PART NO.
062-7456-00

TEKNIQUES
VOL. 7 NO. 4 T2

PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE
PROGRAM EXCHANGE

Tektronix®
COMMITTED TO EXCELLENCE

Prepared by the IDD Program Exchange. The IDD Program Exchange is maintained as a service for our customers by the Information Display Division of Tektronix, Inc., 63-635, P.O. Box 1000, Wilsonville, Oregon 97070 U.S.A.

Tektronix®
COMMITTED TO EXCELLENCE

TEKNIQUES VOL. 7 NO. 4 T2
062-7456-01
DOCUMENTATION

IDD PROGRAM EXCHANGE
63-635
TEKTRONIX, INC.
P.O. BOX 1000
WILSONVILLE, OREGON 97070

1/84

Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
TEKNIQUES VOL. 7 NO. 4 T2	062-7456-01
ORIGINAL DATE: January, 1984 / REVISION DATE:	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM)
AUTHOR	

ABSTRACT

TEKNIQUES VOL. 7 NO. 4 T2 tape consits of 11 programs: one Education/Research, three Graphing, three Interfacing, two Mapping, and two Programming Aids. These programs will run on any of the 4050 Series systems.

One of the programs must be transferred to a 4907 File Manager, and four of the programs must be transferred to their own dedicated tapes. The individual abstracts describe the programs.

Read the documentation before running the programs!

TITLE	PART NUMBER
TEKNIQUES VOL. 7 NO. 4 T2	062-7456-01

TRANSFERRING FILES TO A NEW TAPE

PLOT 50 General Utilities Vol. 1 (TEKTRONIX Part #4050A08) contains a program to transfer any type of 4050 files (program/data/text) quickly and easily along with the header names; however, it requires a 4924 Tape Drive, as does TAPEDUPE program contained on TEKniques Vol. 5 No. 4 T1 tape.

Transferring ASCII or BINARY PROGRAMS without a transfer program

Step 1. Do a TLIST of the MASTER program tape.

Step 2. Record which files go with which program (they are all named) and the size of each file.

Step 3. MARK your new tape to accept the respective files for that program, e.g.,

```
FIND 0
MARK 1,20000
FIND 2
MARK 1,4000
etc.
```

Step 4. Insert the MASTER tape.

```
FIND a file
OLD for ASCII or CALL "BOLD" for BINARY
```

Step 5. Insertthe new tape

```
FIND the file to receive the file in memory
SAVE for ASCII or CALL "BSAVE" for BINARY
```

REPEAT Steps 4 and 5 until all files comprising that program are transferred to the new tape. Note: This procedure will not retain the file header names.

Transferring ASCII or BINARY DATA to a new tape

The 4051R06 Editor ROM could be used to transfer ASCII DATA files.

4050 Applications Library program "Binary Data File Duplicator" will transfer BINARY DATA files without any peripheral.

4050 Applications Library program "Tape Duplication" will transfer ASCII or BINARY DATA or PROGRAM files, but requires a 4924 Tape Drive.

Both of these programs are contained on the 4050 Applications Library UTILITIES T1 tape (TEKTRONIX Part # 062-5974-01), and UTILITIES D1 disk (TEKTRONIX Part #062-5975-01).

Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
Multivariate Bargraph with Enhancements	062-7456-01 - Program 1
ORIGINAL DATE: *12/81* / REVISION DATE: *5/83*	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM): *4050 with 4662 plotter optional (16k memory)*

AUTHOR
Lysander Ng. (803)762-1200
Box 12607, Charleston SC, 29412

ABSTRACT

Multivariate data consisting of one continuous variable in addition to up to four discrete variables may be represented using this program in one output. The discrete variables are denoted as treatment, group, block, and story. If, for example, data only consist of one continuous variable and three discrete variables, then output may for instance be produced with one level of story. Or, we may use just one block level, and so forth. Users have complete latitude in assigning their discrete variables according to the above adopted terminology, depending on desired output format, desired arrangement of meaningful data patterns, and priority of comparisons among various levels of discrete variables.

users have the option of labeling each variable and the graph. If plotter outputs are required, character sizes for the labels may be specified as well. In addition, the y axis label (the continuous variable) may be rotated 90 degrees.

Seven shade-enhancements, in addition to blank (no enhancement), are available to distinguish bars of each treatment level.

As an option, standard deviation of each bar may also be represented.

Users may first view a graph on screen, and after being satisfied with it, direct the graph to the plotter by pressing user-definable-key #1.

1. *PROGRAM OPERATION*

   a. *Overlay*

      after a graph is plotted, user can request to have the graph plotted again on either screen or plotter by key #1.

   b. *Program Loading*

      No special procedures are required.

   c. *Program Execution*

      1. *During interactive prompting session, the program queries as to the range of the continuous variable, number of levels for each discrete variable, labels, character sizes of labels, and corresponding data of continuous variable.*

      2. *Three sample outputs are attached. In Example 1, range of the continuous variable is (0,5), with increment being set at 1. Since the data consists of only three discrete variables, we chose to include only one block level for each story. Data of Example 1 consists of 3 treatment levels (represented by the three enhancements), 2 group levels, and 3 story levels. In Example 2, we have 3 treatment levels, 2 group levels, 2 block levels, and 3 story levels. In Example 3, there are 2 levels each for group, block, and story, while there are 8 treatment levels, each of which corresponds respectively to a numeric designation from 1 through 8. The character sizes of Example 3 are all set about 2 GDU wide and 3 GDU high.*

2. *PROGRAM ORGANIZATION*

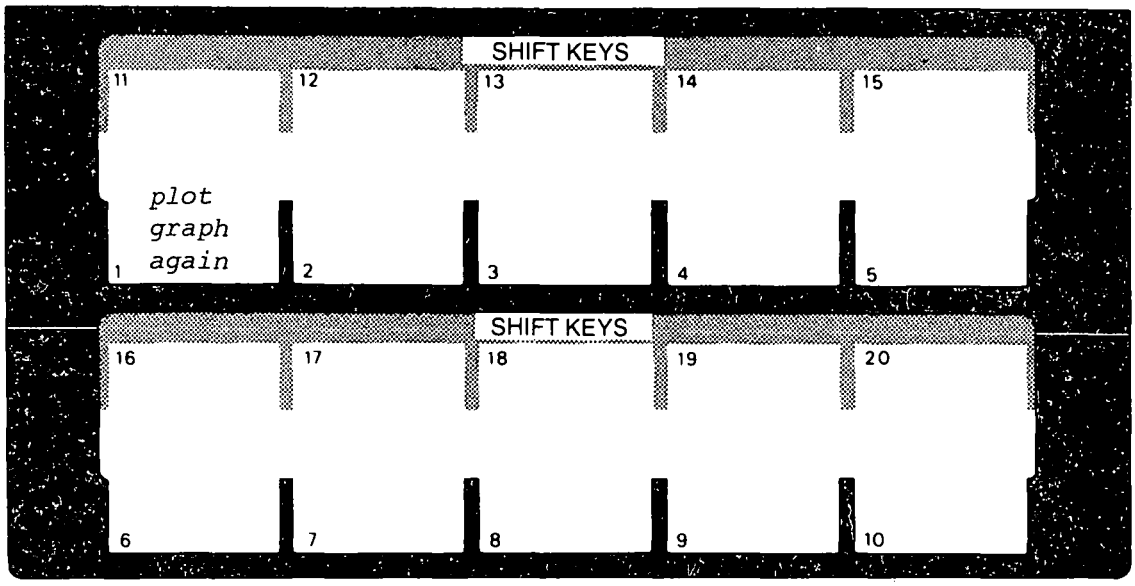
   The program can be divided into two main sections : the first section prompts users for inputs, and the second section plots bargraphs according to the inputs.

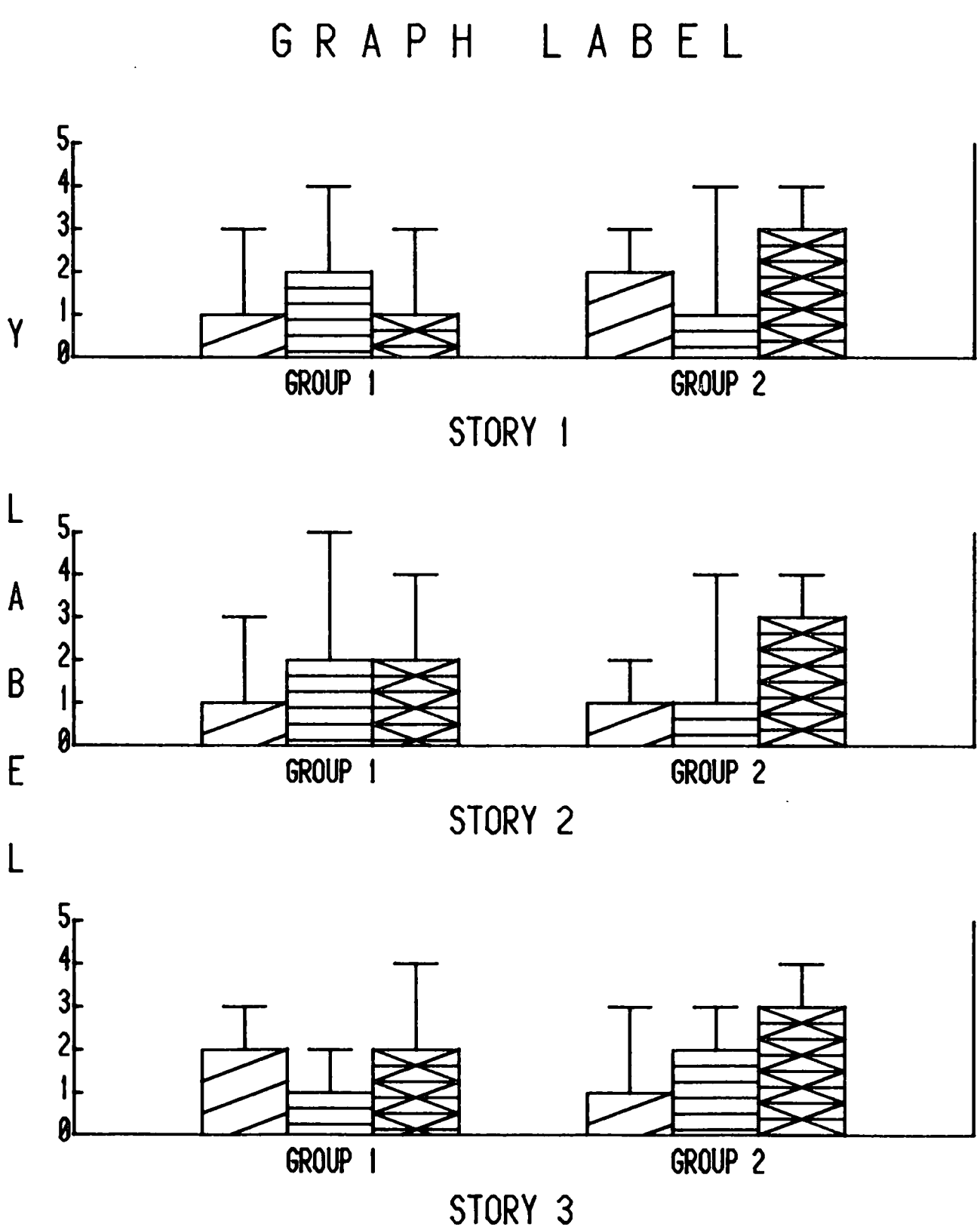
3. INTERNAL DATA STORAGE

LIST OF INPUT VARIABLES

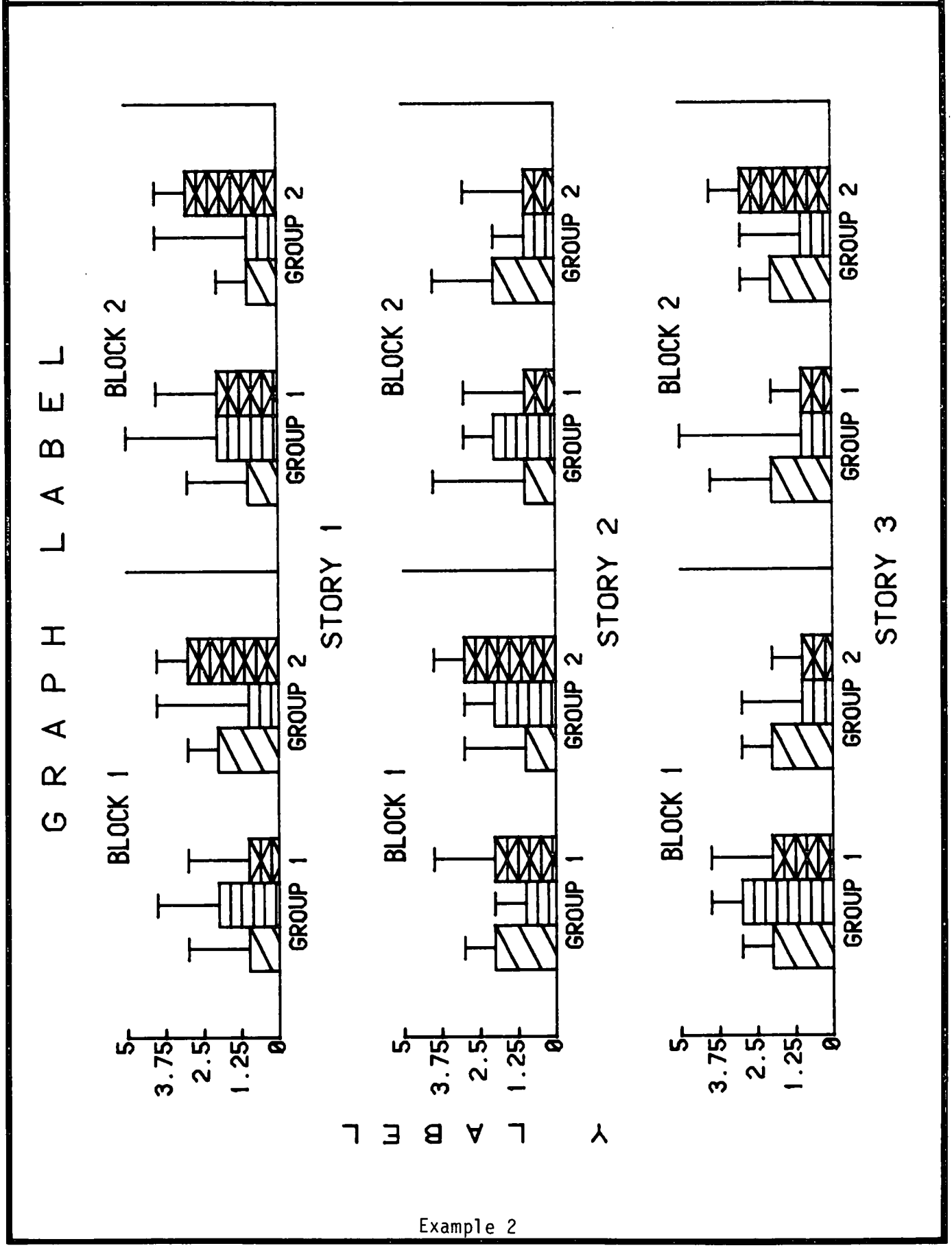
V1	minimum of continuous variable
V2	maximum of continuous variable
V6	tic mark increment on y axis
V7,V8	width and height of character size for y axis numerals respectively
S6	total number of stories
P$	story labels
Y2,Y3	width and height of character size for story labels respectively
B1	total number of blocks at each story
B$	block labels
B2,B3	width and height of character size for block labels respectively
G1	total number of groups in each block
G$	group labels
G2,G3	width and height of character size for group labels respectively
T1	total number of treatments in each group
K(I)	corresponding numeric enhancement for treatment I
X$	y axis label
X2,X3	width and height of character size for y axis label respectively
Z$	graph label
Z2,Z3	width and height of character size for graph label respectively
D	array containing values of continuous variable corresponding to treatments, groups, blocks, and stories
L	standard deviations of the values in array D
O	output location

