) );
            CALL WRITE$LINE$CRT( .('TRANSLATE CANNOT SET END ', CR, LF, '$$' ) );
            END;     /* ELSE DO */

      END DRAW;
```

```
DRAW$COORDINATES:
                /* DRAWS THE NUMBER OF LINES REQUESTED AT THE ORIGINS GIVEN */

        PROC( ORG$X, ORG$Y, LINES ) PUBLIC;
                DCL     ( ORG$X, ORG$Y, COUNT ) ADDRESS;
                DCL     LINES   BYTE;

        LOOP:  DO COUNT = 1 TO LINES;
                CALL COL( ORG$X + COUNT - 1 );
                CALL ROW( ORG$Y + COUNT - 1 );
        END LOOP;

END DRAW$COORDINATES;
```

```
FXG:
                /* FUNCTION X GRAPH */

        PROC PUBLIC;
                DCL      ( X0, Y0, X, Y, X1, Y1 )         ADDRESS;
                DCL      SET      BYTE;
                DCL      VECTOR( 6 )      BYTE;

        VECTOR( 0 ) = CG;
        VECTOR( 1 ), VECTOR( 2 ), VECTOR( 3 ) = 0;
        VECTOR( 4 ), VECTOR( 5 ) = '$';
        X1, Y1 = 0;
        X, Y = 1;
        X0, Y0 = 255;
        SET = TRUE;

        CALL WRITE$LINE$PS( .( STX, 1, 8,
                'Y = (X - X**2 / 16 + X**3 / 64) / 99', STX, 10, 12,
                'SCALE = 60 PER INCH $$' ) );

        IF TRANSLATE( .X0, .Y0, SET ) THEN
                CALL DRAW$COORDINATES( X0, Y0, 2 );

        CALL SET$X( X0, .VECTOR( 1 ) );
        CALL SET$Y( Y0, .VECTOR( 1 ) );
        CALL WRITE$VECTOR( .VECTOR );
        VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;

        POSITIVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 );
                X, X1 = X1 + 4;
                Y, Y1 = Y1 + 1;
                Y = ( Y - ( Y * Y ) + Y * Y * Y ) / 99;
                IF TRANSLATE( .X, .Y, FALSE ) THEN
```

```
                        DO;
                        CALL SET$X( X, .VECTOR( 1 ) );
                        CALL SET$Y( Y, .VECTOR( 1 ) );
                        CALL WRITE$VECTOR( .VECTOR );
                        END;     /* TRANSLATE */
                END POSITIVE;

        VECTOR( 3 ) = VECTOR( 3 ) XOR VECTOR( 3 );
        CALL SET$X( X0, .VECTOR( 1 ) );
        CALL SET$Y( Y0, .VECTOR( 1 ) );
        CALL WRITE$VECTOR( .VECTOR );
        X1, Y1 = 0;
        X, Y = 1;

        NEGATIVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 );
                X, X1 = X1 - 4;
                Y, Y1 = Y1 - 1;
                Y = - ( ( - ( Y - ( Y * Y )  + Y * Y * Y ) ) / 99 );
                IF TRANSLATE( .X, .Y, FALSE ) THEN
                        DO;
                        CALL SET$X( X, .VECTOR( 1 ) );
                        CALL SET$Y( Y, .VECTOR( 1 ) );
                        CALL WRITE$VECTOR( .VECTOR );
                        END;     /* TRANSLATE */
                END NEGATIVE;
END FXG;
```

```
FSR:

      PROC PUBLIC;

              DCL      ROW$NO  ADDRESS;

      CALL WRITE$LINE$PS( .( STX, 25, 31, 'FILL SCREEN WITH ROWS $$') );

      DO ROW$NO = 0 TO 511;    /* FILL SCREEN WITH ROWS */
              CALL ROW( ROW$NO );
     END;

      END FSR;
```

```
MRCV:

    PROC PUBLIC;

        DCL     ROW$NO  ADDRESS;

    CALL WRITE$LINE$PS( .(STX, 50, 31, 'MOVE ROW VECTOR DOWN SCREEN $$' ) );

    DO ROW$NO = 0 TO 511;    /* MOVE ROW VECTOR DOWN THE SCREEN */
        CALL WRITE$P( CV );
        CALL ROW( ROW$NO );
    END;

    END MRCV;
```

```
FSC:

        PROC PUBLIC;

                DCL      COL$NO  ADDRESS;

        CALL WRITE$LINE$PS( .(STX, 50, 10, 'FILL SCREEN WITH COLUMNS $$' ) );

        DO COL$NO = 0 TO 511;    /* FILL SCREEN WITH COLUMN VECTORS */
                CALL COL( COL$NO );
        END;

        END FSC;
```

```
ESC:

    PROC PUBLIC;

         DCL     COL$NO  ADDRESS;

    CALL WRITE$LINE$PS( .( STX, 50, 16, 'ERASE SCREEN BY COLUMN $$' ) );

    DO COL$NO = 0 TO 511;   /* ERASE SCREEN A COLUMN AT A TIME */
         CALL COL( COL$NO + 512 );
    END;

    END ESC;
```

```
MDC:

        PROC PUBLIC;

                DCL      COL$NO  ADDRESS;

        CALL COL( 511 );
        CALL WRITE$LINE$PS( .(STX, 0, 0, 'MOVING DOUBLE COLUMN $$' ) );

        DO COL$NO = 0 TO 511;   /* MOVING DOUBLE COLUMN */
                CALL COL( 511 - COL$NO );
                CALL COL( 1024 - COL$NO );
        END;

        END MDC;
```

```
FSRC:

      PROC PUBLIC;

            DCL      ( ROW$NO, COL$NO, CHANGE )      ADDRESS;

      CALL WRITE$LINE$PS( .( STX, 0, 31, 'FILL SCREEN BY ROW AND COLUMN $$' ) );

      DO CHANGE = 0 TO 511;    /* FILL SCREEN TOWARD BOTTOM LEFT CORNER */
            ROW$NO = CHANGE;
            CALL ROW( ROW$NO );
            COL$NO = 511 - CHANGE;
            CALL COL( COL$NO );
      END;

      END FSRC;
```

```
MRC:

        PROC PUBLIC;

                DCL      (ROW$NO, COL$NO, CHANGE )       ADDRESS;

        CALL WRITE$LINE$PS( .( STX, 0, 0, 'MOVING ROW AND COLUMN $$' ) );

        DO CHANGE = 0 TO 511;    /* MOVING CROSS */
                ROW$NO = CHANGE XOR 01FFH;
                CALL ROW( ROW$NO );
                CALL COL( ROW$NO );
                COL$NO = CHANGE XOR 03FFH;
                CALL ROW( COL$NO );
                CALL COL( COL$NO );
        END;

        END MRC;
```

```
FANS:

      PROC PUBLIC;

            DCL      ( CHANGE, X, Y )          ADDRESS;
            DCL      SET      BYTE;

     Y$ORIGIN = 511;
      X$ORIGIN = 0;
      SET = TRUE;
      CALL WRITE$P( CV );
      CALL WRITE$P( CS );
      CALL WRITE$LINE$PS( .( STX, 37, 4, 'F A N S $$' ) );

      FAN:  DO CHANGE = 5 TO 45 BY 5; /* DRAW RADIAL LINES FROM BOTTOM CORNERS */
            X = ( 99 - ( CHANGE * CHANGE / 70 ) ) * 5;
            Y = ( 8 * CHANGE / 5 ) * 5;

      IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
            DO;
            CALL DRAW( X, Y );
           IF CHANGE <> 45 THEN CALL DRAW( Y, X );
            END;

      IF TRANSLATE( .Y$ORIGIN, .Y$ORIGIN, SET ) THEN
            DO;
            CALL DRAW( -X, Y );
            IF CHANGE <> 45 THEN CALL DRAW( -Y, X );
            END;

      END FAN;

      END FANS;
```

```
HW:

     PROC PUBLIC;

          DCL      ( X, Y, RADIUS, CHANGE )       ADDRESS;
          DCL      SET      BYTE;

     CALL WRITE$LINE$PS( .( STX, 52, 8, 'H E A T W A V E $$' ) );
     X, Y = 1;
     RADIUS = 1;
     SET = TRUE;

     HEAT$WAVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 ) AND ( X > 0 ) AND ( X < 511 )
                             AND ( RADIUS <> 0 );

         RADIUS = ( RADIUS + 20 ) AND 00FFH;
          DO CHANGE = 0 TO 45;
               X = ( 99 - ( CHANGE * CHANGE /70 ) ) * RADIUS / 100;
               Y = ( CHANGE * 8 / 5 ) * RADIUS / 100;
               IF TRANSLATE( .X, .Y, SET ) THEN
                       DO;
                       CALL START( X, Y );
                       CALL START( Y, X );
                       CALL START( -X, -Y );
                       CALL START( -Y, -X );
                       END;
          END;     /* DO CHANGE */
     END HEAT$WAVE;

     END HW;
```

```
GE:

     PROC PUBLIC;

               DCL      ( X, Y, X1, Y1, RADIUS, CHANGE )         ADDRESS;
               DCL      SET      BYTE;
      CALL WRITE$LINE$PS( .( STX, 30, 2, 'G O O S E   E G G S $$' ) );