TITLE TAPE # FILE #

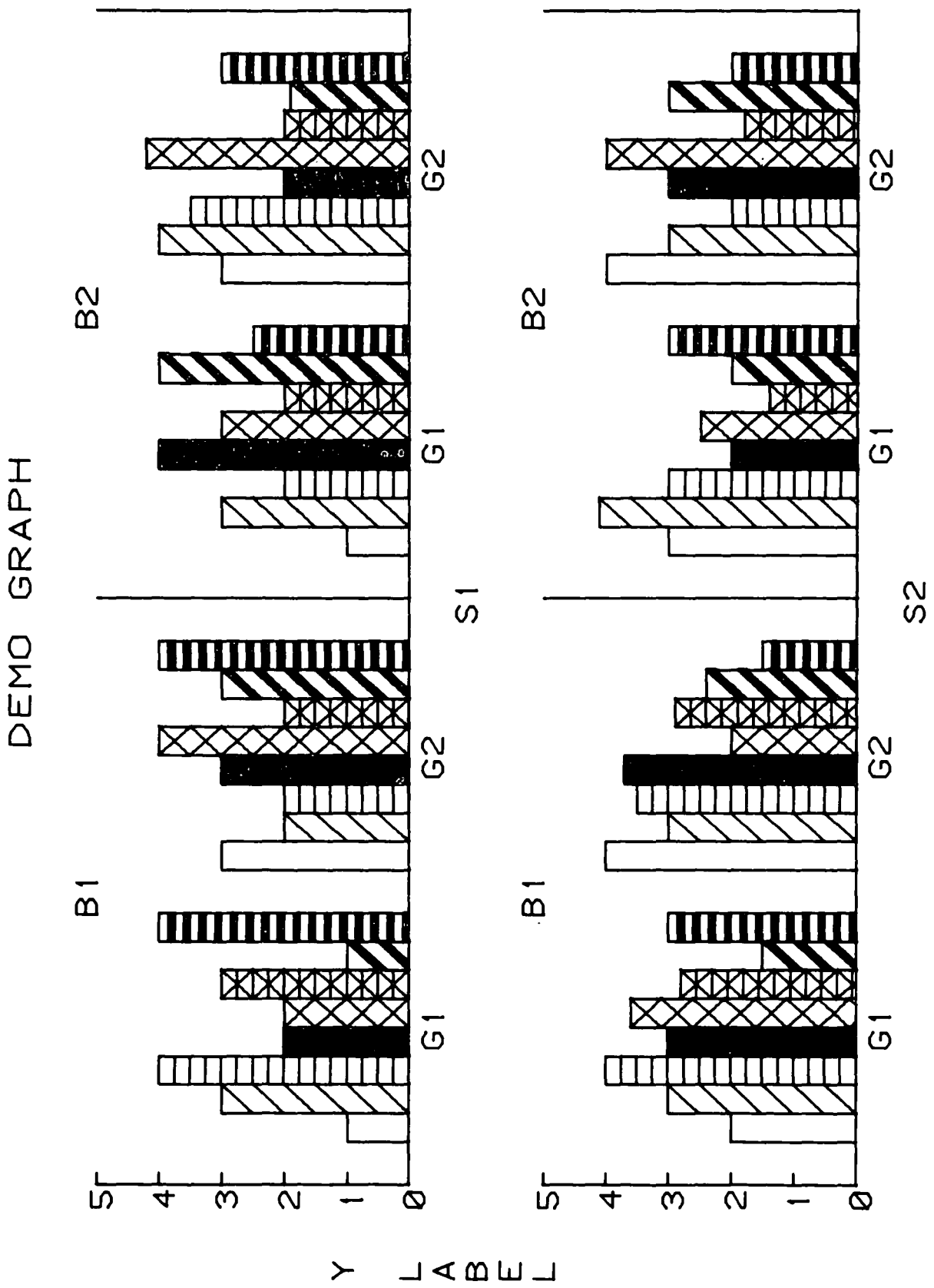



GRAPH LABEL
Y
5
4
3
2
1
0
GROUP 1
GROUP 2
STORY 1
LABEL
STORY 2
STORY 3

Example 1


GRAPH LABEL
BLOCK 1
BLOCK 2
GROUP 1
GROUP 2
STORY 1
STORY 2
STORY 3
5
3.75
2.5
1.25
0
Y LABEL

Example 2



Example 3

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	ABSTRACT NUMBER
Log Axis	062-7456-01 - Program 2
ORIGINAL DATE: 3/1/83 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 4050 Series Computer
AUTHOR: Dr. Jay R Herman Code 964, Goddard Space Flight Center, Greenbelt, Md 20771	PERIPHERALS: None

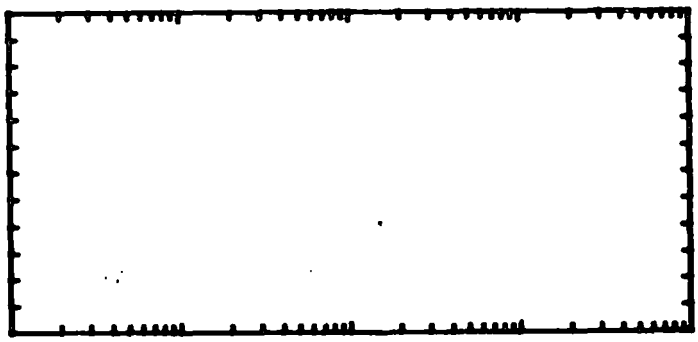
Phone: 301 344 7821

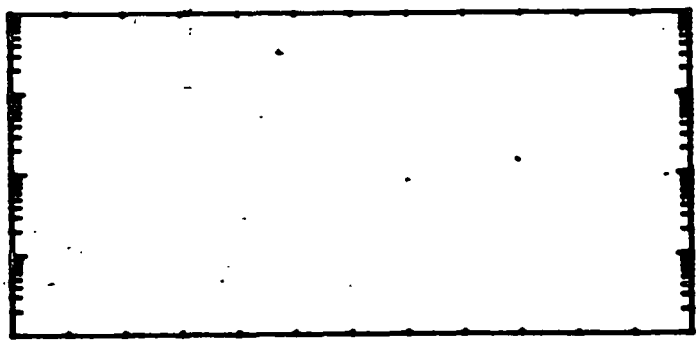
ABSTRACT

The programs Lxaxis and Lyaxis consist of two short sub-programs and two sample drivers to produce logarithmic tic marks on a plot using externally defined WINDOW X1,X2,Y1,Y2, VIEWPORT statements, and axis statements. The size of the tic marks are adjustable using the parameters W and W1. A reverse logarithmic grid can be produced by setting W2=-1 in place of the default W2=0. The two sub-programs in lines 330 to 520 and 770 to 960 are intended to be APPENDED to the users driver program.

1. Equipment required: Any 4050 series computer and plotting devices. Lines 400 and 840 (DASH 0) may have to be removed for the 4051 computer.

2. Sample output: Type (RUN) or (RUN 550). NOTE: UNIT 12 is assumed for the 4662 plotter device number in the drivers, but not in the sub-programs.





TITLE

Log Axis

ABSTRACT NUMBER

```
100 REM ------DRIVER FOR XAXIS ROUTINE------
110 LIST 100
120 LIST 550
130 REM  PLOT LOG TIC MARKS ON ABSCISSA AXIS
140 PRINT "ENTER PLOT DEVICE #= ";
150 INPUT I3
160 PAGE
170 VIEWPORT 0,130+20*(I3=12),0,100
180 REM   "ENTER #S FOR WINDOW COMMAND, X1,X2,Y1,Y2"
190 REM   ASSUME THAT X IS A LOG AXIS FROM 1E2 TO 1E6
200 DATA 2,6,100,220
210 READ X1,X2,Y1,Y2
220 WINDOW X1,X2,Y1,Y2
230 AXIS @I3:1,10,X1,Y1
240 AXIS @I3:1,10,X2,Y2
250 REM  SET TIC MARK SIZE PARAMETERS
260 DATA 2,1,0
270 READ W,W1,W2
280 GOSUB 330
290 GO TO 980
300 REM
310 REM   ----------END DRIVER----------
320 REM
330 REM  START LXAXIS SUBROUTINE
340 REM   DEFAULT FOR W AND W1:   W=1  W1=0.5 (W2=0 OR -1)
350 REM   DEFAULT FOR X1,X2,Y1,Y2:  SET IN EXTERNAL WINDOW STATEMENT.
360 REM W AND W1 CONTROL TIC MARK SIZE. W1 IS FOR THE DECADES.
370 REM W2=0: FOWARD LOG TIC.  W2=-1: REVERSE LOG TIC
380 REM IF X1,X2,Y1,Y2 ARE NOT SPECIFIED, USE THE MOST RECENT WINDOW
390 REM    STATEMENT TO DEFINE X1,X2,Y1,Y2.
400 DASH 0
410 WINDOW X1,X2,Y1,Y2
420 FOR Y0=Y1 TO Y2 STEP Y2-Y1
430    FOR I=X1 TO X2
440       FOR J=1-8*W2 TO 9+8*W2 STEP 1+2*W2
450          MOVE @I3:(1+2*W2)*LGT(J)+I,Y0
460          SCALE 1,1
470          RDRAW @I3:0,(W+W1*(J=1))*(1-2*(Y0<>Y1))
480          WINDOW X1,X2,Y1,Y2
490       NEXT J
500    NEXT I
510 NEXT Y0
520 RETURN
530 REM ----------END LXAXIS ROUTINE----------
540 REM
```

TITLE

Log Axis

ABSTRACT NUMBER

```
550 REM  ------DRIVER FOR YAXIS ROUTINE----------
560 REM  PLOT LOG TIC MARKS ON ORDINATE AXIS
570 PRINT "ENTER PLOT DEVICE #= ";
580 INPUT I3
590 PAGE
600 VIEWPORT 0,130+20*(I3=12),0,100
610 REM   "ENTER #S FOR WINDOW COMMAND, X1,X2,Y1,Y2"
620 REM   ASSUME THAT Y IS A LOG AXIS FROM 1E2 TO 1E6
630 DATA 100,220,2,6
640 RESTORE 630
650 READ X1,X2,Y1,Y2
660 WINDOW X1,X2,Y1,Y2
670 AXIS @I3:10,1,X1,Y1
680 AXIS @I3:10,1,X2,Y2
690 REM SET TIC MARK SIZE PARAMETERS
700 DATA 2,1,0
710 READ W,W1,W2
720 GOSUB 770
730 GO TO 980
740 REM
750 REM   ------------END DRIVER----------------------
760 REM
770 REM  START LYAXIS SUBROUTINE
780 REM   DEFAULT FOR W AND W1:   W=1  W1=0.5  (W2=0 OR -1)
790 REM   DEFAULT FOR X1,X2,Y1,Y2:  SET IN EXTERNAL WINDOW STATEMENT.
800 REM W AND W1 CONTROL TIC MARK SIZE. W1 IS FOR THE DECADES.
810 REM W2=0: FOWARD LOG TIC.  W2=-1: REVERSE LOG TIC
820 REM IF X1,X2,Y1,Y2 ARE NOT SPECIFIED, USE THE MOST RECENT WINDOW
830 REM                  STATEMENT TO DEFINE X1,X2,Y1,Y2.
840 DASH 0
850 WINDOW X1,X2,Y1,Y2
860 FOR X0=X1 TO X2 STEP X2-X1
870    FOR I=Y1 TO Y2
880       FOR J=1-8*W2 TO 9+8*W2 STEP 1+2*W2
890          MOVE @I3:X0,(1+2*W2)*LGT(J)+I
900          SCALE 1,1
910          RDRAW @I3:(W+W1*(J=1))*(1-2*(X0<>X1)),0
920          WINDOW X1,X2,Y1,Y2
930       NEXT J
940    NEXT I
950 NEXT X0
960 RETURN
970 REM          ----------------END LYAXIS ROUTINE----------------
980 REM
```

TITLE	PAGE NUMBER
Log Axis	12

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	ABSTRACT NUMBER
INTERP	062-7456-01 - Program 3
ORIGINAL DATE: AUG 15, 1983 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 4050 8K
AUTHOR: Dr. JAY R. HERMAN	PERIPHERALS:

ABSTRACT

INTERP performs linear interpolation on data stored in vector form. Given a numerically defined function y(x) where the data is stored in the vectors Y and X, and a list of new inddependent variable values stored in the vector D, the linear interpolation

```
C(I)=(Y(J+1)-Y(J)) /(X(J+1)-X(J))*(B(I)-X(J))+Y(J)
```

is solved for all C(I) corresponding to the B(I). The program insures that B(I) falls in the interval between some pair [X(J),X(J+1)], if not then a default value is inserted into C(I). The vectors X and B must be monotonic.

Additional documentation and a working example are contained in the program listing.

No special equipment or peripherals.

Array sizes are specified externally to the subroutine so that all values in X,Y, and C must be defined prior to calling INTERP.

Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
Loran-C Distance and Time Difference Readings	062-7456-01 - Program 4
ORIGINAL DATE: 9-82 / REVISION DATE: 8-83	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM): 4050 with 16K
AUTHOR: Michael A. Lombardi, National Bureau of Standards Boulder, CO	

ABSTRACT

This program computes distances and time difference (TD) readings from any given receiver site to any of the fourteen Loran-C navigation chains in operation worldwide. The user is only required to select a Loran-C chain, and to enter the coordinates of the receiving site (these coordinates may be permanently entered into the program and used as a default value).

The program is useful in determining which Loran-C chain is best for navigation purposes from your location, which stations your receiver has acquired, and whether or not your receiver is tracking these stations on the right cycle. The program can be used by those who use Loran-C to navigate, as well as by those who use Loran navigation receivers for purposes of frequency calibration.

Description of Loran-C

Loran is an acronym that stands for Long Range Navigation. The Loran-C navigation system is composed of groups of radio transmitters, called Loran-C chains, which broadcast continuously, 24 hours a day. There are from three to five stations in each chain, with some stations belonging to more than one chain. One station in each chain is designated as the Master, the others are known as secondaries, or "slave" stations. The secondaries are designated as Whisky (W), Xray (X), Yankee (Y), and Zulu (Z). Signals transmitted by the secondaries are synchronized with the signal transmitted by the Master.

All Loran stations broadcast on the same frequency, 100 kHz. The only way a Loran receiver can distinguish between various chains is by searching for the Group Repetition Interval (GRI) of a specific chain. The GRI is the amount of time that elapses after the beginning of each second before the Master station transmits it's signal. For example, in a Loran-C chain with a GRI of 9940, the Master will transmit at exactly 99,400 microseconds after the beginning of each second. There is a group of cesium oscillators at each Loran-C transmitting site to insure the accuracy of the GRI.

A Loran receiver tuned to a specific GRI will begin to look for the signal from the Master station at the same interval every second. Once it has found the Master, it looks for the secondaries, which transmit at a predetermined delay after the Master station. This delay is a constant, and is coded in such a way that no matter where the receiver is located, the stations will always be received in the same order, Master, W,X,Y, and Z. Once the Loran-C user has

acquired the Master station and two or more secondaries, he can plot his line of position in relation to the Loran stations and thus determine his coordinates.

There are currently fourteen Loran-C stations in operation worldwide. Typically, a chain is usable if you are within 1500 miles of the Master (provided you have a good antenna). The one exception is the Suez Canal chain (4990), which is extremely low power and usable only in the immediate area. If you are able to obtain a strong signal, the results obtained from using Loran-C are extremely accurate.

For a further discussion of Loran-C, consult the following publications. Both were useful in the writing of this program.

Dept. of Transportation, United States Coast Guard, Loran C User Handbook, COMDTINST M16562.3, May 1980.

George Kamas and Sandra L. Howe, Ed., Time and Frequency User's Manual, National Bureau of Standards Special Publication 559, November 1979.

Method of Operation

This program uses the Great Circle method to compute distances. This method involves creating a triangle between the two points in question, and either the North or South pole, and then using geometry to determine the lengths of the sides. The distance obtained is originally in nautical miles, so it is converted to statute miles by multiplying by 1.151. To convert this program from statute miles to kilometers, change variables M and M1 in statement 210 to 3.34141956869 and 1.852, respectively.

Program Operation

To use the program, simply select a Loran-C chain from the menu and enter the coordinates or your receiving site. A new page will be brought to the screen showing distances to the Master and secondary stations and the time difference readings from the secondaries. A time difference reading will not be shown for the master station. This is due to the fact that Loran-C navigation receivers use the signal from the Master station as a reference frequency, and do not display a TD reading for the Master.

You will notice that the coordinates for the National Bureau of Standards in Boulder, Colorado are used as a default value. This is where this program was written and originally used. To insert your own coordinates as a default value, (for example, the coordinates of your boat while docked, your frequency standards lab, your Loran monitoring site, etc.), use the following procedure:

1. Change V$ in statement 210 to the name of your receiving site.

2. Convert your coordinates to decimal form using the following formulas:

```
R1=L1 + (L2 * 60 + L3) / 3600
R2=L4 + (L5 * 60 + L6) / 3600
```

where:
L1 = latitude degrees
L2 = latitude minutes
L3 = latitude seconds
L4 = longitude degrees
L5 = longitude minutes
L6 = longitude seconds

Take the values obtained for R1 and R2 and insert them in place of the NBS coordinates in statements 3070-3080.

3. If your latitude is measured in degrees south, insert this line (Loran reception from the southern hemisphere is unlikely):

```
3084 K0=1
```

4. If your longitude is measured in degrees east, insert this line:

```
3086 L0=1
```

NOTE

The information obtained by using this program is intended to be a useful aid in the use of Loran-C for navigation and other purposes. At the very least, it should allow you to determine which stations your receiver is tracking, and the numbers produced by the program can be assumed to reasonably accurate. However, for a variety of reasons, such as receivers tracking the wrong cycle, atmospheric conditions, equipment delays, weak signals, inexact coordinates, etc., the numbers produced by this program and the numbers displayed by your receiver will almost certainly vary to some degree.

PROGRAM VARIABLES

Variable	Used to Store	Type
A	emission delays for all stations	Array
AO	distance from master to whisky	Simple
BO	distance from master to xray	Simple
C	number used to find coordinates	Simple
CO	distance from master to yankee	Simple
C$	selected Loran-C chain (menu)	String
DO	distance from master to zulu	Simple
E$	GRI of selected Loran chain	String
G	counter	Simple
H$,I$,L$	Y or N question inputs	String
KO	determines if latitude is N or S	Simple
LO	determines if longitude is E or W	Simple
M	path delay for one mile propagation	Simple
M1	nautical to statute mile conversion	Simple
M4,M5	master station coordinates	Simple
M6,M7	sine and cosine of M4	Simple
M$	name of master station	String
O	emission delay locater	Simple
Q	counter	Simple
R1,R2	default receiving site coordinates	Simple
R3,R4	sine and cosine of R1	Simple
T	receiving site coordinates (input)	Array
U,U1,V	scratch math variables	Simple
V1	distance to master	Simple
V2	distance to whisky	Simple
V3	distance to xray	Simple
V4	distance to yankee	Simple
V5	distance to zulu	Simple
V6,V7,V8,V9	time difference readings for second.	Simple
V$	name of default receiving site	String
W1,W2	whisky coordinates	Simple
W6,W7	sine and cosine of W1	Simple
W$	name of whisky station	String
X1,X2	xray coordinates	Simple
X6,X7	sine and cosine of X1	Simple
X$	name of xray station	String
Y1,Y2	yankee coordinates	Simple
Y6,Y7	sine of cosine of Y1	Simple
Y$	name of yankee station	String
Z1,Z2	zulu coordinates	Simple
Z6,Z7	sine and cosine of Z1	Simple
Z$	name of zulu station	String

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	ABSTRACT NUMBER
GPIB GENERAL DEVICE EXERCISER	062-7456-01 - Program 5
ORIGINAL DATE: March, 1982 / REVISION DATE: Oct. 26, 1982	EQUIPMENT AND OPTIONS REQUIRED: 405X with 32K Memory
AUTHOR: John Burgess, Tektronix, 50-352, X1795	PERIPHERALS: IEEE-488 Device Under Test

ABSTRACT

This program provides the facilities to send commands to a device under test (DUT) connected to the IEEE-488 port, and to receive responses from it. Some features are: 1) The Carriage Return character (0D hex) may be sent within the command string by including the characters "**<CR>**"; 2) If a "**?**" is found in the command string, the program will automatically try to get a response from the DUT; 3) Several options are available concerning communication protocol -- use "**ITRM**" or UDK #18; 4) All program control commands are three characters or less, preceded by "**CTRL-I**" -- the TAB character.

The specific commands for these and the other features are provided by a "directions" section within the program. All program control commands prompt the user for input, tell what the default is, and the allowable range of responses, where appropriate.

There are three commands to control the display mode when receiving information from the DUT: 1) Print the ASCII representation of the character with control characters sent unmodified -- for example, **CTRL-L** would page the 405X screen; 2) Same as #1, except the mnemonics of control characters are printed in <> -- for example, **CTRL-L** would be printed as **<FF>**; 3) The decimal value of each character will be printed. In any display mode, if a character whose ASCII value is greater than 127 (DIO8 asserted) is received, its decimal value will be printed. If EOI is asserted with a byte, the characters **< >&EOI** will surround the character. For example, **<FF&EOI>**.

The program can handle up to 9 instruments on the bus at once. Only 1 of the 9 is the "currently active device" (the one all commands are sent to). To change this primary device, enter "**In**"; where "n" is the new device number. There is a command to set up and alter the device list.

In response to the SRQ line being asserted, every instrument on the instrument list is serial polled. There is also a command to get the status byte from the currently active device.

TITLE	ABSTRACT NUMBER
GPIB GENERAL DEVICE EXERCISER	

1. DESCRIPTION

The description on the previous page, along with the program-provided directions should be sufficiently detailed.

2. DATA STRUCTURE

No external data files or additional program files are used.

3. INTERNAL DATA STORAGE

VARIABLE	TYPE	USED TO STORE . . .
A	SIMPLE	When receiving the DUT's response, this contains the decimal value of the byte just read.
A$ (33*4)	ARRAY	Gives the mnemonics of each of the 33 control characters when displaying their mnemonics.
B	SIMPLE	This is the status byte when a serial poll is done.
B1 (8)	ARRAY	Each element will be either 1 or 0; used to display the 8 bits of the status byte
C	SIMPLE	Tells the number of interface/program control commands. Currently set to 28.
C$ (C*4)	ARRAY	Contains the mnemonics for all of the program control commands; used as the lookup table for prog control cómmands.
D0	SIMPLE	Tells which instrument in the instrument list is the "Currently Active Device".
D1 (?)	ARRAY	Contains the primary addresses of the instruments in the instrument list. The size is determined at run-time.
D2 (?)	ARRAY	Contains the secondary addresses of the instruments in the instrument list. As with D2, its size is determined at run-time.

TITLE	ABSTRACT NUMBER
GPIB GENERAL DEVICE EXERCISER	

I0	SIMPLE	FOR/NEXT loop counter used in the "send command" routine.
I1	SIMPLE	FOR/NEXT loop counter when displaying/ altering the device list.
I2	SIMPLE	Another FOR/NEXT loop counter in the device list routine.
I3	SIMPLE	The FOR/NEXT loop counter when polling all devices in response to SRQ.
I4	SIMPLE	Used in FOR/NEXT counter when figuring the 8 bits of the status byte.
I5	SIMPLE	This FOR/NEXT loop counter (last one) is used in the routine (in 2 places) for determining communication protocol.
I$ (75)	ARRAY	After minor processing, contains the command from the keyboard.
M	SIMPLE	Tells which of the 3 display modes is selected. 1 = decimal, 2 = act on control characters, 3 = print control characters' mnemonics.
M0	SIMPLE	After a serial poll is done, tells whether to leave the bus unaddressed (0), or restore the addressed status of the active device (1).
M1	SIMPLE	Maintains the addressed status of the primary device (talk addressed, listen addressed, or idle). Used in conjunction with M0 to restore the addressed status.
N0	SIMPLE	Tells how many instruments are listed in the instrument list.
T$	STRING	Used as a temporary variable for a lot of things (answering questions, etc.)
T	SIMPLE	Another temporary variable.
T0 (15)	ARRAY	Each element will be either 1 or 0; used to control communication protocol. When 1, the corresponding protocol type (send UNT, UNL, etc will be sent; when 0, it won't be sent.
X$ (75)	ARRAY	Inputs the command from the keyboard.

TITLE	ABSTRACT NUMBER
GPIB GENERAL DEVICE EXERCISER	

4. METHODS

Most of the time the program waits patiently for input from the keyboard in response to its "**COMMAND:** " prompt. At this point one of several responses may be provided: 1) Type in a device-dependent command which will be sent to the DUT; 2) Enter "**Icom**" where "com" is either a single digit to select the active device, or a 2 or 3 letter program command mnemonic; or 3) Press a UDK to activate one of their functions. The program then analyzes your input and reacts (hopefully) correctly.

A. INTERFACE COMMANDS: The abbreviations for the interface commands are the same ones used in the IEEE-488 Standard. The routines to send the commands are separate and fairly simple. The Addressed Commands (GTL, SDC, and GET) are preceded by the instrument's listen address and, optionally, its secondary address. After the command itself is sent, the device is left listen-addressed. The Universal Commands (LLO, DCL, SPE, SPD, UNT, and UNT) are sent with just the single byte representing the command. MTA and MLA send the primary Talk or Listen address followed by the optional Secondary Address.

B. SELECT DISPLAY MODE: A group of 3 simple commands (**IDEC**, **IASC**, or **IPCN**) that set the value of M to 1, 2, or 3.

C. DISPLAY/ALTER DEVICE LIST: First the current status is given. This consists of listing the device numbers, primary addresses, secondary addresses, and the "addressed status restoration" mode. The user is then asked if it is desired to alter the status. If no, the routine exists. If yes, the entire status must be re-entered. First it is necessary to enter the number of instruments on the bus (1 to 9). Then the user is prompted for the primary address (1 to 30) and secondary address (0 to 32) of all devices, and the addresses are checked for validity. If there is more than 1 device available, he is asked to select the "currently active device" and, of course, the response is checked for validity. The last question is whether or not he wants the addressed status of the active device restored after a serial poll is done. Finally, the new status is displayed and the opportunity is again given to change it.