      RADIUS = 0;
      SET = FALSE;
      X, Y = 1;

      GOOSE$EGG:  DO WHILE ( Y > 0 ) AND ( Y < 511 ) AND ( X > 0 ) AND ( X < 511 )
                               OR ( RADIUS < 256 );

               RADIUS = RADIUS + 20;
               DO CHANGE = 0 TO 45;
                        X, X1 = ( 99 - ( CHANGE * CHANGE / 70 ) ) * RADIUS / 100;
                        Y, Y1 = ( CHANGE * 8 / 5 ) * RADIUS / 100;
                        IF TRANSLATE( .X, .Y, SET ) THEN
                                 CALL START( X, Y );
                        IF CHANGE <> 45 THEN CALL START( Y, X );
                        X = -X1;
                        Y = Y1;
                        IF TRANSLATE( .X, .Y, SET ) THEN
                                 CALL START( X, Y );
                        IF CHANGE <> 45 THEN CALL START( Y, X );
                        X = -X1;
                        Y = -Y1;
                        IF TRANSLATE( .X, .Y, SET ) THEN
                                 CALL START( X, Y );
                        IF CHANGE <> 45 THEN CALL START( Y, X );
                        X = X1;
```

```
                Y = -Y1;
                IF TRANSLATE( .X, .Y, SET ) THEN
                     CALL START( X, Y );
                IF CHANGE <> 45 THEN CALL START( Y, X );

          END;  /*DO CHANGE */

     END GOOSE$EGG;

     END GE;
```

```
TB:

      PROC PUBLIC;

            DCL      ( X, Y, CHANGE )          ADDRESS;
            DCL      ( SET, COUNT )  BYTE;

      CALL WRITE$LINE$PS( .( STX, 29, 31, 'T H U N D E R B I R D $$' ) );
      SET = TRUE;

      IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
            THUNDERBIRD:  DO;
            SET = FALSE;
            RADIUS:  DO CHANGE = 20 TO 220 BY 20;
                  Y = CHANGE + 20;
                  X = CHANGE;

                  DO WHILE ( X := X - 20 ) <> -( CHANGE + 20 );
                        CALL DRAW( X, Y );
                  END;

                  DO WHILE ( Y := Y - 20 ) <> -( CHANGE + 20 );
                        CALL DRAW( X, Y );
                  END;

                  DO WHILE ( X := X + 20 ) <> ( CHANGE + 20 );
                        CALL DRAW( X, Y );
                  END;

                  DO WHILE ( Y := Y + 20 ) <> ( CHANGE + 20 );
                        CALL DRAW(X, Y );
                  END;
```

```
                    COUNT = LOW( CHANGE );
                    CALL WRITE$HEXADECIMAL( COUNT );

          END RADIUS;

          END THUNDERBIRD;

     END TE;
```

```
RS:
      PROC PUBLIC;

            DCL      ( X, Y, CHANGE, TIME )  ADDRESS;
            DCL      CHAR     BYTE;
            DCL      BUFFER( 128)     BYTE;
            DCL      VECTOR( 4 )      BYTE;

      CALL WRITE$LINE$PS( .( STX, 5, 2, 'RADAR SCAN $$' ) );
      X, Y = 1;
      CHANGE = 0;
      TIME = 0;

      DO WHILE CHANGE = 0;
            CALL WRITE$LINE$CRT( .('SWEEP RATE? $$') );
            CALL READ$LINE$CRT( .BUFFER );
            CHANGE = CONVERT$HEXADECIMAL( .BUFFER );
      END; /* CHANGE = 0 */

      TIMES:  DO WHILE ( TIME := TIME + 1 ) <> 21;

            MOVE$Y:  DO WHILE ( Y > 0 ) AND  ( Y < 511 );
                     CALL START( X$ORIGIN, Y$ORIGIN );
                     CALL SET$VECTOR( X, Y, .VECTOR );
                     VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;
                     CALL WRITE$VECTOR( .VECTOR );
                     CALL SET$VECTOR( X$ORIGIN, Y$ORIGIN, .VECTOR );
                     VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
                     CALL WRITE$VECTOR( .VECTOR );
                     CALL SET$VECTOR( X, Y - CHANGE, .VECTOR );
                     VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE OR SET$END;
                     CALL WRITE$VECTOR( .VECTOR );
                     IF ( ( Y := Y + CHANGE ) > 510 ) OR ( Y = 0 ) THEN
```

```
                VERTICAL:  DO;
                CALL SET$VECTOR( X$ORIGIN, Y$ORIGIN, .VECTOR );
                VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
                CALL WRITE$VECTOR( .VECTOR );
                CALL SET$VECTOR( X, Y - CHANGE, .VECTOR );
                VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE OR SET$END;
                CALL WRITE$VECTOR( .VECTOR );
                IF ( Y > 1000 ) OR ( Y = 0 ) THEN
                        DO;
                        Y = 0;
                        X = 510;
                        END;
                ELSE
                        DO;
                        Y = 511;
                        X = 1;
                        END;
        END VERTICAL;
END MOVE$Y;

MOVE$X:  DO WHILE ( X > 0 ) AND ( X < 511 );
        CALL START( X$ORIGIN, Y$ORIGIN );
        CALL SET$VECTOR( X, Y, .VECTOR );
        VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;
        CALL WRITE$VECTOR( .VECTOR );
        CALL SET$VECTOR( X$ORIGIN, Y$ORIGIN, .VECTOR );
        VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
        CALL WRITE$VECTOR( .VECTOR );
        CALL SET$VECTOR( X - CHANGE, Y, .VECTOR );
        VECTOR( 3 ) = VECTOR( 3 ) OR SET$END OR SET$ERASE;
        CALL WRITE$VECTOR( .VECTOR );
        IF ( X := X + CHANGE ) > 510 OR ( X = 0 ) THEN
                HORIZONTAL:  DO;
```

```
                        CHANGE = - CHANGE;
                        CALL SET$VECTOR( X$ORIGIN, Y$ORIGIN, .VECTOR );
                        VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE;
                        CALL WRITE$VECTOR( .VECTOR );
                        CALL SET$VECTOR( X + CHANGE, Y, .VECTOR );
                        VECTOR( 3 ) = VECTOR( 3 ) OR SET$ERASE OR SET$END;
                        CALL WRITE$VECTOR( .VECTOR );
                        IF ( X > 1000 ) OR ( X = 0 ) THEN
                                DO;
                                X = 0;
                                Y = 1;
                                END;
                        ELSE
                                DO;
                                X = 511;
                               Y = 510;
                                END;
                        END HORIZONTAL;
        END MOVE$X;
END TIMES;

END RS;
```

```
DISPLAY$MENU:
                    /* LISTS AVAILABLE COMMANDS ON THE CRT */

          PROC PUBLIC;

                    DCL       MENU( * )          BYTE
                              DATA( CR, LF, 'F ILL  S CREEN WITH  R OWS', CR, LF,
                              'M OVE R OW USING  C LEAR  V ECTOR', CR, LF,
                              'F ILL  S CREEN WITH  C OLUMNS', CR, LF,
                              'E RASE  S CREEN BY  C OLUMNS', CR, LF,
                             'M OVING  D OUBLE  C OLUMN', CR, LF,
                              'F ILL  S CREEN WITH  R OWS AND  C OLUMNS', CR, LF,
                              'M OVING  R OW AND  C OLUMN', CR, LF,
                              'T RANS L ATE', CR, LF,
                              'F UNCTION  X  G RAPH', CR, LF,
                              'FANS', CR, LF,
                              'H EAT W AVE', CR, LF,
                              'G OOSE  E GGS', CR, LF,
                              'T HUNDER B IRD', CR, LF,
                              'R ADAR  S CAN', CR, LF,
                              'MENU CALLS THIS TABLE', CR, LF,
                              'D ISPLAY  A LL GRAPHS', CR, LF,
                              'CONTROL-L CLEARS ALPHANUMERICS', CR, LF,
                              'CONTROL-O CLEARS VECTORS', CR, LF,
                              'EXIT', CR, LF,
                              '$$' );

          CALL WRITE$LINE$CRT( .( CR, LF, 'USE ISOLATED UPPER CASE LETTERS FOR CALLING
          ROUTINES.', CR, LF, '$$' ) );
          CALL WRITE$LINE$CRT( .('FOR EXAMPLE, MRCV WILL CALL A ROUTINE TO
          DEMONSTRATE A MOVING ROW USING CLEAR  VECTOR TO ELIMINATE THE OLD
          ROW.', CR, LF, '$$' ) );
          CALL WRITE$LINE$CRT( .MENU );
```

```
    END DISPLAY$MENU;
```

```
DISPLAY$ALL:

        PROC PUBLIC;

                DCL      ( SET, COUNT, STOP )     BYTE;