D. SEND COMMAND TO DUT: As mentioned earlier, **<CR>** may be included in the device-dependent command to cause the actual Carriage Return character to be sent; the routine first converts all occurrances of **<CR>** into the real CR character. If TAB characters are found in the command string (after the first character), they are replaced with **?**; (it is easier to press the TAB key than SHIFT and ?).

TITLE

GPIB GENERAL DEVICE EXERCISER

ABSTRACT NUMBER

The first 2 communication protocol controls are tested to see if the command should be preceded with UNL and/or MLA [MSA]. Then the command string is sent over the GPIB -- except for the last byte of the message. The next 3 protocol commands are evaluated so the program can determine how to terminate the message (EOI with the last byte, CR, LF, or any combination of the 3). Finally, protocol commands 6 and 7 are tested to see if UNT and/or UNL should be sent.

E. GET RESPONSE FROM DUT: Protocol controls 8 and 9 are tested to see if UNL and/or MTA [MSA] should be sent. Then a byte is read from the GPIB and a lot of things happen to it: 1) the selected display mode is tested, the byte is tested to see if it can be displayed as requested, then it is displayed accordingly; 2) the value of the byte is compared to protocol controls 10 to 13 to see if reception should be terminated. If not, another byte is read and the testing and display are repeated. If reception should be ended, controls 14 and 15 are tested and UNT and/or UNL are sent.

F. SERIAL POLL HANDLING: If SRQ becomes asserted all devices on the device list are polled and the status byte of each is printed. If a poll is requested by use of the program control command, only the "currently active device" is polled. The poll itself does an UNT, UNL, SPE, MTA [MSA], reads the status byte, and sends UNT and SPD. The value of the status byte is analyzed and the 8-bit binary pattern is determined. The device list number, the decimal value of the status byte, and the binary representation of the byte are printed. The nature of a serial poll results in the bus being unaddressed afterward. If it is desired to reconfigure the bus after the poll, the addressed status of the active device will be restored. The status of other devices is ignored; they are always left unaddressed.

G. DIRECTIONS: This simply prints the instructions to the screen. Hopefully the instructions accurately reflect the latest program mods!

H. REVIEW/CHANGE COMMUNICATION PROTOCOL (OVERVIEW): There are 15 options availabels to establish communication control protocol; any combination of the 15 may be selected. The variable T0 is dimensioned to 15; each element will be 1 or 0 to enable or disable the corresponding function. The following chart gives details on their operation:

1) Sending device-dependent messages
   a) T0(1) - precede message with UNL
   b) T0(2) - precede message with MLA [MSA]
   c) send the message (except the last byte)
   d) T0(3) - add a Carriage Return to the tail end
   e) T0(4) - add a Line Feed to the tail end

TITLE	ABSTRACT NUMBER
GPIB GENERAL DEVICE EXERCISER	

f) T0(5) - assert EOI when sending the very last byte
NOTE: any combination of the above 3 may be done, but they will be done in the order given above.
g) T0(6) - send UNT as cleanup
h) T0(7) - send UNL as cleanup

2) Reception of device-dependent messages
a) T0(8) - precede reception with UNL
b) T0(9) - precede reception with MTA [MSA]
c) receive a byte
d) T0(10) - terminate with receipt of Carriage Return
e) T0(11) - terminate with receipt of Line Feed
f) T0(12) - terminate with receipt of FF (hex)
g) T0(13) - terminate if EOI is asserted with byte
NOTE: the above 4 conditions are OR'ed; if a condition which is enabled occurs, reception will end.
h) T0(14) - send UNT as cleanup
i) T0(15) - send UNL as cleanup

I. REVIEW/CHANGE COMMUNICATION PROTOCOL (OPERATION): The current settings are listed and the user is asked is he wants to change anything. If not, the routine ends. If so, 1 of 5 actions may be requested: 1) DEF will establish conditions to emulate standard 405X PRINT and INPUT operation; 2) LF sets things to be readily useable with instruments that use Line Feed as the terminator; 3) EOI sets up to send EOI with the last message byte, and to end reception when a byte with EOI is received; 4) UDF is used for a user-defined default (there is a DATA statement with 15 zero's in it -- the leftmost zero corresponds to T0(1); the rightmost is T0(15) -- as mentioned above, 1 enables the condition, 0 disables it); 5) MAN allows manual selection of values. After the setup is complete, it will be redisplayed for verification.

5. OPERATING INSTRUCTIONS

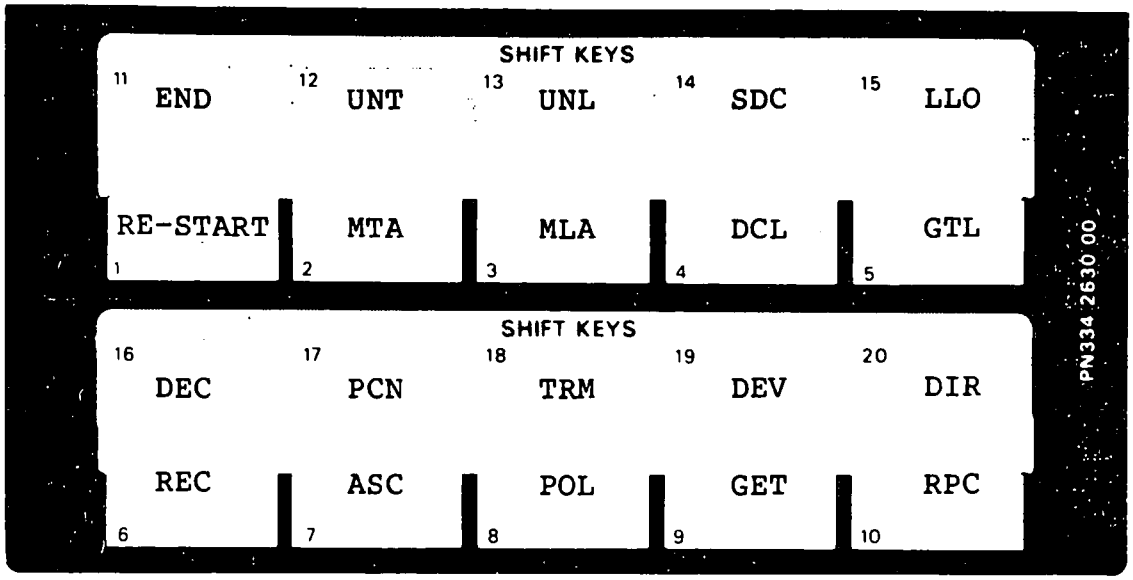
When the program is loaded and run, the first prompt asks whether or not the user wants the directions printed. Directions for operation, including the function of each User Definable Key and each program control command may then be given by the program.

6. REFERENCES

As mentioned the main purpose of this program is for interactively sending and receiving commands to and from IEEE-488 instruments connected to the 405X. Although no knowledge of the IEEE-488 Standard is **necessary**, the user will be able to make better use of the program's features is he at least knows what response to expect from the instrument when some of the interface commands are sent to it.

TITLE GPIB GENERAL DEVICE EXERCISER	ABSTRACT NUMBER

TITLE TAPE # FILE #



GPIB GENERAL DEVICE EXERCISER
BY: John Burgess, TEKTRONIX, Oct. 27, 1982

FOR 405X AS CONTROLLER-IN-CHARGE AND GPIB INSTRUMENTATION

THIS PROGRAM PROVIDES THE FACILITIES TO SEND COMMANDS TO A DEVICE UNDER TEST, AND TO RECEIVE RESPONSES FROM IT. SOME FEATURES ARE: 1) THE CARRIAGE RETURN CHARACTER ($0D) MAY BE SENT WITHIN THE COMMAND STRING BY INCLUDING THE CHARACTERS: "<CR>"; 2) IF A "?" IS FOUND IN THE COMMAND STRING, THE PROGRAM WILL AUTOMATICALLY TRY TO GET A RESPONSE FROM THE DEVICE; 3) SEVERAL OPTIONS ARE AVAILABLE CONCERNING COMMUNICATION PROTOCOL -- (USE "ITRM" OR UDK #18); 4) ALL PROGRAM CONTROL COMMANDS EXCEPT "LF" ARE 3 LETTERS PRECEDED BY CTRL-I (THE TAB).

THERE ARE THREE COMMANDS TO CONTROL THE DISLAY MODE WHEN RECEIVING INFORMATION FROM THE DUT:

"IDEC" (UDK #16) PRINTS THE ASCII VALUE (IN DECIMAL) OF EACH CHARACTER
"IASC" (UDK #7) PRINTS THE ASCII REPRESENTATION OF EACH CHARACTER RECEIVED. CONTROL CHARACTERS ARE ACTED UPON.
"IPCN" (UDK #17) SAME AS "ASC", EXCEPT THE MNEMONIC OF THE CONTROL CHARACTER IS PRINTED.

IN ANY DISPLAY MODE, IF A CHARACTER WHOSE ASCII VALUE IS GREATER THAN 127 IS RECEIVED, ITS ASCII VALUE (IN DECIMAL) IS PRINTED.

THE PROGRAM CAN HANDLE UP TO 9 INSTRUMENTS ON THE BUS AT ONCE. ONLY 1 OF THE 9 IS THE "CURRENTLY ACTIVE DEVICE". TO CHANGE THE PRIMARY DEVICE, ENTER "In"; WHERE "n" IS THE NEW DEVICE NUMBER. THE COMMAND TO DISPLAY OR ALTER THE INSTRUMENT LIST IS "IDEV" (OR UDK #19). IN RESPONSE TO THE 'SRQ' LINE BEING ASSERTED, EVERY INSTRUMENT ON THE INSTRUMENT LIST IS SERIAL POLLED. ONLY THE CURRENT DEVICE IS POLLED IN RESPONSE TO THE "IPOL" COMMAND (OR UDK #8).

<CR> TO CONTINUE:

OTHER PROGRAM CONTROL COMMANDS ARE:
"IDIR" REPEATS THIS LIST OF DIRECTIONS
"IEND" (OR UDK #11) ENDS THE PROGRAM NICELY
UDK #1 PROVIDES AN EASY RESTART AFTER A <BREAK> OR 'END'.

SEVERAL GPIB INTERFACE COMMANDS ARE EASILY SENT:

COMMAND	ATN MESSAGE SENT
"IGTL"	MLA, [MSA], GTL ($01)
"ISDC"	MLA, [MSA], SDC ($04)
"IGET"	MLA, [MSA], GET ($08)
"ILLO"	LLO ($11)
"IDCL"	DCL ($14)
"ISPE"	SPE ($18)
"ISPD"	SPD ($19)
"IMLA"	MLA, [MSA]
"IMTA"	MTA, [MSA]
"IUNL"	UNL ($3F)
"IUNT"	UNT ($5F)

NOTES: 1) [MSA] IS OPTIONALLY USED FOR EXTENDED ADDRESSING.
2) PPC, PPU, IFC AND TCT ARE NOT IMPLEMENTED.

<CR> FOR FEATURE SUMMARY:

P R O G R A M C O N T R O L C O M M A N D S

IDEC	UDK #16	SELECTS DECIMAL DISPLAY MODE
IASC	UDK # 7	SELECTS ASCII DISPLAY MODE (ACTIVE CONTROL CHARACTERS)
IPCN	UDK #17	SAME AS "ASC"; PRINTS CONTROL CHARACTERS' MNEMONICS
IDEV	UDK #19	DISPLAY/ALTER INSTRUMENT LIST
IREC	UDK # 6	MAKES ACTIVE DEVICE TALKER, & 4051 LISTENS
IPOL	UDK # 8	SERIAL POLLS CURRENTLY ACTIVE DEVICE
ITRM	UDK #18	SET UP COMMUNICATION PROTOCOL
IDIR	UDK #20	DISPLAYS THIS FEATURE SUMMARY
IEND	UDK #11	TERMINATES PROGRAM EXECUTION
	UDK # 1	CONVENIENT RESTART AFTER <BREAK>
IRPC	UDK #10	REPEATS PREVIOUS COMMAND
IDEF		SETS DEFAULT PROTOCOL TO SIMULATE 405X I/O
ILF		SETS PROTOCOL TO LINE-FEED MODE
IEOI		SETS PROTOCOL TO EOI MODE
IUDF		SETS THE USER-PROGRAMMED DEFAULT PROTOCOL
IASR		TOGGLES "ADDRESSED STATUS RESTORATION" MODE
In		SETS "ACTIVE DEVICE" TO n

I N T E R F A C E C O M M A N D S

IGTL	UDK # 5	SENDS "MLA, [MSA], GO TO LOCAL"
ISDC	UDK #14	SENDS "MLA, [MSA], SELECTED DEVICECLEAR"
IGET	UDK # 9	SENDS "MLA, [MSA], GROUP EXECUTE TRIGGER
ILLO	UDK #15	SENDS "LOCAL LOCK-OUT" (NO ADDRESSING)
IDCL	UDK # 4	SENDS "DEVICE CLEAR" (NO ADDRESSING)
ISPE		SENDS "SERIAL POLL ENABLE" (NO ADDRESSING)
ISPD		SENDS "SERIAL POLL DISABLE" (NO ADDRESSING)
IMLA	UDK # 3	SENDS "MY LISTEN ADDRESS, [MY SECONDARY ADDRESS]"
IMTA	UDK # 2	SENDS "MY TALK ADDRESS, [MY SECONDARY ADDRESS]"
IUNT	UDK #12	SENDS "UNTALK"
IUNL	UDK #13	SENDS "UNLISTEN"