CALL INITIALIZE$PS;
        CALL WRITE$LINE$CRT( .('DO YOU WANT TO STOP AFTER EACH GRAPH? $$' ) );
        IF ( ( CHAR := ECHO$CRT ) = 'N' ) OR ( CHAR = 'N' )
                THEN STOP = FALSE;
                ELSE STOP = TRUE;
        CALL WRITE$LINE$PS( .( STX, 0, 0, 00H ) );
        CALL WRITE$P( CS );
        CALL WRITE$P( CV );
        CALL FSR;
        IF STOP THEN IF WAIT THEN RETURN;
        CALL WRITE$P( CS );
        CALL MRCV;

        IF STOP THEN IF WAIT THEN RETURN;
        CALL WRITE$P( CV );
        CALL WRITE$P( CS );
        CALL FSC;
        CALL ESC;
        IF STOP THEN IF WAIT THEN RETURN;
        CALL WRITE$P( CS );
        CALL WRITE$P( CV );
        CALL MDC;

        IF STOP THEN IF WAIT THEN RETURN;
        CALL WRITE$P( CS );
        CALL WRITE$P( CV );
        CALL FSRC;
```

```
IF STOP THEN IF WAIT THEN RETURN;
CALL WRITE$P( CS );
CALL WRITE$P( CV );
CALL MRC;

IF STOP THEN IF WAIT THEN RETURN;
Y$ORIGIN = 511;
X$ORIGIN = 0;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
CALL FANS;
IF STOP THEN IF WAIT THEN RETURN;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
X$ORIGIN, Y$ORIGIN = 255;
SET = TRUE;
IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
        CALL DRAW$COORDINATES( X$ORIGIN, Y$ORIGIN, 1 );
CALL HW;
IF STOP THEN IF WAIT THEN RETURN;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
X$ORIGIN,Y$ORIGIN = 255;
IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
        CALL DRAW$COORDINATES( X$ORIGIN, Y$ORIGIN, 2 );
CALL GE;

IF STOP THEN IF WAIT THEN RETURN;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
CALL FXG;

IF STOP THEN IF WAIT THEN RETURN;
```

```
        CALL WRITE$P( CV );
        CALL WRITE$P( CS );
        X$ORIGIN, Y$ORIGIN = 255;
        CALL TB;
        IF STOP THEN IF WAIT THEN RETURN;
        CALL WRITE$P( CV );
        CALL WRITE$P( CS );
        CALL RS;
        CALL WRITE$LINE$PS( .(STX, 0, 0, 'DO IT AGAIN? $$' ) );
        CALL DISPLAY$MENU;

END DISPLAY$ALL;
```

```
CALL$ROUTINES:
                 /* CALLS DEMONSTRATION ROUTINES BASED ON TOKEN */

      PROC( TOKEN$ADDRESS ) BYTE PUBLIC;

              DCL TOKEN$ADDRESS       ADDRESS;
              DCL       ( TOKEN BASED TOKEN$ADDRESS ) ( 16 )    BYTE;
              DCL       TOKEN$TABLE( * )          BYTE
                        DATA('FSR$', 'MRCV$', 'FSC$', 'ESC$', 'MDC$', 'FSRC$',
                        'MRC$', 'FXG$', 'FANS$', 'HW$', 'GE$', 'TB$', 'RS$', 'MENU$',
                        'TL$', 'DA$', 'EXIT$', '$$' );
              DCL       TOKEN$NUMBER     BYTE;

      DO CASE ( TOKEN$NUMBER := SEARCH( .TOKEN, .TOKEN$TABLE ) );
/* CASE 0 */            CALL FSR;
/* CASE 1 */            CALL MRCV;
/* CASE 2 */           CALL FSC;
/* CASE 3 */            CALL ESC;
/* CASE 4 */            CALL MDC;
/* CASE 5 */            CALL FSRC;
/* CASE 6 */            CALL MRC;
/* CASE 7 */            CALL FXG;
/* CASE 8 */            CALL FANS;
/* CASE 9 */            CALL HW;
/* CASE 10 */           CALL GE;
/* CASE 11 */           CALL TB;
/* CASE 12 */           CALL RS;
/* CASE 13 */           CALL DISPLAY$MENU;
/* CASE 14 */           CALL SET$ORIGIN;
/* CASE 15 */           CALL DISPLAY$ALL;
/* CASE 16 */           RETURN ESCAPE;
/* CASE 17 INVALID */ DO;
                        CALL WRITE$LINE$CRT( .('INVALID', CR, LF, '<$$' ) );
```

```
                    RETURN FALSE;
                    END;

     END;    /* DO CASE */

     RETURN TRUE;

END CALL$ROUTINES;
```

```
VDEMO:
                /* EXERCISE VECTOR DEMONSTRATION COMMANDS */

        PROC PUBLIC;

                DCL     BUFFER( 128 )   BYTE;

        CALL DISPLAY$MENU;

        DO FOREVER;
                CALL WRITE$LINE$CRT( .( CR, LF, '< $$' ) );
                CALL READ$LINE$CRT( .BUFFER );
                IF BUFFER( 0 ) = ESCAPE THEN RETURN;
                ELSE IF BUFFER( 0 ) = CS THEN CALL WRITE$PS( CS );
                ELSE IF BUFFER( 0 ) = CV THEN CALL WRITE$PS( CV );
                IF CALL$ROUTINES( .BUFFER ) = ESCAPE THEN RETURN;
        END;    /* DO FOREVER */

        END VDEMO;

END VECTOR$DEMONSTRATION;
```

```
INIT:
      /* CLEAR SCREEN AND POST ON LINE */

DO;

 EXIT:
      PROCEDURE EXTERNAL;
      END EXIT;

 INITIALIZE$PS:
      PROCEDURE EXTERNAL;
      END INITIALIZE$PS;

CALL INITIALIZE$PS;
CALL EXIT;

END INIT;
```

```
CG$TEST:

DO;

$ INCLUDE( :F1:INIT )
$ INCLUDE( :F1:PSCODE )
$ INCLUDE( :F1:CG )

$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

$ INCLUDE( :F1:CG1.EXT )

DO FOREVER;
      CALL WRITE$CRT( '?' );
      CHAR = READ$CRT;
      IF CHAR = ESCAPE THEN
      DO;
      CALL WRITE$LINE$CRT( .(' I  Q U I T  !$$' ) );
      CALL EXIT;
      END;
      IF CHAR = CS THEN CALL WRITE$PS( CS );
      IF CHAR = CV THEN CALL WRITE$PS( CV );
      CALL CGS;
END;  /* FOREVER */

END;  /* CG TEST */
```

```
TEST$HEX:

DO;

$INCLUDE( :F1:INIT )
$INCLUDE (:F1:CRT.EXT )
$ INCLUDE( :F1:MISS.EXT )

        DCL HEX ADDRESS;
        DCL BIN ADDRESS;
        DCL CHAR BYTE;

DO FOREVER;
CHAR = READ$CRT;
HEX = DISPLAY$HEXADECIMAL( CHAR );
CALL WRITE$LINE$CRT( HEX );
BIN = DISPLAY$BINARY( CHAR );
CALL WRITE$LINE$CRT( BIN );
END;
END TEST$HEX;
```

```
/*WAIT: */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CG.DCL )

$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:MISC.EXT )
$ INCLUDE( :F1:CG1.EXT )
$ INCLUDE( :F1:CG2.EXT )

      DCL      ( X$ORIGIN, Y$ORIGIN )  ADDRESS;
      DCL      CHANGE  ADDRESS;
      DCL      ( SET, COUNT, STOP, RADIUS )    BYTE;
      DCL      ( ROW$NO, COL$NO, X1, Y1 )      ADDRESS;

 WAIT:
               /*  POSTS "CONTINUE?" ON CRT AND WAITS FOR A REPLY FROM KEYBOARD.  */
               /*  CALLS EXIT IF REPLY IS "N" OR "N" */

      PROC;
               DCL      CHAR    BYTE;

      CALL WRITE$LINE$CRT( .('CONTINUE? (Y/N) $$' ) );
      IF ( ( CHAR := READ$CRT ) = 'N' ) OR ( CHAR = 'N' ) THEN
               CALL EXIT;

      END WAIT;
```

```
START:
             /* DRAW A START POINT FOR A VECTOR */

      PROC( X, Y );
             DCL      ( X, Y )          ADDRESS;
             DCL VECTOR( 4 ) BYTE;

      VECTOR( 0 ) = CG;
      VECTOR(1), VECTOR(2), VECTOR(3) = 0;
      VECTOR( 1 ) = LOW( X ) AND 7FH;
      VECTOR( 2 ) = LOW( Y ) AND 7FH;
      VECTOR( 3 ) = HIGH( SHL( Y AND 0180H, 3 ) )
             OR HIGH( SHL( X AND 0180H, 1 ) );
      CALL WRITE$VECTOR( .VECTOR );

      END START;
```

```
DRAW:
                /* DRAWS A VECTOR FROM ANY ORIGIN EXTABLISHED BY TRANSLATE TO THE
                POINTS GIVEN */

        PROC( X, Y );
                DCL       ( X, Y )          ADDRESS;
                DCL       VECTOR( 6 )       BYTE;
                DCL       ( X0, Y0 )        ADDRESS;

        X0, Y0 = 0;
        VECTOR( 0 ) = CG;
        VECTOR( 1 ), VECTOR( 2 ), VECTOR( 3 ) = 0;
        VECTOR( 4 ), VECTOR( 5 ) = '$';

        IF TRANSLATE( .X0, .Y0, FALSE ) THEN
                DO;
                CALL SET$X( X0, .VECTOR( 1 ) );
                CALL SET$Y( Y0, .VECTOR( 1 ) );
                CALL WRITE$VECTOR( .VECTOR );
                END;     /* IF TRANSLATE */
        ELSE
                DO;
                CALL DISPLAY$VECTOR$ATTRIBUTES( .VECTOR( 1 ) );
                CALL WRITE$LINE$CRT( .('TRANSLATE CANNOT SET ORIGIN', CR, LF, '$$' ) );
                END;     /* ELSE DO */

        VECTOR( 3 ) = 20H;        /* SET END */

        IF TRANSLATE( .X, .Y, FALSE ) THEN
                DO;
                CALL SET$X( X, .VECTOR( 1 ) );
                CALL SET$Y( Y, .VECTOR( 1 ) );
                CALL WRITE$VECTOR( .VECTOR );
```

```
            END;    /* IF TRANSLATE */
        ELSE
            DO;
            CALL DISPLAY$VECTOR$ATTRIBUTES( .VECTOR( 1 ) );
            CALL WRITE$LINE$CRT( .('TRANSLATE CANNOT SET END ', CR, LF, '$$' ) );
            END;     /* ELSE DO */

        END DRAW;
```

```
DRAW$COORDINATES:
                /* DRAWS THE NUMBER OF LINES REQUESTED AT THE ORIGINS GIVEN */

      PROC( ORG$X, ORG$Y, LINES );
                DCL      ( ORG$X, ORG$Y, COUNT ) ADDRESS;
                DCL      LINES   BYTE;

      LOOP:  DO COUNT = 1 TO LINES;
                CALL COL( ORG$X + COUNT - 1 );
                CALL ROW( ORG$Y + COUNT - 1 );
      END LOOP;

END DRAW$COORDINATES;
```

```
YXGPH:
    PROC;
        DCL     ( X0, Y0, X, Y, X1, Y1 )          ADDRESS;
        DCL     SET      BYTE;
        DCL     VECTOR( 6 )      BYTE;

    VECTOR( 0 ) = CG;
   VECTOR( 1 ), VECTOR( 2 ), VECTOR( 3 ) = 0;
    VECTOR( 4 ), VECTOR( 5 ) = '$';
    X1, Y1 = 0;
    X, Y = 1;
    X0, Y0 = 255;
    SET = TRUE;

    CALL WRITE$LINE$PS( .( STX, 0, 2,
            'Y  =  (X - X**2 / 16  +  X**3 / 64) / 99', STX, 10, 12,
            'SCALE = 60 PER INCH $$' ) );

    IF TRANSLATE( .X0, .Y0, SET ) THEN
            CALL DRAW$COORDINATES( X0, Y0, 2 );

    CALL SET$X( X0, .VECTOR( 1 ) );
    CALL SET$Y( Y0, .VECTOR( 1 ) );
    CALL WRITE$VECTOR( .VECTOR );
    VECTOR( 3 ) = VECTOR( 3 ) OR SET$END;

    POSITIVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 );
            X, X1 = X1 + 4;
            Y, Y1 = Y1 + 1;
            Y = ( Y - ( Y * Y ) + Y * Y * Y ) / 99;
            IF TRANSLATE( .X, .Y, FALSE ) THEN
                    DO;
                    CALL SET$X( X, .VECTOR( 1 ) );
```

```
                        CALL SET$Y( Y, .VECTOR( 1 ) );
                        CALL WRITE$VECTOR( .VECTOR );
                       END;    /* TRANSLATE */
               END POSITIVE;

      VECTOR( 3 ) = VECTOR( 3 ) XOR VECTOR( 3 );
      CALL SET$X( X0, .VECTOR( 1 ) );
      CALL SET$Y( Y0, .VECTOR( 1 ) );
      CALL WRITE$VECTOR( .VECTOR );
      X1, Y1 = 0;
      X, Y = 1;

      NEGATIVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 );
               X, X1 = X1 - 4;
               Y, Y1 = Y1 - 1;
               Y = - ( ( - ( Y - ( Y * Y )  + Y * Y * Y ) ) / 99 );
               IF TRANSLATE( .X, .Y, FALSE ) THEN
                       DO;
                        CALL SET$X( X, .VECTOR( 1 ) );
                        CALL SET$Y( Y, .VECTOR( 1 ) );
                        CALL WRITE$VECTOR( .VECTOR );
                       END;    /* TRANSLATE */
               END NEGATIVE;
END YXGPH;
```

```
/*MAIN$MVVEC$PROGRAM: */

CALL INITIALIZE$PS;
CALL WRITE$LINE$CRT( .('DO YOU WANT TO STOP AFTER EACH GRAPH? $$' ) );
IF (CHAR := ECHO$CRT = 'N' ) OR ( CHAR = 'N' )
      THEN STOP = FALSE;
      ELSE STOP = TRUE;
CALL WRITE$P( CS );

DO FOREVER;

      CALL WRITE$P( CV );
      DO ROW$NO = 0 TO 511;    /* FILL SCREEN WITH ROWS */
              CALL ROW( ROW$NO );
      END;

      IF STOP THEN CALL WAIT;

      DO ROW$NO = 0 TO 511;    /* MOVE ROW VECTOR DOWN THE SCREEN */
              CALL WRITE$P( CV );
              CALL ROW( ROW$NO );
      END;

      IF STOP THEN CALL WAIT;
      CALL WRITE$P( CV );

      DO COL$NO = 0 TO 511;    /* FILL SCREEN WITH COLUMN VECTORS */
              CALL COL( COL$NO );
      END;

      DO COL$NO = 0 TO 511;    /* ERASE SCREEN A COLUMN AT A TIME */
              CALL COL( COL$NO + 512 );
      END;
```

```
IF STOP THEN CALL WAIT;
CALL WRITE$P( CV );
CALL COL( 511 );

DO COL$NO = 0 TO 511;    /* MOVING DOUBLE COLUMN */
        CALL COL( 511 - COL$NO );
        CALL COL( 1024 - COL$NO );
END;

IF STOP THEN CALL WAIT;
CALL WRITE$P( CV );

DO CHANGE = 0 TO 511;    /* FILL SCREEN TOWARD BOTTOM LEFT CORNER */
        ROW$NO = CHANGE;
        CALL ROW( ROW$NO );
        COL$NO = 511 - CHANGE;
        CALL COL( COL$NO );
END;

IF STOP THEN CALL WAIT;
CALL WRITE$P( CV );

DO CHANGE = 0 TO 511;    /* MOVING CROSS */
        ROW$NO = CHANGE XOR 01FFH;
        CALL ROW( ROW$NO );
        CALL COL( ROW$NO );
        COL$NO = CHANGE XOR 03FFH;
        CALL ROW( COL$NO );
        CALL COL( COL$NO );
END;

IF STOP THEN CALL WAIT;
```

```
Y$ORIGIN = 511;
X$ORIGIN = 0;
SET = TRUE;
CALL WRITE$P( CV );
CALL WRITE$P( CS );

FANS:  DO CHANGE = 5 TO 45 BY 5;          /* DRAW RADIAL LINES FROM BOTTOM CORNERS */
        X = ( 99 - ( CHANGE * CHANGE / 70 ) ) * 5;
        Y = ( 8 * CHANGE / 5 ) * 5;

IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
        DO;
       CALL DRAW( X, Y );
        IF CHANGE <> 45 THEN CALL DRAW( Y, X );
        END;

IF TRANSLATE( .Y$ORIGIN, .Y$ORIGIN, SET ) THEN
        DO;
        CALL DRAW( -X, Y );
        IF CHANGE <> 45 THEN CALL DRAW( -Y, X );
        END;

END FANS;

IF STOP THEN CALL WAIT;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
X$ORIGIN, Y$ORIGIN = 255;
IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, TRUE ) THEN
        CALL DRAW$COORDINATES( 0, 0, 2 );
RADIUS = 0;

HEAT$WAVE:  DO WHILE ( Y > 0 ) AND ( Y < 511 ) AND ( X > 0 ) AND ( X < 511 );
```

```
        RADIUS = RADIUS + 20;
        DO CHANGE = 0 TO 45;
                X = ( 99 - ( CHANGE * CHANGE /70 ) ) * RADIUS / 100;
                Y = ( CHANGE * 8 / 5 ) * RADIUS / 100;
                IF TRANSLATE( .X, .Y, SET ) THEN
                        DO;
                        CALL START( X, Y );
                        CALL START( Y, X );
                        CALL START( -X, -Y );
                        CALL START( -Y, -X );
                        END;
        END;    /* DO CHANGE */
END HEAT$WAVE;

IF STOP THEN CALL WAIT;
CALL WRITE$P( CV );
CALL WRITE$P( CS );
IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, TRUE ) THEN
        CALL DRAW$COORDINATES( 0, 0, 2 );
RADIUS = 0;