<CR> TO CONTINUE:

```
FOLLOWING IS A LIST OF AVAILABLE DEVICES, AND THE
CURRENTLY SELECTED ACTIVE DEVICE:


DEVICE NUMBER      PRIMARY ADDRESS    SECONDARY ADDRESS

1                        1                   32

THERE ARE 1 DEVICES AVAILABLE;
#1 IS THE CURRENTLY ACTIVE DEVICE; AFTER A SERIAL
POLL, THE BUS WILL BE UNLISTENED AND UNTALKED.

WOULD YOU LIKE TO ALTER THE LIST?  NOTE: IF YOU ANSWER
"YES", THE ENTIRE LIST MUST BE RE-ENTERED.
```

```
                    4051 OUTPUTTING INFO TO DUT
                   (DEFAULTS SIMULATE 4051 "PRINT")

 1)                 UNL                          >>>>>>   Don't send UNL (Def)
 2)       >>>>>>    MLA (Def)                             Don't send MLA

 3)       >>>>>>    <CR> (Def)                            Don't send <CR>
 4)                 <LF>                         >>>>>>   Don't send <LF> (Def)
 5)       >>>>>>    EOI (Def)                             suppress EOI
 6)       >>>>>>    UNT (Def)                             Don't send UNT
 7)       >>>>>>    UNL (Def)                             Don't send UNL


                    4051 RECEIVING INFO FROM DUT
                   (DEFAULTS SIMULATE 4051 "INPUT")

 8)                 UNL                          >>>>>>   Don't send UNL (Def)
 9)       >>>>>>    MTA (Def)                             Don't send MTA

10)       >>>>>>    <CR> (Def)                            not on <CR>
11)                 <LF>                         >>>>>>   not on <LF> (Def)
12)       >>>>>>    $FF (Def)                             not on $FF
13)       >>>>>>    EOI (Def)                             not on EOI
14)       >>>>>>    UNT (Def)                             Don't send UNT
15)       >>>>>>    UNL (Def)                             Don't send UNL

   NOW ENTER:
 DEF    TO RESTORE THE DEFAULT CONDITIONS,
 LF     TO SET UP FOR LINE-FEED TERMINATION MODE,
 EOI    TO SET UP FOR EOI TERMINATION MODE,
 UDF    TO SET UP USER-DEFINED CUSTOM SETUP,
 MAN    TO INVOKE MANUAL SELECTION,
 <CR>   TO EXIT (NO CHANGES DESIRED).
```

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	ABSTRACT NUMBER
405X/1980 Interface Program	062-7456-01 - Program 6
ORIGINAL DATE: 8-20-82 / REVISION DATE:	EQUIPMENT AND OPTIONS REQUIRED: 405X, Opt.1, UDK Overlay 1980
AUTHOR: William C. Bean, PAE	PERIPHERALS: Hardcopy (optional)

ABSTRACT

Tektronix
The/1980 Automatic Video Measurement Set provides the television facility widely variable measurement capabilities under software control. However, non-volatile storage of program data is limited to the space provided in the 1980's internal Non-volatile memory (8K words) and limits the possibility of transporting data external to the 1980.

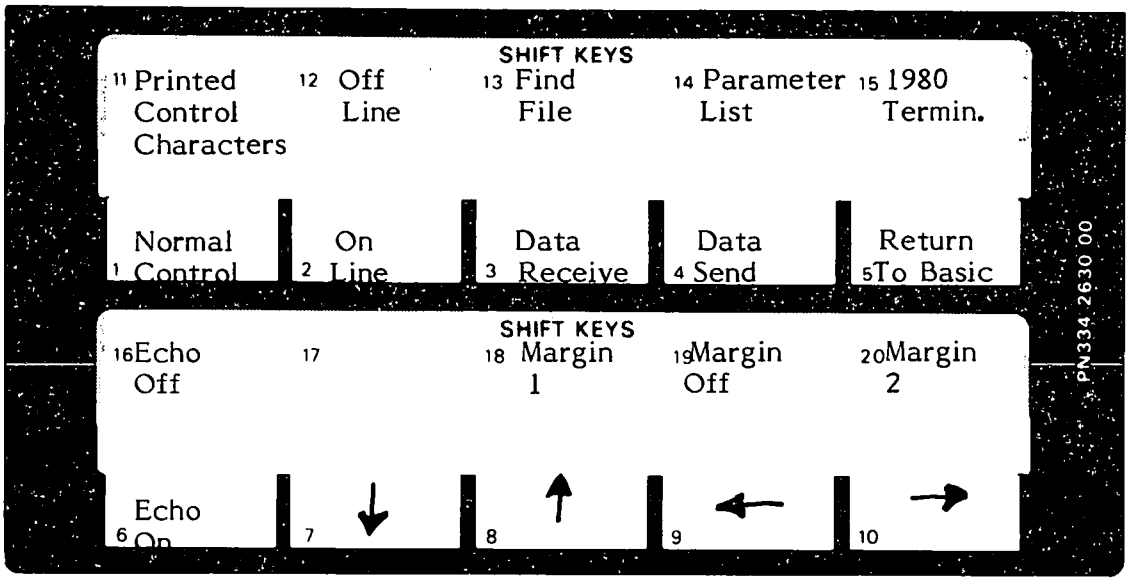
This program provides increased flexibility in non-volatile memory space and transportation by using the tape cartridge in the 405X. The user will be able to save many of his own programs, transport 1980 program patches from one site to another and archive routines previously stored in non-volatile memory for occasions when the memory fails and must be replaced or repaired. In addition, the 405X will be used as the controlling terminal for normal 1980 operations.

TITLE

405X/1890 Interface Program

ABSTRACT NUMBER

TITLE TAPE # FILE #



User Instructions

Basically, initial set-up is connecting the 405X Opt. 1 cable to 1980 port Ø, entering the program into 405X memory and running it. However, some additional items must be considered.

1. The 1980 I/O Interface board must have its master port selection and baud rate jumpers selected to port Ø and a baud rate which the 405X is comfortable with. Refer to the 1980 operators manual (061-2254-01). 2400 baud is a good starting point.

2. This program should be saved on a tape file.

3. Data files for 1980 programs must be allowed for and previously "MARKED". A general rule of thumb is to allow approximately 60-70 bytes of tape space per line of code.

4. The files will be ASCII DATA files, and will not be readily processable by the 405X. (The code is written in "ANSWER BASIC", which has several departures from 4050 BASIC). They will be strictly up-loaded and down-loaded as data files.

Running and Using the Program

Enter the program code and run it. The first prompt to the user will ask for baud rate. Enter the desired baud rate for communication with the 1980. The program will then provide the user with the following selections:

Terminal/Data-Receive Mode - -> UDK -03
Terminal/Data-Send Mode - -> UDK -04
Terminal/Only Mode - -> UDK -15

TITLE	ABSTRACT NUMBER
405X/1980 Interface Program	

From here on out, operation between sub modes is controlled by use of the User-Definable Keys. The standard UDK overlay is applicable. An additional entry on the overlay to define UDK-15 as "1980 TERMIN" is helpful.
(Illustrated in figure)

Entry of UDK-15 will put the 405X into the terminal mode, and it can now exercise the 1980 similar to a standard 4012 terminal. UDK-05 returns you to 405X BASIC. UDK-03 and -04 are available for "modified" Data-Receive and Data-Send sub modes.

Putting 1980 Program Files on Tape

Once a program has been developed and debugged on the 1980, the user may desire to store it on the 405X's tape cartridge. To load the 1980 file on tape, procede as follows:

1. Enter UDK-05 to get back to 405X operation. (you will receive the selection prompt again).

2. Enter UDK-03 (Data Receive). You will be prompted to find a file and enter UDK-03 again. At this time, entry of UDK-13 will allow you to pre-position the tape file and open it. (Note-select on empty file.)

3. Enter UDK-03 again. The 1980 will now start loading its program on to the tape. Once the 1980 responds wiht a "READY*", the data transfer is complete.

4. Enter UDK-05 to get back to 405X operation, where the selection prompt is once again displayed.

Putting Tape Files into 1980

A previously stored 1980 program is retrieved from the tape in the following manner:

1. Enter UDK-05 to get back to 405X operation and the selection mode prompt.

2. Enter UDK-04 (Data-Send). You will be prompted to find a file and enter UDK-04 again. At this time, entry of UDK-13 will allow you to pre-position the tape to the desired file and open it.

3. Enter UDK-04 again. The tape file will now be transferred to the 1980's user memory.

4. Enter UDK-05 to get back to 405X operation, where the selection mode prompt is once again displayed.

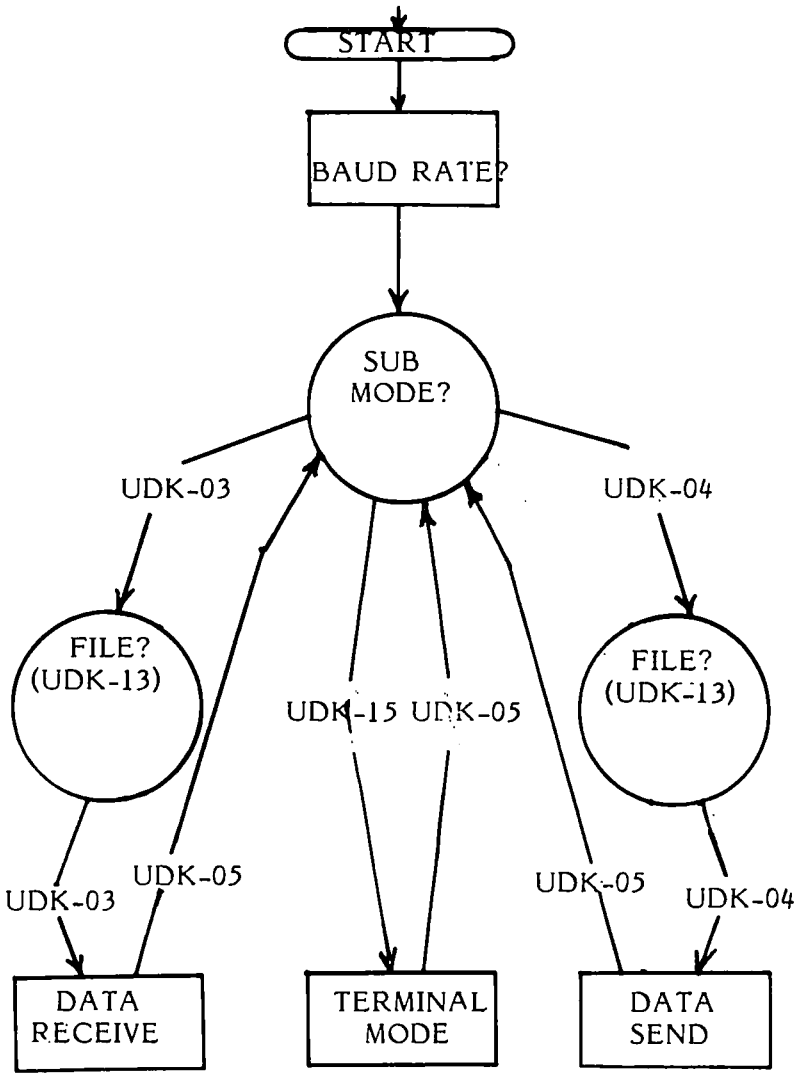
TITLE

405X/1980 Interface Program

ABSTRACT NUMBER

Note - During tape-to-1980 transfers, any data in the 1980's user memory will be over-written. It will usually be best to load the file into "empty" memory, to prevent accidental loss of previously written code.

The requirement to return to 405X BASIC between each sub-mode operation is a limit that is imposed by the necessity to pre-set communications parameters for the 405X-to-1980 interface.



Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
BASIC LINKER	062-7456-01 - Program 7
ORIGINAL DATE: October 21, 1983 / REVISION DATE:	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM): 4050 Series Computer, 4907 Disk Drive
AUTHOR: James A. Bains, Jr., Ph.D.	

ABSTRACT

The LINKER is a set of BASIC programs assigned to link together any number of files to form one BASIC program. A syntax (using REMARK statements) has been defined to allow symbolic labels for BASIC line numbers to be defined and referenced. The LINKER is invoked by running "$LINK". The operator responds to prompts to direct the LINKER. The user's files are concatonated and the symbolic references are resolved. The sections of the LINKER program are automatically paged in and out to allow the maximum size of user program to be concatonated.

The label definitions and references may be left in the linked program or not, as desired. Error messages are given for a variety of error conditions.

PRELIMINARY OPERATING INSTRUCTIONS

The five files comprising this program must be transferred to disk as binary host program files in the SYSTEM library.

Insert the TEKNIQUES VOL. 7 NO. 4 T2 tape into your 4050 Tape Drive.

Insert a formatted disk into UNIT 0 of your 4907 File Manager.

Key in:

```
FIN 8
OLD
SAVE "$LINK"

FIN 9
OLD
SAVE "$LINK1"

FIN 10
OLD
SAVE "$RESOLVE"

FIN 11
OLD
SAVE "$LINKOPER"

FIN 12
OLD
SAVE "$LINKOVER"
```

LINKER OPERATING INSTRUCTIONS

The LINKER uses a "renumberable" and a "non-renumberable" portion. The renumberable portion is made up of a concatonation of one or more files using the BASIC append function. The files used in the renumberable portion may be either ASCII or (host) binary in any combination.

The non-renumberable portion is one file (only) which must be ASCII. To invoke the LINKER, run program "$LINK". In response to the prompts, enter the file I.D.'s of the files which make up the renumberable portion. To stop, hit return. If no renumberable portion is wanted, hit return in response to prompt for first file. Entering a "#", "?", or "*" as the first character will give a directory listing. A USERLIB can be specified (which can be over-ridden by "@"). If no USERLIB is specified, USERLIB defaults to SCRATCHLIB. Choose the beginning line number of the renumberable portion to avoid overlaying the non-renumberable portion.

To label a line number in a BASIC program, put a "REMLBL" statement ahead of it as shown:

```
150 REMLBL LABEL1
160 Z=X*P+Y
```

Leading and trailing spaces around labels are ignored. Embedded spaces are not ignored. To reference a label, put a "REMREF" statement ahead of the line containing the reference as shown:

```
200 REMREF LABEL1
210 GOSUB 1
```

When the LINKER is run, Line 210 will be changed to:

```
210 GOSUB 160
```

Multiple labels may be used in reference statements for those BASIC statements which require them. For example:

```
300 REMREF LABEL1,,120
310 GOSUB X OF 1,250
```

This example illustrates the use of all three reference types.

(1) Symbolic Reference
"LABEL1" is a symbolic reference because it contains a non-numeric character. The number in the referenced BASIC line is changed to the line number specified by a "REMLBL".

(2) Null Reference
If there is no reference, the line number in that position of the BASIC statement is taken.