GOOSE$EGG:  DO WHILE ( Y > 0 ) AND ( Y < 511 ) AND ( X > 0 ) AND ( X < 511 );
        RADIUS = RADIUS + 20;
        DO CHANGE = 0 TO 45;
                X, X1 = ( 99 - ( CHANGE * CHANGE / 70 ) ) * RADIUS / 100;
                Y, Y1 = ( CHANGE * 8 / 5 ) * RADIUS / 100;
                IF TRANSLATE( .X, .Y, FALSE ) THEN
                        CALL START( X, Y );
                CALL START( Y, X );
                X = -X1;
                Y = Y1;
                IF TRANSLATE( .X, .Y, FALSE ) THEN
                        CALL START( X, Y );
```

```
                    CALL START( Y, X );
                    X = -X1;
                    Y = -Y1;
                    IF TRANSLATE( .X, .Y, FALSE ) THEN
                            CALL START( X, Y );
                    CALL START( Y, X );
                    X = X1;
                    Y = -Y1;
                    IF TRANSLATE( .X, .Y, FALSE ) THEN
                             CALL START( X, Y );
                    CALL START( Y, X );

               END;     /*DO CHANGE */

          END GOOSE$EGG;

          IF STOP THEN CALL WAIT;
          CALL WRITE$P( CV );
          CALL WRITE$P( CS );
          CALL YXGPH;

          IF STOP THEN CALL WAIT;
          CALL WRITE$P( CV );
          CALL WRITE$P( CS );
          X$ORIGIN, Y$ORIGIN = 255;
          SET = TRUE;

          IF TRANSLATE( .X$ORIGIN, .Y$ORIGIN, SET ) THEN
               DO;
               SET = FALSE;
               DO CHANGE = 20 TO 220 BY 20;
                    Y = CHANGE + 20;
                    X = CHANGE;
```

```
                      DO WHILE ( X := X - 20 ) <> -( CHANGE + 20 );
                              CALL DRAW( X, Y );
                      END;

                      DO WHILE ( Y := Y - 20 ) <> -( CHANGE + 20 );
                              CALL DRAW( X, Y );
                      END;

                      DO WHILE ( X := X + 20 ) <> ( CHANGE + 20 );
                             CALL DRAW( X, Y );
                      END;

                      DO WHILE ( Y := Y + 20 ) <> ( CHANGE + 20 );
                              CALL DRAW(X, Y );
                      END;

                      COUNT = LOW( CHANGE );
                      CALL WRITE$HEXADECIMAL( COUNT );

              END;    /* DO CHANGE */
              END;    /* IF TRANSLATE */

      CALL WRITE$LINE$PS( .(STX, 0, 0, 'DO IT AGAIN? $$' ) );

      CALL WAIT;

END;  /* FOREVER */

END MVVEC;
```

```
/*READ$LINE$CRT: */

DO;

$ INCLUDE ( :F1:INIT )
$ INCLUDE ( :F1:CRT.EXT )

             DECLARE BUFFER(123)      BYTE;

 READ$LINE$CRT:       /* READ LINE FROM CRT AND STORE IN BUFFER */
                       /* CONVERT LOWER CASE TO UPPER CASE */

PROCEDURE( BUFFER$ADDRESS );

      DCL     (LBP, CHAR)                BYTE;
      DCL     BUFFER$ADDRESS             ADDRESS;
      DCL     (LINE$BUFFER BASED BUFFER$ADDRESS)(123) BYTE;

LBP = 0;
CHAR = ' ';
LINE$BUFFER( 121 ), LINE$BUFFER( 122 ) = '$';

DO WHILE ((CHAR <> CR) AND (LBP < 120));
      CHAR = READ$CRT;
      IF (((CHAR = BS) OR (CHAR = RUBOUT)) AND (LBP > 0))
              THEN DO;
                      LBP = LBP - 1;
                      CALL WRITE$CRT (LINE$BUFFER( LBP ));
                   END;
              ELSE IF (CHAR = CTL$R)
                      THEN DO;
                              LINE$BUFFER(LBP) = '$';
                              LINE$BUFFER(LBP + 1) = '$';
```

```
                              CALL CRLF$CRT;
                              CALL WRITE$LINE$CRT( BUFFER$ADDRESS );
                           END;
                      ELSE IF (CHAR = CTL$X)
                              THEN DO;
                                      LBP = 0;
                                      CALL CRLF$CRT;
                                   END;
      ELSE DO;
              CHAR = ((((CHAR) AND (MASK$6)) AND (SHR(((CHAR) AND (MASK$7)),1)))
                                                                      XOR (CHAR));
              LINE$BUFFER( LBP ) = CHAR;
              LBP = LBP + 1;
              CALL WRITE$CRT( CHAR );
              END;

END;  /* END DO WHILE */

LINE$BUFFER( LBP ), LINE$BUFFER( LBP + 1 ) = '$';

END;  /* READ$LINE$CRT */

DO FOREVER;
      CALL READ$LINE$CRT(.BUFFER);
      CALL WRITE$LINE$CRT(.BUFFER);
      END;

END; /* READ$LINE */
```

```
FIND:

DO;

$ INCLUDE( :F1:INIT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:MISC.EXT )

      DCL      TABLE( 256 )     BYTE;
      DCL      TPTR    BYTE;
      DCL HEX ADDRESS;
      DCL BUFFER( 123 ) BYTE;
      DCL BPTR BYTE;
      DCL NUMBER( 5 ) BYTE;
      DCL NPTR BYTE;
      DCL      TN       BYTE;


DO FOREVER;
CALL READ$LINE$CRT( .TABLE );
CALL READ$LINE$CRT( .BUFFER );
TN = SEARCH( .BUFFER, .TABLE );
HEX= DISPLAY$HEXADECIMAL( TN );
CALL WRITE$LINE$CRT( HEX );
CALL WRITE$LINE$CRT( .TABLE );
END;
END FIND;
```

```
TYPE:

      /* TYPE OPENS AN UNQUALIFIED INPUT FILE, CONVERTS TABS TO BLANKS AND
      WRITES TO AN OUTPUT FILE.
      TYPE DEFAULTS ARE:
              NO INPUT FILE   --      REQUEST FILE NAME
              NO OUTPUT FILE  --      ASSIGN CONSOLE DEVICE
              NO TAB SIZE     --      TAB SIZE = 3
              NO PAGE SIZE   --      PAGE SIZE = NO PAGING
              NO LINES        --      ON CALL LINES = 0
                                      SUBSEQUENTLY LINES = 1

      ALL PARAMETERS MUST BE IN SEQUENCE GIVEN WITH AT LEAST ONE SEPARATING
      BLANK.  IF OUTPUT IS THE LINE PRINTER, THEN PAGE$SIZE MAY BE SET.
      ANY LINE WITH A STRING OF ALPHANUMERICS, EXCLUDING BLANKS, THAT
      ENDS WITH A COLON, GENERATES AN EXPANDED TITLE LINE AT TOP OF FORM.
      TITLES ARE REPEATED ON EACH PAGE.  TITLES MAY BE AVOIDED BY INCLUDING
      AT LEAST ONE BLANK PRIOR TO THE FIRST OCCURRENCE OF A COLON.

      SAMPLE CALLS:
              T
              T  [F0] PNAME.PLM
              T    ABC.SRC    :F1:ABC.BAK  9999
              T XYZ.ASM :LP:  9
              T         XXX.PLM      :LP:     T9       T99   9999
                                                                        */


      DO;

              DECLARE LIT LITERALLY 'LITERALLY';
              DECLARE DCL LIT 'DECLARE',
                      PROC LIT 'PROCEDURE';
```

```
DCL FALSE LIT '0';
DCL TRUE LIT '1';
DCL CR LIT '15Q';
DCL LF LIT '0AH';
DCL PAGE LIT '0CH';
DCL ESCAPE LIT '1BH';
DCL VT LIT '0BH';
DCL PARITY LIT '0111$1111B';

DCL BO ADDRESS;
DCL BOLIM BYTE;
DCL BI ADDRESS;
DCL LEN BYTE;
DCL LP BYTE INITIAL( FALSE );
DCL PAGE$LINES BYTE INITIAL (0);
                /* TOP OF FORM EXPANDED */
DCL TITLE$LINE( 128 ) BYTE INITIAL( 0CH, 0EH );
DCL BIPTR ADDRESS;
DCL ( TOTAL$CHARS, TOTAL$LINES ) ADDRESS;
DCL BUFFER(128) BYTE;
DCL ( ACTUAL$COUNT, STATUS, AFT$IN, AFT$OUT ) ADDRESS;
DCL READ$ACCESS ADDRESS INITIAL ( 1 );
DCL WRITE$ACCESS ADDRESS INITIAL ( 2 );
DCL ECHOAFTN ADDRESS INITIAL ( 256 );
DCL LINES ADDRESS;
DCL TAB BYTE INITIAL (09H);
DCL TAB$SIZE BYTE INITIAL (3);
DCL TAB$STOP BYTE;
DCL PAGE$SIZE BYTE INITIAL (-1);
DCL IN$BUFF(128) BYTE;
DCL EOD BYTE INITIAL (FALSE);
DCL OUT$BUFF(128) BYTE;
DCL CRT$OUT ADDRESS INITIAL (0);
```

```
        DCL CRT$SIZE BYTE INITIAL ( 80 );
        DCL BYTES$READ ADDRESS;
        DCL CRT$IN ADDRESS INITIAL (1);
        DCL PARM$SIZE ADDRESS INITIAL (100);
        DCL REC$SIZE ADDRESS INITIAL (128);
        DCL EMPTY BYTE INITIAL ( TRUE );
        DCL T$C(30 ) BYTE INITIAL ( CR, LF, ' TOTAL CHARACTERS =         ', CR, LF );
        DCL T$L( 23 ) BYTE INITIAL ( ' TOTAL LINES =        ', CR, LF );

OPEN:
        PROC (AFT, FILE, ACCESS, MODE, STATUS ) EXTERNAL;
                DCL ( AFT, FILE, ACCESS, MODE, STATUS ) ADDRESS;
        END OPEN;

CLOSE:
        PROC ( AFT, STATUS ) EXTERNAL;
                DCL (AFT, STATUS ) ADDRESS;
       END CLOSE;

READ:
        PROC ( AFT, BUFFER, COUNT, ACTUAL, STATUS ) EXTERNAL;
                DCL ( AFT, BUFFER, COUNT, ACTUAL, STATUS ) ADDRESS;
        END READ;

WRITE:
        PROC ( AFT, BUFFER, COUNT, STATUS ) EXTERNAL;
                DCL ( AFT, BUFFER, COUNT, STATUS ) ADDRESS;
        END WRITE;

EXIT:
        PROC EXTERNAL;
                DCL STATUS ADDRESS;
        END EXIT;
```

```
ERROR:
          PROC ( ERRNUM ) EXTERNAL;
                    DCL ( ERRNUM, STATUS ) ADDRESS;
          END ERROR;
```

```
DISPLAY$DECIMAL:
        /* CONVERTS 16 BIT HEXADECIMAL NUMBER TO FIVE ASCII DECIMALS
        AT THE ADDRESS SPECIFIED.
        SAMPLE CALL:
                CALL DISPLAY$DECIMAL( HEX$NUMBER, .PRINTABLE$DECIMAL$NUMBERS )
                */

        PROC( NUMBER, BUFFER$ADDRESS );
                DCL ( NUMBER, BUFFER$ADDRESS ) ADDRESS;
                DCL ( ASCII$NUMBER BASED BUFFER$ADDRESS ) ( 5 ) BYTE;
                DCL ANPTR BYTE;
                DCL ZILCH ADDRESS DATA ( 0 );
                DCL TEN ADDRESS DATA ( 10 );

        ANPTR = 5;

        DO WHILE ( NUMBER <> ZILCH ) AND ( ANPTR > 0 );
                ASCII$NUMBER( ANPTR ) = ( LOW( ( NUMBER MOD TEN ) ) ) OR 30H;
                NUMBER = NUMBER / TEN;
                ANPTR = ANPTR - 1;
        END;

END DISPLAY$DECIMAL;
```

```
HEX:

      /* CONVERTS THE NUMBER OF DECIMAL DIGITS SPECIFIED TO
      A 16 BIT HEXADECIMAL NUMBER.
      SAMPLE CALL:
              XXX = HEX( .ASCII$STRING, NUMBER$OF$DIGITS )
      */

      PROC( CONVERT, DIGITS ) ADDRESS;
              DCL DIGITS  BYTE;
             DCL CONVERT ADDRESS;
              DCL (NUMBER BASED CONVERT) (5) BYTE;
              DCL NI BYTE;
              DCL NHEX ADDRESS;

      IF DIGITS >5 THEN
              DO;
              CALL WRITE(CRT$OUT, .('MORE THAN 5 DIGITS'), 18, .STATUS);
              RETURN 0;
              END;

      NHEX = 0;
      NI = 0;

      DO WHILE (( NI < DIGITS ) AND ( NUMBER( NI ) >= '0' ) AND ( NUMBER( NI ) <= '9' ) );
              NHEX = NHEX * 10;
              NHEX = NHEX + ( NUMBER( NI ) AND  0FH  );
              NI = NI + 1;
      END;

      RETURN NHEX;
END HEX;
```

```
DETRASH:
        /* RETURNS THE NUMBER ( 16 BITS ) OF BYTES  FROM THE LOCATION GIVEN TO
        THE FIRST BYTE THAT IS NOT A BLANK, COMMA, OR SEMICOLON.
        SAMPLE CALL:
                BLANK$COUNT = DETRASH( .BUFFER( 0 ) )
        */

        PROC( BUFFER$ADDRESS ) ADDRESS;
                DCL BUFFER$ADDRESS ADDRESS;
                DCL PTR ADDRESS;
                DCL ( BUFFER BASED BUFFER$ADDRESS ) ( 123 ) BYTE;

        PTR = 0;

        DO WHILE ( PTR < 120 ) AND
                ( ( BUFFER( PTR ) = ',' ) OR ( BUFFER(PTR ) = ' ' )
                OR ( BUFFER( PTR ) = ';' ) );
                PTR =PTR + 1;
        END;

        RETURN PTR;

END DETRASH;
```

```
FIND$BLANK:
      /* RETURNS NUMBER ( 16 BITS ) OF BYTES TO FIRST BLANK.
      SAMPLE CALL:
              WORD$LENGTH = FIND$BLANK( .BUFFER( 0 ) )
      */

      PROC (A) BYTE;
              DCL COUNT BYTE;
             DCL A ADDRESS;
              DCL (BUFFER BASED A) BYTE;

      COUNT = 0;
      DO WHILE (BUFFER <> ' ' AND BUFFER <> CR);
              A = A + 1;
              COUNT = COUNT + 1;
      END;
      RETURN COUNT;
END FIND$BLANK;
```

```
TITLE:
    /* ESTABLISHES TITLE LINE FOR CENTRONICS PRINTER.  IF TITLE LINE IS
    BLANK, TITLE IS SET UP ONE INCH FROM LEFT MARGIN.  OTHERWISE TITLE IS
    SET UP AT RIGHT MARGIN FOR AN 11 INCH PAGE.  TITLES ARE IDENTIFIED BY
    A LINE WHOSE FIRST BLANK CHARACTER IS PRECEDED BY A COLON.  THE
    COLON INDICATES THE END OF TITLE MATERIAL.  TITLE RETURNS TRUE IF THE
    CURRENT LINE IS A NEW TITLE.
    SAMPLE CALL:
            IF TITLE( .BUFFER( 0 ) ) THEN PRINT$IT
    */

    PROC( TITLE$ADDRESS ) BYTE;

    DCL TITLE$ADDRESS ADDRESS;
    DCL ( LINE BASED TITLE$ADDRESS ) ( 128 ) BYTE;
    DCL ( LPTR, LEN, PGM$LEN ) BYTE;

    IF LINE(( LEN := FIND$BLANK( TITLE$ADDRESS ) - 1 )) = ':' THEN
            DO;
            IF TITLE$LINE( 7 ) = ' ' THEN
                    DO;
                    PGM$LEN = LEN;
                    DO LPTR = 0 TO PGM$LEN - 1;
                            TITLE$LINE( LPTR + 7 ) = LINE( LPTR );
                    END;
                    END;

            DO LPTR = 7 + PGM$LEN TO LAST( TITLE$LINE );
                    TITLE$LINE( LPTR ) = ' ';
            END;

            DO LPTR = 0 TO ( LEN - 1 );
                    TITLE$LINE( LPTR + 51 - LEN ) = LINE( LPTR );
```

```
                END;
                RETURN TRUE;
                END;

     ELSE RETURN FALSE;

END TITLE;
```

```
OPEN$FILE:
        /* OPENS FILE SPECIFIED WITH REQUESTED ACCESS MODE IF POSSIBLE.   IF FILE
        NAME IS NOT QUALIFIED; I.E., DRIVE NUMBER NOT GIVEN, THEN BOTH DRIVES ARE
        CHECKED FOR INPUT FILE.   IF FILE IS NOT FOUND, A NEW FILE NAME MAY BE
        ENTERED, HOWEVER THE REST OF THE COMMAND LINE IS USUALLY LOST.   THE ESCAPE
        CHARACTER TERMINATES THE PROGRAM AT ANY REQUEST POINT.   IF OUTPUT FILES
        CAN NOT BE OPENED, THE CONSOLE DEVICE IS SUBSTITUTED.
        SAMPLE CALL:
                FILE$NUMBER = OPEN$FILE( .FILE$NAME, .ACCESS$MODE )
        */

        PROC( FILE$NAME, ACC$ADD ) ADDRESS;

        DCL NI BYTE;
        DCL ( STATUS, X$STATUS, ACTUAL, ACC$ADD, AFT, AFTO ) ADDRESS;
        DCL F1( 14 ) BYTE DATA (':F1:          ');
        DCL FX( 14 ) BYTE;
        DCL NOFILE ADDRESS DATA ( 13 );
        DCL NONAME ADDRESS DATA ( 23 );
        DCL ACCESS BASED ACC$ADD ADDRESS;
        DCL FILE$NAME ADDRESS;
        DCL (NAME BASED FILE$NAME) ( 14 ) BYTE;
        DCL KBD ADDRESS DATA( 1 );
        DCL CRT ADDRESS DATA ( 0 );
        DCL COUNT BYTE;
        DCL LEN ADDRESS;