(3) Numeric Reference
"120" is a numeric reference because all of the characters it contains are numeric. The number in the referenced BASIC line is changed to the numeric reference itself.

When the LINKER is run, Line 310 will be changed to:

```
310 GOSUB X OF 160,250,120
```

The number of arguments (line numbers) is the number in the "REMREF" statement. Only one argument needs to be put on the BASIC statement unless null references are used. Note that the main distinction between the null reference and numeric reference is that the number corresponding to the null reference will be renumbered by an APPEND or RENUMBER command; whereas, the numeric reference will not be.

If the same BASIC line has both a label definition ("REMLBL") and a label reference ("REMREF"), the "REMLBL" must be first. Any number of label names may be given to the same line--simply use more than one "REMLBL" statement as shown:

```
400 REMLBL LABEL3
410 REMLBL LABEL2
420 REMREF LABEL1
430 GO TO 1
```

After the LINKER is run, the linked program will be saved as an ASCII file under the file name given by the operator. If there is no non-renumberable portion, the program is also in the computer's memory. All errors are considered fatal.

To copy the LINKER package, copy "$LINK", "$LINK1", and "$RESOLVE". To copy the documentation, copy "$LINKOPER", and "$LINKOVER". The user files may be on the same disk drive as the system files (the LINKER) or not, as desired.

Two working files are created on the user disk:

(1) "@SCRATCHLIB/LINKFILE" (an ASCII file), and
(2) "@SCRATCHLIB/LABELTABLE" (a binary file).

EXAMPLES

Two files (CONV and CONV1) in library TESTLIB are to be linked to give a completed program (which, incidentally, is a decimal-to-binary and decimal-to-hex converter) to be stored under the file name DUD0. In this example the user files were on a disk on Unit 1, and the system files (the LINKER) were on a disk on Unit 0. (The user files and system files could have been on the same disk.)

The file CONV2 is the same as CONV except that a statement has been removed (causing an error). Listings of the three programs CONV,

CONV1 and CONV2 are given first. Then, the user interaction is shown for three abortive attempts and a successful attempt. After the RUN command, the only things entered by the operator are the characters following the colons. The three abortive attempts are included to show some typical error messages. Finally, a listing of the linked program DUDO is given.

```
UNI1
OLD"@TESTLIB/CONV"
LIST
100 REMLBL START
110 PRINT "ENTER NUMBER"
120 INPUT N
125 REMREF END IT ALL
130 IF N<0 THEN 5
140 REMREF DECBINARY
150 GOSUB 1
160 PRINT N;"D = ";H$;"B"
170 REMREF DECHEX
180 GOSUB 1
190 PRINT N;"D = ";H$;"H"
200 REMREF START
210 GO TO 1
215 REMLBL END IT ALL
220 END
```

```
UNI1
OLD"@TESTLIB/CONV1"
LIST
100 REMLBL DECHEX
110 B=16
120 GO TO 150
130 REMLBL DECBINARY
140 B=2
150 W=N
160 H$=""
180 W=INT(W)/B
190 A1=INT((W-INT(W))*B+0.5)
200 P$=CHR(A1+48)
210 IF A1<10 THEN 230
220 P$=CHR(A1+55)
230 H$=P$&H$
240 IF W=>1 THEN 180
250 RETURN
```

```
UNI1
OLD"@TESTLIB/CONV2"
LIST
100 REMLBL START
110 PRINT "ENTER NUMBER"
120 INPUT N
130 REMREF END IT ALL
140 IF N<0 THEN 5
150 REMREF DECBINARY
160 GOSUB 1
170 PRINT N;"D = ";H$;"B"
180 REMREF DECHEX
190 PRINT N;"D = ";H$;"H"
200 REMREF START
210 GO TO 1
220 REMLBL END IT ALL
230 END

OLD"$LINK"
RUN
ENTER DEVICE NUMBER (UNIT NUMBER) FOR USER FILES: 1
ENTER DEVICE NUMBER (UNIT NUMBER) FOR SYSTEM FILES: 0
ENTER USERLIB: TESTLIB
ENTER FILE I.D.: CONV
ENTER BEGINNING LINE NUMBER OF RENUMBERABLE PORTION: 200
TO CONCATENATE FILES ENTER NEXT FILE I.D.:
TO INCLUDE A NON-RENUMBERABLE ASCII FILE,
ENTER FILE I.D.:
ENTER 'L' TO PURGE 'REMLBL' AND 'REMREF' STATEMENTS,
ENTER 'A' TO PURGE ALL 'REM' STATEMENTS:
ENTER FILE I.D. FOR LINKED FILE: DUD0
UNRESOLVED LABEL REFERENCE DECBINARY

OLD"$LINK"
RUN
ENTER DEVICE NUMBER (UNIT NUMBER) FOR USER FILES: 1
ENTER DEVICE NUMBER (UNIT NUMBER) FOR SYSTEM FILES: 0
ENTER USERLIB: TESTLIB
ENTER FILE I.D.: CONV
ENTER BEGINNING LINE NUMBER OF RENUMBERABLE PORTION: 200
TO CONCATENATE FILES ENTER NEXT FILE I.D.: CONV
TO CONCATENATE FILES ENTER NEXT FILE I.D.:
TO INCLUDE A NON-RENUMBERABLE ASCII FILE,
ENTER FILE I.D.:
ENTER 'L' TO PURGE 'REMLBL' AND 'REMREF' STATEMENTS,
ENTER 'A' TO PURGE ALL 'REM' STATEMENTS:
DUPLICATE LABEL START
```

```
OLD"$LINK"
RUN
ENTER DEVICE NUMBER (UNIT NUMBER) FOR USER FILES: 1
ENTER DEVICE NUMBER (UNIT NUMBER) FOR SYSTEM FILES: 0
ENTER USERLIB: TESTLIB
ENTER FILE I.D.: CONV2
ENTER BEGINNING LINE NUMBER OF RENUMBERABLE PORTION: 200
TO CONCATENATE FILES ENTER NEXT FILE I.D.: CONV1
TO CONCATENATE FILES ENTER NEXT FILE I.D.:
TO INCLUDE A NON-RENUMBERABLE ASCII FILE,
ENTER FILE I.D.:
ENTER 'L' TO PURGE 'REMLBL' AND 'REMREF' STATEMENTS,
ENTER 'A' TO PURGE ALL 'REM' STATEMENTS:
ENTER FILE I.D. FOR LINKED FILE: DUD0
SYNTAX ERROR IN USE OF LABEL REFERENCE DECHEX
BEFORE 290 PRINT N;"D = ";H$;"H"


OLD"$LINK"
RUN
ENTER DEVICE NUMBER (UNIT NUMBER) FOR USER FILES: 1
ENTER DEVICE NUMBER (UNIT NUMBER) FOR SYSTEM FILES: 0
ENTER USERLIB: TESTLIB
ENTER FILE I.D.: CONV
ENTER BEGINNING LINE NUMBER OF RENUMBERABLE PORTION: 200
TO CONCATENATE FILES ENTER NEXT FILE I.D.: CONV1
TO CONCATENATE FILES ENTER NEXT FILE I.D.:
TO INCLUDE A NON-RENUMBERABLE ASCII FILE,
ENTER FILE I.D.:
ENTER 'L' TO PURGE 'REMLBL' AND 'REMREF' STATEMENTS,
ENTER 'A' TO PURGE ALL 'REM' STATEMENTS:
ENTER FILE I.D. FOR LINKED FILE: DUD0
```

```
UNI1
OLD"@TESTLIB/DUD0","A"
LIST
200 REMLBL START
210 PRINT "ENTER NUMBER"
220 INPUT N
230 REMREF END IT ALL
240 IF N<0 THEN 340
250 REMREF DECBINARY
260 GOSUB 390
270 PRINT N;"D = ";H$;"B"
280 REMREF DECHEX
290 GOSUB 360
300 PRINT N;"D = ";H$;"H"
310 REMREF START
320 GO TO 210
330 REMLBL END IT ALL
340 END
350 REMLBL DECHEX
360 B=16
370 GO TO 400
380 REMLBL DECBINARY
390 B=2
400 W=N
410 H$=""
420 W=INT(W)/B
430 A1=INT((W-INT(W))*B+0.5)
440 P$=CHR(A1+48)
450 IF A1<10 THEN 470
460 P$=CHR(A1+55)
470 H$=P$&H$
480 IF W=>1 THEN 420
490 RETURN
```

Tektronix
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
MODIFIED DATA GRAPHING	062-7456-01 - Program 8
ORIGINAL DATE: 1977 / REVISION DATE: 1982	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM): 16K, Optional - 4662 Plotter
AUTHOR: Lynn Cueto — Tektronix, Inc. Rockville, MD	

ABSTRACT

Files: 7 ASCII Program

Statements: 1279

The popular Data Graphing program authored by Chuck Eng has been separated into several overlays so that it may run in a 4051 with 16K memory. Computer expertise is not required since you areprompted for the minimal inputs.

Up to six curves of positive data may be created from:

Keyboard input
Sum of all previous curves
Average of all previous curves
Cumulative sum of the previous curve
Least Squares fit of the previous curves
Data from a tape file

X-axis values may be user-input or auto-sequence. Graph design allows selection of:

Type of curve (bar, solid, phantom, dash) with variations of each
Grid (optional-horizontal)
Labels (user-input or auto-sequencing by month or number)
Hidden line removal
Scaling of Y-axis (automatic or manual)
Saving graph or curve data
Pen color changes on plotter

Other functions allow changes in curve data or graph parameters, listing of data, drawing to the 4050 screen or 4662 Plotter.

Pie Charts and Slides may also be generated by calling in the appropriate programs through the User-Definable Keys.

PRELIMINARY OPERATING INSTRUCTIONS

The files comprising the Data Graphing program must be transferred to a separate dedicated tape.

Following the instructions on page ii, transfer the following files:

From the TEKNIQUES VOL. 7 NO. 4 T2 tape			To the 16K Data Graphing tape		
File #	Bytes	Type	File #	Bytes	Type
13	4608	ASCII Program	1	4608	ASCII Program
14	4864	" "	2	4864	" "
15	4864	" "	3	4864	" "
16	2560	" "	4	2560	" "
17	3840	" "	5	3840	" "
18	6656	" "	6	6656	" "
19	3840	" "	7	3840	" "

Now, mark your file to hold curve data and graph data.

Suggest you reserve files 8 through 13 to hold data for six curves:

Insert the Data graphing tape:

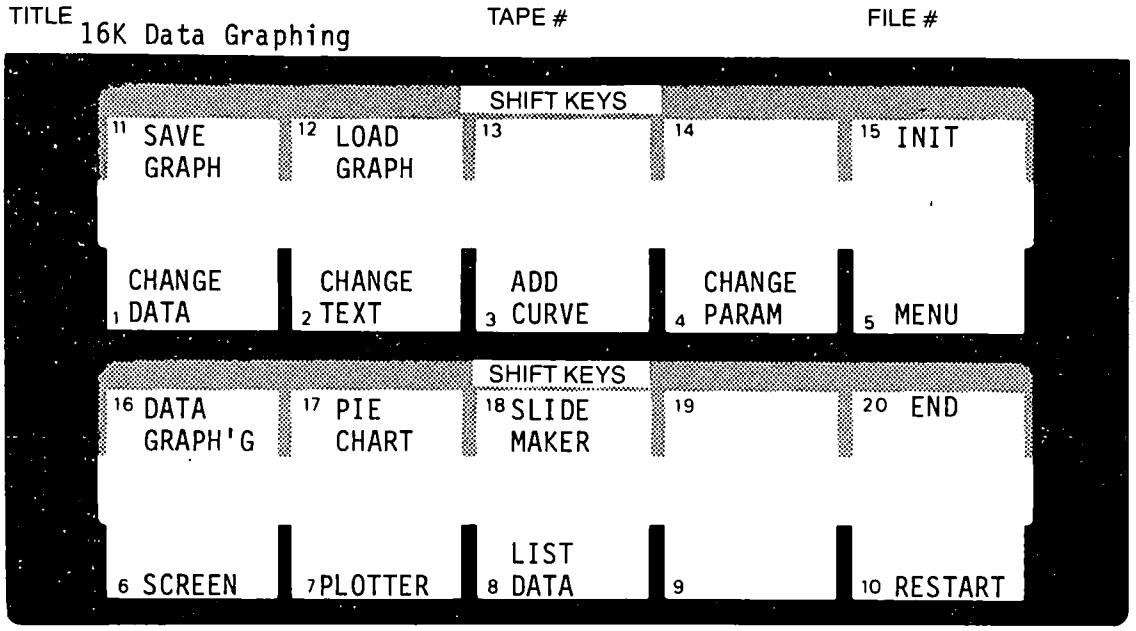
```
FIND 8
MARK 6,768
```

Reserve files 14 through 20 to hold data for seven complete graphs:

```
FIND 14
MARK 7, 3072
```

To insert a check for the correct file number for the curve data, you'll have to change program lines in file 1 - 308 through 314.

To insert a check for the correct file number for graph data, change program lines 12510 through 12559 in file 2.



USER-DEFINABLE KEY FUNCTIONS

#1	CHANGE DATA	- Prompts for the curve #, then the data item to change. Only positive data may be entered. Will store the change on tape if desired.
#2	CHANGE TEXT	- Changes graph title, Y- or X-axis label, curve label(s), or any X-axis tic label.
#3	ADD CURVE	- Prompts for the next curve: name, created from what, curve type, grid, staggered X-axis labels, hidden line removal, Y-axis scaling.
#4	CHANGE PARAM	- Prompts for curve to change: name, curve type; or all curves: grid, staggered X-axis labels, hidden line removal, Y-axis scaling.
#5	MENU	- Returns you to the menu. Data in memory remains intact.
#6	SCREEN	- Graph displayed on the screen.
#7	PLOTTER	- Graph sent to the plotter. Will prompt for pen changes if desired.
#8	LIST DATA	- Lists data for all curves in tabular form.
#10	RESTART	- Initializes the program. Any data in memory will be lost. Prompts you for a new graph.

USER-DEFINABLE KEY FUNCTIONS (continued)

#11 SAVE GRAPH - Prompts you for the number of the file in which to save your graph. (Must be pre-marked.)

#12 LOAD GRAPH - Prompts you for the number of the file from which to load your graph.

#15 INIT - Initializes the program. Any data in memory will be lost. Prompts you for a new graph.

#16 DATA GRAPH'G - Initializes the program (as above).

#17 PIE CHARGE - OLDs in the PIE CHART program and prompts you for pie chart data.

#18 SLIDEMAKER - OLDs in the SLIDEMAKER program and prompts you for text, character size or box/line/diamond and position on plotter. (Plotter required to run this program.)

DATA FILE STORAGE

To store data on tape, the files must be pre-marked. (See PRELIMINARY OPERATING INSTRUCTIONS.)

Variables Stored

N	- number of X-axis values
M$(72)	- X-axis label
S$(72)	- Graph subtitle
X$(1)	-
U$(72)	- Graph title
L$(N*16)	- X-axis tic labels
D$(16*6)	- Curve titles
E$(72)	- Y-axis label
D4(6)	- dash type for curve
N1	- number of curves
Z	- output device num-er
S	- Y-axis upper limit
G	- grid flag
W	- horizontal spacing for labels/titles
H2	- vertical spacing for labels/titles
S5	- staggered label flag
H7	- hidden line removal flag
N9	- comparative bar flag
S1(7)	- Y-axis tic mark control
Z4(6,4)	-
D(6)	- shading type
B2(6)	- type of curve (solid, dash, etc.)
B5(N)	
Y(N,6)	- X-axis data values
N4(6)	- type of data (keyboard input, sum of previous curve, etc.)

OPERATING INSTRUCTIONS

Insert the program tape into the 4050 tape drive and press AUTOLOAD.

Press RETURN to begin your graph. You will be prompted for all data and graph parameters. Almost every parameter has a default which is specified or underlined. If the default is okay with you, press RETURN to respond.

If you enter something wrong, leave it and finish your graph. Then change it using the UDK edit routines (see USER-DEFINABLE KEY FUNCTIONS on previous pages).

Data for the curves may be: entered from the keyboard, input from a file on tape, or calculations of previous curves. Once data is entered for a curve, it may be saved on tape.

The maximum number of data points retrieved from the curve data file can only be the number of points saved; however, less than the full set of points may be retrieved. For example, suppose you specified four X-axis values. If a curve data file containing 10 points is selected as the source of data, only the first four points will be used, but there must be at least four points.

Curve Type - Bars may be stacking (one on top of another) or comparative (side-by side).

Phantom curves remove the plotting of a curve from the graph; however, the curve data will remain and be included in any summation or save function.

Grid Lines - You may choose to have horizontal grid lines drawn on your graph.

Staggered X-Labels - If X-axis labels overlap, they will be automatically staggered; however, you may elect to have them staggered if they don't overlap.

Hidden Line Removal - Used with stacking bars; only curve data in excess of that in the previous curve(s) will be displayed.

Scaling (Y-Axis) - Automatic scaling sets the maximum axis scale at the maximum data value entered. Manual scaling allows you to override the automatic scale value and enter your own. NOTE: Since only positive data is allowed, the minimum axis scale is always set at zero.

NOTE: When generating transparencies for overlays, use the manual scaling so the Y-axis values remain constant for each transparency.

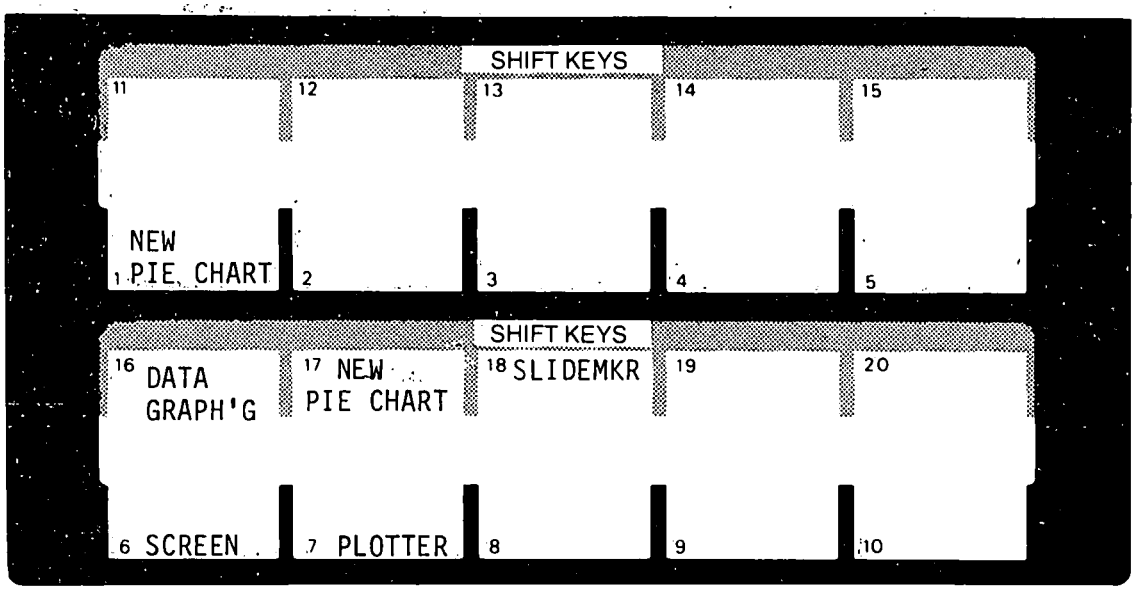
Changing Text - All newly created graphs will carry default labels bearing Tektronix legends. Press UDK #2 to change the graph labels.

Storing Graph Data - Individual curve data is stored as your data is entered. However, to store an entire graph, which includes the curve data along with the graph parameter data, you must press UDK #11.

See PRELIMINARY OPERATING INSTRUCTIONS for marking data files.

The rest of the User-Definable Keys are pretty well explained in the overlay section.

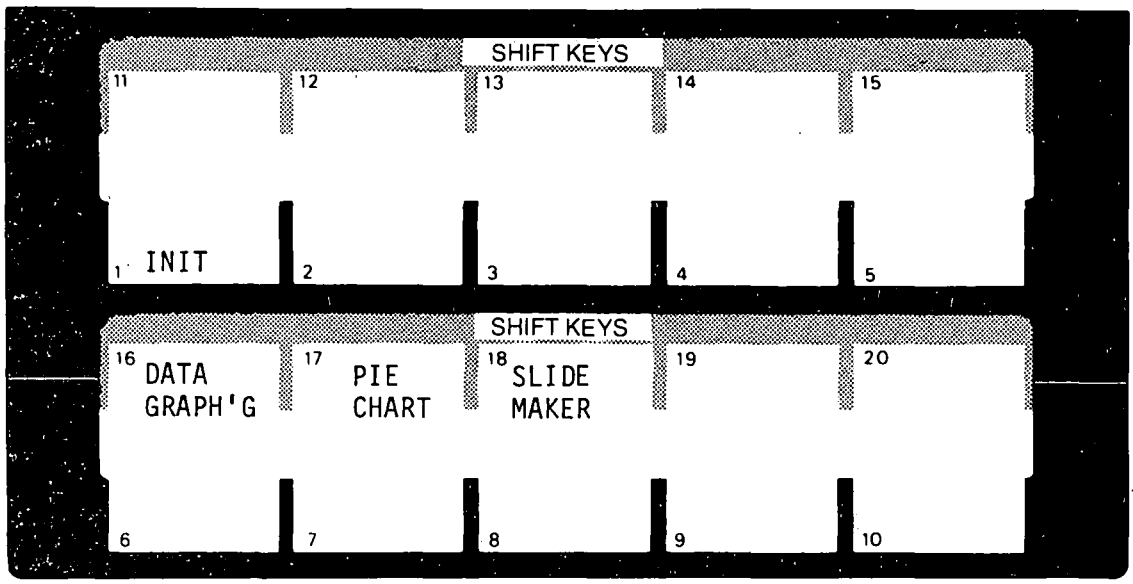
TITLE Pie Chart TAPE # FILE #



USER-DEFINABLE KEY FUNCTIONS

1	BEGINS PIE CHART PROCESS
6	PLOTS ON SCREEN
7	PLOTS ON PLOTTER
16	OLDS IN DATA GRAPHING PROGRAM
17	BEGINS PIE CHART PROCESS
18	OLDS IN SLIDEMAKER PROGRAM

TITLE SLIDEMAKER TAPE # FILE #



USER-DEFINABLE KEY FUNCTIONS

#1	INIT	- Initializes the program. Prompts you for text and character size, or diamond/box/line and position on plotter.
#16	DATA GRAPHING	- OLDS in the DATA GRAPHING PROGRAM and runs it.
#17	PIE CHART	- OLDS in the PIE CHART program and runs it.
#18	SLIDEMAKER	- Restarts the SLIDEMAKER program.

SLIDEMAKER - OPERATING INSTRUCTIONS

The plotter must be interfaced to the 4050, loaded with paper, turn on and the LOAD button up.

Once you have entered the final parameters for your SLIDEMAKER function, that function will be output to the plotter. There is no provision for storing your data.

Nine functions are possible:
- TITLES
- BOX
- LINE
- DIAMOND
- POSITION
- RIGHT JUSTIFY
- DISPLAY VALUES
- CENTER TEXT
- DUPLICATE LAST ENTRY

Text Functions

Text functions include TITLES, POSITION, RIGHT JUSTIFY and CENTER TEXT.

If you enter T, P, R or "CR" (simply press RETURN for the latter), you will be prompted for the text of a line.

After enter the text, you will be asked for the character size by X for width and Y for height. Any fraction of a size may be input. If the size desired is too large for the text to fit into the space allocated, the 4050 will ask for the size again.

Next you will be prompted for placement on the plotter:

TITLE - Position the pen at the desired Y axis position, and press RETURN. The title will be centered from this position. The text will be rewritten six times for a bold effect. After execution, the pen will be repositioned one line below this beginning point.

POSITION - Position the pen for the left margin desired. This left margin will be retained for subsequent lines of text if 'P' is designated each time and the defaults used for positioning (pressing RETURN without physically positioning the pen).

RIGHT JUSTIFY - Position the pen to the desired right margin. The text is then printed to the left of this position. The pen will end at this margin and be moved down one line ready for the next line of text. This right margin will be retained for subsequent lines if 'R' is designated each time and the defaults used for positioning.

CENTER TEXT - Position the pen at the left of the page at the desired Y-axis position. Press RETURN. The text will be centered from this position.

Symbol Functions

A box, diamond or line of any size may be drawn.

After choosing your symbol, you will be prompted for positioning on the plotter.

BOX - Move the plotter pen to the desired lower left corner of the box and briefly press the CALL button on the plotter. Then move the plotter pen to the desired upper right corner of the box and briefly press the CALL button on the plotter. The bell indicates that the 4050 acknowledge the positioning. The box will be drawn in this location.

LINE - As above, you will be prompted for the position, in this case the beginning of the line and ending of the line.

DIAMOND - Positioning is required as in 'B' and 'L', however, for the diamond, the top and right corner is required.

Display Values

Used for Text Functions.

Displays the current X and Y scale factors for the text, the current pen position, the current text and the previous function.

Duplicate Last Entry

Will re-plot the last entry. You need enter an abbreviated position, since it will keep the same scale of the text, diamond, etc.

Aborting a Function

If you wish to abort a function (other than text entry), type NO instead of a number or instead of pressing the RETURN key. Both the N and O are required. If NO is entered and RETURN pressed, the menu will be displayed. The current width, height and position will not be affected.

T = TITLES

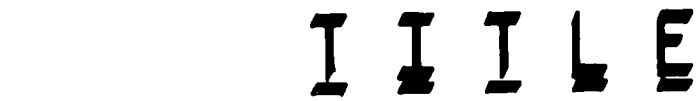
Selecting T draws a centered title. This differs from the centered text option (RETURN) by being drawn five times, slightly offset each time. The effect is dense lettering.

Double spacing produces a dramatic effect.

You may underscore or overstrike text using Control-H (backspace). Allow for accurate centering when using Control-H, i.e., allow twic the number of spaces at the beginning of your text as there are Control-H characters in your text.

```
T= TITLES    B= BOX              L= LINE                    D= DIAMOND
P= POSITION  R= RIGHT JUSTIFY    V= DISPLAY VALUES  "CR"= CENTER TEXT : T

 ENTER TEXT OF LINE :            TH_ IH_ TH_ LH_ EH_
ENTER X FACTOR : 4.8
ENTER Y FACTOR : 4.8
POSITION PEN TO STARTING POSITION AND HIT 'RETURN'
```



P= POSITION

The intent is to create a column of text on the page as if the Plotter had a resettable left margin for carriage returns. Once 'P' is selected and the pen is placed at the beginning position, after the text is printed, the pen will move back to that point and down one line.

A LEFT MARGIN
IS SET BY USING
'P'.

USING THE PLOTTER
FOR TEXT IS EASY!

R= RIGHT JUSTIFY

This establishes a right margin and is useful for alighment of dollar figures in tables of columns. Like 'P', once 'R' is selected, the pen is placed at the desired margin--the far right margin in this case. The text will be printed on the page so that the pen will end at this margin. The pen is then moved down one line ready for the next right justified text.

RIGHT JUSTIFY
USING 'R'.

$ 10.00
25.00
.25

IT'S EASY TO LINE UP
COLUMNS OF FIGURES

"CR"= CENTER TEXT

If the 'RETURN' key is pressed without any keyboard entry, the text will be centered on the page.

The pen will be returned to the left margin and down one line ready for the next entry.

CENTERING TEXT IS DEFAULT

WITH THIS FEATURE
PROFESSIONAL SLIDES
ARE EASY!!!

TRY IT

Tektronix
COMMITTED TO EXCELLENCE

DESKTOP COMPUTER APPLICATIONS LIBRARY PROGRAM

TITLE	ABSTRACT NUMBER
Stereonet Plot (Equal Area)	062-7456-01 - Program 9
ORIGINAL DATE 9/24/82 / REVISION DATE	EQUIPMENT AND OPTIONS REQUIRED: 405 (32K) w/RS-232
AUTHOR Steve Wilson, Dept. Earth Sciences University of Calif., Riverside, CA 92521	PERIPHERALS: 4662 Plotter

ABSTRACT

Statements: 764

Files: 4 ASCII Program

A general graphics program for plotting and manipulating geologic (or any other 3-D orientation data expresed as direction and amount of dip) data (eg. fault planes, joints pebble orientation, paleocurrents, petrofabrics, paleomagnetic poles) on a Equal Area Projection.