        AFT = 0;
        AFTO = 0;
        NI = 0;
        LEN = 0;
        X$STATUS = 0;
        STATUS = 1;
```

```
TRY$OPEN:  DO WHILE STATUS <> 0;

        GET$NAME:  DO WHILE(( NI:= NI + DETRASH( NAME( NI ))) >12 )
                        OR ( ( NAME( NI ) AND PARITY ) = CR );

                CALL WRITE( CRT, .('FILENAME? '), 10, .X$STATUS );
                CALL READ( KBD, .NAME, 14, .ACTUAL, .X$STATUS );
                NI = DETRASH( .NAME( 0 ) );
                IF NAME( NI ) = ESCAPE THEN CALL EXIT; /* ESCAPE IS EOF */
                DO WHILE ( ACCESS < 1 OR ACCESS > 3 );
                        CALL WRITE( CRT, .('ACCESS? '), 8, .X$STATUS );
                        CALL READ( KBD, .ACCESS, 2, .ACTUAL, .X$STATUS );
                        ACCESS = HEX( ACC$ADD, 2 );
                END;
        END GET$NAME;

        LEN = FIND$BLANK( .NAME( NI ) );
        CALL OPEN(.AFTO, .NAME, ACCESS, 0, .STATUS );
        IF(( STATUS = NOFILE OR STATUS = NONAME )
                AND (  NAME( NI ) <> ':' )) THEN
        DRIVE$1:  DO;
                CALL WRITE( CRT, .('CHECKING DRIVE 1    ' ), 19, .X$STATUS );
                COUNT = 0;
                DO WHILE COUNT < 14;      /* SET DRIVE # */
                        FX( COUNT ) = F1( COUNT );
                        COUNT = COUNT + 1;
                END;    /* WHILE COUNT < 14 */
                COUNT = 0;
                DO WHILE COUNT < LEN;     /* GET FILENAME */
                        FX(COUNT + 4) = NAME( COUNT + NI );
                        COUNT = COUNT + 1;
                END;    /* WHILE COUNT < LENGTH */
```

```
                      DO WHILE COUNT < 10;       /* BLANK REST OF NAME FIELD */
                              FX(COUNT + 4) = ' ';
                              COUNT = COUNT + 1;
                      END;    /* WHILE COUNT < 10 */

                      CALL OPEN( .AFTO, .FX, ACCESS, 0, .STATUS );     /* TRY DRIVE 1 */
                      IF STATUS > 0 THEN DO;     /* NOT ON DRIVE 1, PRINT MESSAGE */
                              CALL ERROR( STATUS );
                              IF STATUS = NOFILE THEN
                                      CALL WRITE( CRT, .('FILE NOT FOUND '), 15, .X$STATUS );
                              IF STATUS = NONAME THEN
                                      CALL WRITE( CRT, .('IMPROPER FILENAME '), 18, .X$STATUS );
                      END;    /* THEN DO */
                      ELSE DO;        /* ON DRIVE ONE, PRINT FILENAME */
                              CALL WRITE( CRT, .FX, 14, .X$STATUS );
                              CALL WRITE( CRT, .(CR, LF ), 2, .X$STATUS );
                      END;    /* ELSE DO */

              END DRIVE$1;
              ELSE DO;        /* DRIVE 0 */
                      IF NAME( NI ) <> ':' THEN
                              CALL WRITE( CRT, .(':F0:'), 4, .X$STATUS );
                      CALL WRITE( CRT, .NAME( NI ), LEN, .X$STATUS );
                      CALL WRITE( CRT, .( CR, LF ), 2, .X$STATUS );
              END;    /* DRIVE 0 */

      NI = 13;        /* FORCES READ ON SUBSEQUENT LOOPS */

      END TRY$OPEN;

      RETURN AFTO;
END OPEN$FILE;
```

```
/*--MAIN-PROGRAM--: */

      CALL READ( CRT$IN, .BUFFER, PARM$SIZE, .ACTUALCOUNT, .STATUS );
      CALL WRITE( CRT$OUT, .BUFFER, ACTUAL$COUNT, .STATUS );
      BI = DETRASH(.BUFFER( 0 ));
      CALL WRITE( CRT$OUT, .('FILE IN:  '), 10, .STATUS );
      AFT$IN = OPEN$FILE( .BUFFER( BI ), .READ$ACCESS );
      IF AFT$IN = 0 THEN
                  CALL EXIT;

      BI = BI + FIND$BLANK(.BUFFER( BI ));      /* FIND END OF FILENAME */
      BI = BI + DETRASH( .BUFFER( BI ) );      /* FIND 1ST OF NEXT PARM */
      IF  BUFFER( BI ) = ':'
              THEN DO;
              CALL WRITE( CRT$OUT, .('FILE OUT:  '), 11, .STATUS );
              AFT$OUT = OPEN$FILE( .BUFFER( BI ), .WRITE$ACCESS );

              IF ( ( BUFFER( BI + 1 ) = 'L' ) OR ( BUFFER( BI + 1 ) = 'L' ) )
                      AND ( ( BUFFER( BI + 2 ) = 'P' ) OR ( BUFFER(BI + 2 ) = 'P' ) )
                      THEN LP = TRUE;
              BI = BI + FIND$BLANK( .BUFFER( BI ) ); /* END OF OUTPUT FILENAME */
      END;    /* IF : */
      ELSE AFT$OUT = CRT$OUT;

      BI = BI + DETRASH( .BUFFER( BI ) );
      IF ( ( BUFFER( BI ) = 'T' ) OR ( BUFFER( BI ) = 'T' ) )
              THEN DO;
              TAB$SIZE = LOW( HEX( .BUFFER( BI + 1 ), 2 ) );
              IF TAB$SIZE = 0 THEN TAB$SIZE = 1;
              BI = BI + FIND$BLANK( .BUFFER( BI ) );  /* FIND END OF TAB TOKEN */
      END;    /* IF T */

      BI = BI + DETRASH( .BUFFER( BI ) );
```

```
        IF ( ( BUFFER( BI ) = 'P' ) OR ( BUFFER( BI ) = 'P' ) )
                THEN DO;
                PAGE$SIZE = LOW( HEX( .BUFFER( BI + 1 ), 2 ) );
                BI = BI + FIND$BLANK( .BUFFER( BI ) );
                END;

        BI = BI + DETRASH( .BUFFER( BI ) );      /* GET NUMBER OF LINES */
        LINES = HEX( .BUFFER( BI ), 5 );

                        /* INITIALIZE PARAMETERS */
        BYTES$READ = 0;
        BIPTR = 128;    /* BUFFER IN POINTER */
       DO BO = 0 TO LAST( OUTBUFF );
                OUTBUFF( BO ) = ' ';
        END;

        DO BO = 2 TO LAST( TITLE$LINE );
                TITLE$LINE( BO ) = ' ';
        END;

        BO = 10;        /* BUFFER OUT POINTER */
        TOTAL$CHARS = 0;
        TOTAL$LINES = 0;

        DO WHILE ( NOT EOD ) OR ( BIPTR < BYTES$READ );
                DO WHILE ( LINES = 0 ) OR ( LINES > 9999 );
                        CALL WRITE(CRT$OUT,.('LINES? '),7,.STATUS);
                        CALL READ(CRT$IN,.BUFFER,PARM$SIZE,.ACTUAL$COUNT,.STATUS);
                        BI = DETRASH( .BUFFER( 0 ) );
                        IF BUFFER( BI ) = ESCAPE THEN CALL EXIT;        /* ESCAPE */
                       IF BUFFER( BI ) = CR THEN LINES = 1;
                        ELSE
                                LINES = HEX(.BUFFER(BI), 4);
```

```
                IF LINES = 0 THEN LINES = 1;
        END;    /* WHILE LINES 0 */

DO WHILE ( LINES > 0 );
        IF BIPTR > BYTES$READ
        THEN DO;
                IF NOT EOD THEN
                DO;
                CALL READ(AFT$IN, . IN$BUFF, REC$SIZE, . BYTES$READ, . STATUS);
                BIPTR = 0;
                IF BYTES$READ < REC$SIZE
                        THEN
                        EOD = TRUE;
                END;    /* NOT EOD */
                ELSE
                        LINES = 0;

        END;    /* IF BIPTR */

        IF STATUS > 0
                THEN DO;
                CALL ERROR(STATUS);
                CALL EXIT;
                END;

        IF (( BO > 0 ) AND ( OUT$BUFF( BO - 1 ) = LF )) OR ( BO > 126 )
                THEN
                DO;
                IF LP AND ( PAGE$SIZE < 255 )          /* LINE PRINTER PAGING */
                                AND ( ( PAGE$LINES = PAGE$SIZE )
                                OR TITLE( . OUTBUFF( 10 ) ) )
                        THEN DO;
                        CALL WRITE( AFT$OUT, . TITLE$LINE, 128, . STATUS );
```

```
                        PAGE$LINES = 0;
                        CALL WRITE( AFT$OUT, .( ' ', LF, CR, ' ', LF, CR,
                             ' ', LF, CR, ' ', LF, CR ), 12, .STATUS );
                        END;
                ELSE
                        PAGE$LINES = PAGE$LINES + 1;
                CALL WRITE( AFT$OUT, .OUTBUFF, BO, .STATUS );

                DO BO = 0 TO LAST( OUTBUFF );
                        OUTBUFF( BO ) = ' ';
                END;

                LINES = LINES - 1;
                TOTAL$LINES = TOTAL$LINES + 1;
                BO = 10;
                END;

        IF IN$BUFF( BIPTR) = TAB THEN
                DO;
                BIPTR = BIPTR + 1;
                TOTAL$CHARS = TOTAL$CHARS + 1;
                        TAB$STOP = TAB$SIZE - ( BO MOD TAB$SIZE );
                        DO BOLIM = 1 TO TAB$STOP;
                        OUT$BUFF(BO) = ' ';
                        BO = BO + 1;
                        END;    /* TAB */
                END;

        ELSE IF BIPTR <= BYTES$READ THEN
                DO;
                OUT$BUFF(BO) = IN$BUFF(BIPTR);
                IF IN$BUFF( BIPTR ) = 0CH THEN PAGE$LINES = 0;
                BO = BO + 1;
```

```
                    BIPTR = BIPTR + 1;
                    TOTAL$CHARS = TOTAL$CHARS + 1;
                    END;     /* MOVE CHARACTER */
        END;     /* WHILE LINES > 0 */
END;    /* WHILE NOT EOD OR BIPTR < BYTES$READ */

        DO BO = 0 TO LAST( OUTBUFF );
                IF OUTBUFF( BO ) = ' ' THEN
                        EMPTY =EMPTY OR TRUE;
                ELSE EMPTY = FALSE;
        END;

        IF NOT EMPTY THEN
                CALL WRITE( AFT$OUT, .OUTBUFF, BO, .STATUS );

        IF LF THEN
                CALL WRITE( AFT$OUT, .( PAGE ), 1, .STATUS );

        CALL DISPLAY$DECIMAL( TOTAL$CHARS, .T$C( 21 ) );
        CALL WRITE( CRT$OUT, .T$C, 30, .STATUS );
        CALL DISPLAY$DECIMAL( TOTAL$LINES, .T$L( 16 ) );
        CALL WRITE( CRT$OUT, .T$L, 23, .STATUS );

        CALL CLOSE( AFT$OUT, .STATUS );
        CALL CLOSE( AFT$IN, .STATUS );
        CALL EXIT;

END;
```

```
NULL:          /* SENDS NULL CHARACTER TO PLASMA DISPLAY.  CAUSES NO ACTION ON DISPLAY */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( NULL );
CALL EXIT;

END;  /* NULL */
```

```
CH:             /*  HOME CURSOR MOVES CURSOR TO FIRST FOREGROUND CHARACTER ON THE PAGE  */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( CH );
CALL EXIT;

END;  /* CH */
```

```
ETX:   /* ENABLE PLASMA KEYBOARD AND DISPLAY ALPHANUMERIC CURSOR */

DO;

$INCLUDE( :F1:INIT.DCL )
$INCLUDE( :F1:CRT.EXT )
$INCLUDE( :F1:PS.EXT )
$INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )

CALL WRITE$PS( ETX );
CALL WRITE$LINE$CRT( .(CR,LF,'PLASMA KEYBOARD ENABLED',CR,LF, '$$' ) );
CALL EXIT;

END;   /*      ETX      */
```

FS

```
FS:   /* FORESPACE ROUTINE FORCES CURSOR TO NEXT PRINTABLE POSITION */
      /* LEAVING CURRENT POSITION UNCHANGED */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:FS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:FSCODE.DCL )

CALL WRITE$FS( FS );
CALL EXIT;

END;  /*       FS       */
```

```
BS:   /* BACKSPACE ROUTINE - MOVES CURSOR BACK TO FIRST PRINTABLE */
       /* POSITION ON THE CURRENT LINE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL)

CALL WRITE$PS( BS );
CALL EXIT;

END;   /* BS */
```

```
TAB:   /* TAB ROUTINE - MOVES CURSOR FORWARD TO FIRST PRINTABLE POSITION */
       /* FOLLOWING NEXT BACKGROUND DATA, IF THERE IS ONE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )

CALL WRITE$PS( TAB );
CALL EXIT;

END;   /* TAB */
```

```
VT:             /* VERTICAL TAB MOVES LINE CURSOR UP ONE LINE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( VT );
CALL EXIT;

END;  /* VT */
```

```
LF:             /*  LINE FEED ROUTINE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( LF );
CALL EXIT;

END;  /* LF */
```

```
CS:  /*CLEARS ALPHANUMERICS ON PLASMA */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )

CALL WRITE$PS( CS );
CALL WRITE$LINE$CRT( .(CR, LF, 'CLEARED SCREEN--A/N', CR, LF, '$$' ) );
CALL EXIT;

END;  /*       CS       */
```

```
CR:              /* CARRAGE RETURN */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( CR );
CALL EXIT;

END;  /* CR */
```

```
CV:  /* CLEARS VECTORS ON PLASMA */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )

CALL WRITE$PS( CV );
CALL WRITE$LINE$CRT( .(CR, LF, 'CLEARED VECTORS', CR, LF, '$$' ) );
CALL EXIT;

END;  /*       CV       */
```

```
BG:              /* SET PLASMA TO BACKGROUND MODE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )
$ INCLUDE( :F1:PSCODE.DCL )

CALL WRITE$PS( BG );
CALL WRITE$LINE$CRT( .(CR,LF, 'BACKGROUND MODE SET',CR,LF,'$$') );
CALL EXIT;

END;  /* BG */
```

```
FG:             /* SET FOREGROUND MODE */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( FG );
CALL WRITE$LINE$CRT( .(CR,LF, 'FOREGROUND MODE SET',CR,LF,'$$') );
CALL EXIT;

END;  /* FG */
```

```
CB:            /* CLEAR BACKGROUND FILLS BACKGROUND WITH NULLS */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( CB );
CALL EXIT;

END;   /* CB */
```

```
CF:            /* CLEAR FOREGROUND FILLS FOREGROUND WITH NULLS */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( CF );
CALL EXIT;

END;   /* CF */
```

```
SYN:            /* USED WITH SYNCHRONOUS I/O, CAUSES NO ACTION ON DISPLAY */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )


CALL WRITE$PS( SYN );
CALL EXIT;

END; /* SYN */
```

```
CAN:            /* CANCEL REPLACESALL FOREGROUND DATA FROM PREVIOUS */
                /* BACKGROUND DATA WITH NULLS */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( CAN );
CALL EXIT;

END;  /* CAN */
```

```
IR:             /* INSERT RECORD INSERTS BLANK LINE AT CURSOR LOCATION */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( IR );
CALL EXIT;

END;  /* IR */
```

```
DR:             /* DELETE RECORD DELETES LINE AT CURSOR LOCATION */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( DR );
CALL EXIT;

END;  /* DR */
```

```
ICH:            /*  INSERT CHARACTER INSERTS A BLANK AT CURSOR POSITION */
                /*  MOVE FOLLOWING DATA 1 COLUMN TO THE RIGHT */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE( :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( ICH );
CALL EXIT;

END;  /* ICH */
```

```
DCH:            /* DELETE CHARACTER */
                /* MOVES DATA FROM CURSOR POSITION */
                /* LEFT ONE COLUMN                 */

DO;

$ INCLUDE( :F1:INIT.DCL )
$ INCLUDE  :F1:PSCODE.DCL )
$ INCLUDE( :F1:PS.EXT )
$ INCLUDE( :F1:CRT.EXT )
$ INCLUDE( :F1:SYS.EXT )

CALL WRITE$PS( DCH );
CALL EXIT;

END;  /* DCH */
```

LIST OF REFERENCES

1. Pellerin, Sharon, "Graphic Display Systems," Digital Design, pp. 46-59, July 1977.

2. Benwill Staff Report, "Terminals: Crt, Graphic Display and Printing," Digital Design, pp. 55-77, January 1978.

3. Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill Computer Science Series, 1973.

4. Plasma Disolay Set Technical Manual Vol. I, Science Applications Inc., San Diego, California, March 1976.

5. Plasma Disolay Set Technical Manual Vol. II, Science Applications Inc., San Diego, California, March 1976.

6. ISIS-II System Users Guide, Intel Corporation, Santa Clara, California, 1976.

7. 8080/8085 Assembly Language Programming Manual, Intel Corporation, Santa Clara, California, 1977.

8. PL/M-80 Programming Manual, Intel Corporation, Santa Clara California, 1976.

9. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Associate Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
5.	LT Mark S. Moranville, USN, Code 52Mi Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	LT COL Ronald J. Roland, USAF, Code 52Ro Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	LT Ordale P. Babin, Jr., USN 1268 Parkside Place Virginia Beach, Virginia 23451	1
8.	CAPT Ronald R. Seaman, USMC 735 Ramona Monterey, California 93940	1

175381
Thesis
B10517 Babin
c.1 A microcomputer based plasma display system.