Features include:

1. Data storage/recall.
2. Data rotation.
3. Vector mean calculation
4. Least squares best fit great circle.
5. Data editing (add, delete, change, list).
6. Plotting on a 10 cm radius net with:
   - 3 symbols for pole to plane data
   - 2 symbols for lineation data
   - 1 symbol for vector mean
   - 1 symbol for best fit pole including cyclographic trace.

TITLE	ABSTRACT NUMBER
Stereonet Plot (Equal Area)	

Description

This program allows the user to input data from the keyboard, store the data on tape, plot the results, and then manipulate the data. Each program in this group of four is directly accessible from either program. Each program must be modified if the programs are not filed in sequental order, starting with file number 1 on the tape. This modification is very easy; the modifications are as follows:

program #	line numbers
1	14010-14040, 14080
2	1010-1040, 1080
3	6010-6040, 6080
4	5010-5040, 5080

The first group of each program should be set to the file number (print statements). The second group of each program should be set to the file number (if statement).

Data tape structure

File type: binary

Files used: defined by user

Format: N,X,Y,Z,P (#data points, array,array,array, array)

The data can be stored on the same or seperate tape. The files are marked by the programs (however they can be pre marked if you want to). When questioned (data output only)"Is this file new or old", new means tape will be marked, old means tape is pre-marked (user, or program). If not questioned the tape will be marked (ie; program 1).

PRELIMINARY OPERATING INSTRUCTIONS

Although you may change the code to reflect different locations for these files, it is suggested that they occupy the first four locations on tape. Transfer the files from the TEKNIQUES VOL. 7 NO. 2 program tape to a separate dedicated tape:

FROM TEKNIQUES VOL. 7 NO. 2 tape			TO STEREONET PLOT tape		
File #	Bytes	Type	File #	Bytes	Type
20	6656	ASCII Program	1	6656	ASCII Program
21	2048	" "	2	2048	" "
22	7424	" "	3	7424	" "
23	4608	" "	4	4608	" "

TITLE	ABSTRACT NUMBER
Stereonet Plot (Equal Area)	

Variable	Used to store	Type
A()	x coordinate for plotting circle	A(361) array
A1	azimuth of data before transformation	simple
B()	y coordinate for plotting circle	B(361) array
A3	azimuth angle for plotting cyclographic trace	simple
C	output device number (screen or plotter)	simple
D	dip of data before transformation, and angle for use in drawing stereonet	simple
E	Data entry and file number	simple
E6	Controls which program to run	simple
E8()	x coordinate for symbol number 3	array (37)
E9()	y coordinate for symbol number 3	array (37)
I	counter in FOR/NEXT loops	simple
J	counter in FOR/NEXT loops	simple
L	dummy variable	simple
N	number of data elements in file	simple
P()	data type contained in X,Y,Z	P(N) array
X()	the X-axis coordinate in 3-space	X(N) array
Y()	the Y-axis coordinate in 3-space	Y(N) array
Z()	the Z-axis coordinate in 3-space	Z(N) array
X1()	X-axis plotting coordinate	X1(N) array
Y1()	Y-axis plotting coordinate	Y1(N) array
Z2	dummy variable	simple
X3	x coordinate for cyclographic trace	simple
Y3	y coordinate for cyclographic trace	simple
Z3	z coordinate for cyclographic trace	simple
X4	x coordinate for plotting cyclographic trace	simple
Y4	y coordinate for plotting cyclographic trace	simple
Z4	z coordinate for plotting cyclographic trace	simple

the following variables are special variables used in programs 2,3,4, or have different meanings in the other programs.

TITLE	ABSTRACT NUMBER
Stereonet Plot (Equal Area)	

For Rotation Program (program #2)

E	data file number	simple
E1	data file number (output)	simple
E2	file type(new or old)	simple
S1	side angle of rotation (y-axis)	simple
T1	top angle of rotation (z-axis)	simple
X2()	rotated coordinate	(N) array
Y2()	rotated coordinate	(N) array
Z2()	rotated coordinate	(N) array

For Data Editing Program (program #3)

A,D	same as in #1 but simple	simple
A1	intermediate value of A()	simple
A$,B$,C$,F$,G$	name used to identify data type in output	simple string
B$	(line 1080) dummy	simple string
B$()	st,nd,rd,th— used after counter (ie; 1st,2nd,3rd,4th)	B$(2)
E	input data file number	simple
E1	options available for program	simple
E2	output device number	simple
E2	number of data points to be added to file	simple
E2	index of item to be changed	simple
E2	do you want to change another point	simple
E2	output file number	simple
E3	is this a new or old file	simple
E3	number of data points to be deleted	simple
E4	index of data point to be deleted	simple
I1	index for B$()	simple
X1()	temporary storage for X	(N) array
Y1()	temporary storage for Y	(N) array
Z1()	temporary storage for Z	(N) array
P1()	temporary storage for P	(N) array

For Vector Mean and best fit Pole program (program #4)

A,B	used in calculating least squares fit	simple
C1-C5	used in calculating least squares fit	simple

TITLE
Stereonet Plot (Equal Area)

ABSTRACT NUMBER

E	input file number	simple
E	output file number	simple
E1	option number	simple
E3	new or old file	simple
N1	equal to N but does not change	simple
T	square root of the sum of the squares of X1,Y1,Z1	simple
X1,Y1,Z1	sum of the X,Y,Z coordinates	simple
X1,Y1,Z1	coordinates of the best fit pole	simple
X1(),Y1(),Z1(),P1()	temporary storage for X,Y,Z,P	(N+1) array
X2,Y2,Z2	the x,y,z coordinates of the vector mean	simple
X2(),Y2(),Z2(),P2()	temporary storage for X,Y,Z,P	(N+1) array

Methods

The plot is an equal area, lower hemisphere projection. The following sign convention is used for the axis. X-axis (horizontal) right= positive, left= negative. Y-axis (vertical) up= positive, down= negative. Z-axis (in and out of screen) in= positive, out= negative. Note, the Z-axis sign convention is different than normal. The reason is that the equal area net is a lower hemisphere plot.

The following equations are used to calculate x,y,z given the amount of dip and the direction of dip:

$x=R\cos(D)\sin(A)$, $y= R\cos(D)\cos(A)$, $z= R\sin(D)$

where R= radius of the net (10cm), D= dip (lineation) or 90-dip (pole to plane) , and A= azimuth (lineation) or 180 + azimuth (pole to plane).

The data is converted into x,y,z coordinate so that rotations, vector mean, and least squares best fit great circle calculations can be done by normal methods. The transformation of the polar coordinates of a point on a Schmidt net defined by its azimuth A, and radial distance r, into rectangular components x1,y1 gives x1= r sin A, y1=r cos A where $r = 2^{\frac{1}{2}} R \sin(45-\frac{D}{2})$ these are the equations used for plotting the data after the x,y,z components are converted back to A and D. This conversion (arcsin, arccos) leads to the error of ± 1-2° in plotting.

TITLE
Stereonet Plot (Equal Area)

ABSTRACT NUMBER

The method used for calculating the vector mean is as follows.

$$X1 = \frac{\text{sum } X}{\left((\text{sum } X)^2 + (\text{sum } Y)^2 + (\text{sum } Z)^2 \right)^{\frac{1}{2}}}$$

$$Y1 = \frac{\text{sum } Y}{\left((\text{sum } X)^2 + (\text{sum } Y)^2 + (\text{sum } Z)^2 \right)^{\frac{1}{2}}}$$

$$Z1 = \frac{\text{sum } Z}{\left((\text{sum } X)^2 + (\text{sum } Y)^2 + (\text{sum } Z)^2 \right)^{\frac{1}{2}}}$$

where X1,Y1,Z1 are the coordinates of the vector mean.

The method used for calculating the best fit great circle is to find the pole to a great circle that is normal (perpendicular) to the data set. This is accomplished by the method of least squares (minimize the deviation of the squares) where:

$C1 = \text{sum} (X \times Y)$

$C2 = \text{sum} (X \times Z)$

$C3 = \text{sum} (Y \times Z)$

$C4 = \text{sum} (X^2)$

$C5 = \text{sum} (Y^2)$

$$A = \frac{(C1 \times C3) - (C2 \times C5)}{(C4 \times C5) - C1^2}$$

$$B = \frac{(C1 \times C2) - (C3 \times C4)}{(C4 \times C5) - C1^2}$$

$$Z1 = \left(1 + A^2 + B^2 \right)^{-\frac{1}{2}}$$

$X1 = A \times Z1$

$Y1 = B \times Z1$

where X1,Y1,Z1 are the coordinates for the best fit pole to the great circle.

TITLE

Stereonet Plot (Equal Area)

ABSTRACT NUMBER

Operating instructions

Program Loading:

Find N (File number the program is located on)

Old

RUN

It is advisable that programs are groupd together (ie; file 1,2,3,4).
It is advisable to execute program 1 first (" Stereonet Program Plotting Routine ") until a data file has been established.
The computer will ask the user to " Enter Program Number ", for the first time enter 1 (or wherever file number 1 is located). The computer will respond with " DATA: ENTER FROM KEYBOARD=1 OR DATA FILE(BINARY) =2:"
for the first run enter 1 (after data files have been established enter 1 or 2).
The program then asks for the number of data points and the data. The data in all these programs is to be separated by a comma. The direction and amount of dip for a pole to plane data point, for example 220/30 , would be entered as 220,30,n where n is the symbol type (1,2,3).
If the following data is entered

DATA

		Direction	Amount	Data Type
Data Point no.	1	20	30	POLE TO PLANE
Data Point no.	2	150	45	POLE TO PLANE
Data Point no.	3	210	60	POLE TO PLANE
Data Point no.	4	230	70	LINEATION
Data Point no.	5	30	10	LINEATION

The resulting plot (half normal size) looks like figure 1.

TITLE

Stereonet Plot (Equal Area)

ABSTRACT NUMBER

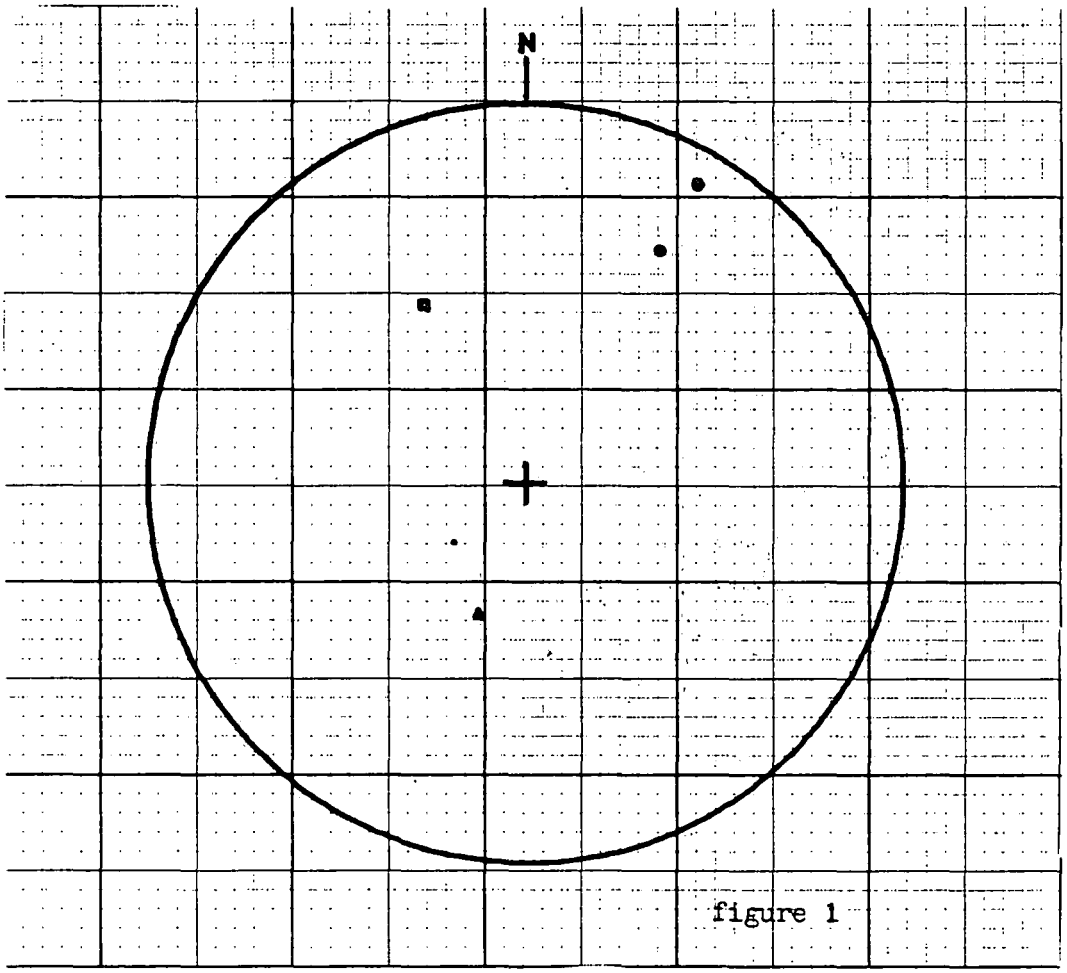


figure 1

Using the data set on page 65 in the rotation program, and using 35° for the top angle and 25° for the side angle the resulting data set would be

DATA

	Direction	Amount	Data Type
Data Point no. 1	62	33	POLE TO PLANE
Data Point no. 2	144	68	POLE TO PLANE
Data Point no. 3	197	65	POLE TO PLANE
Data Point no. 4	270	55	LINEATION
Data Point no. 5	26	7	LINEATION

and the resulting plot would be figure 2.

TITLE

Stereonet Plot (Equal Area)

ABSTRACT NUMBER

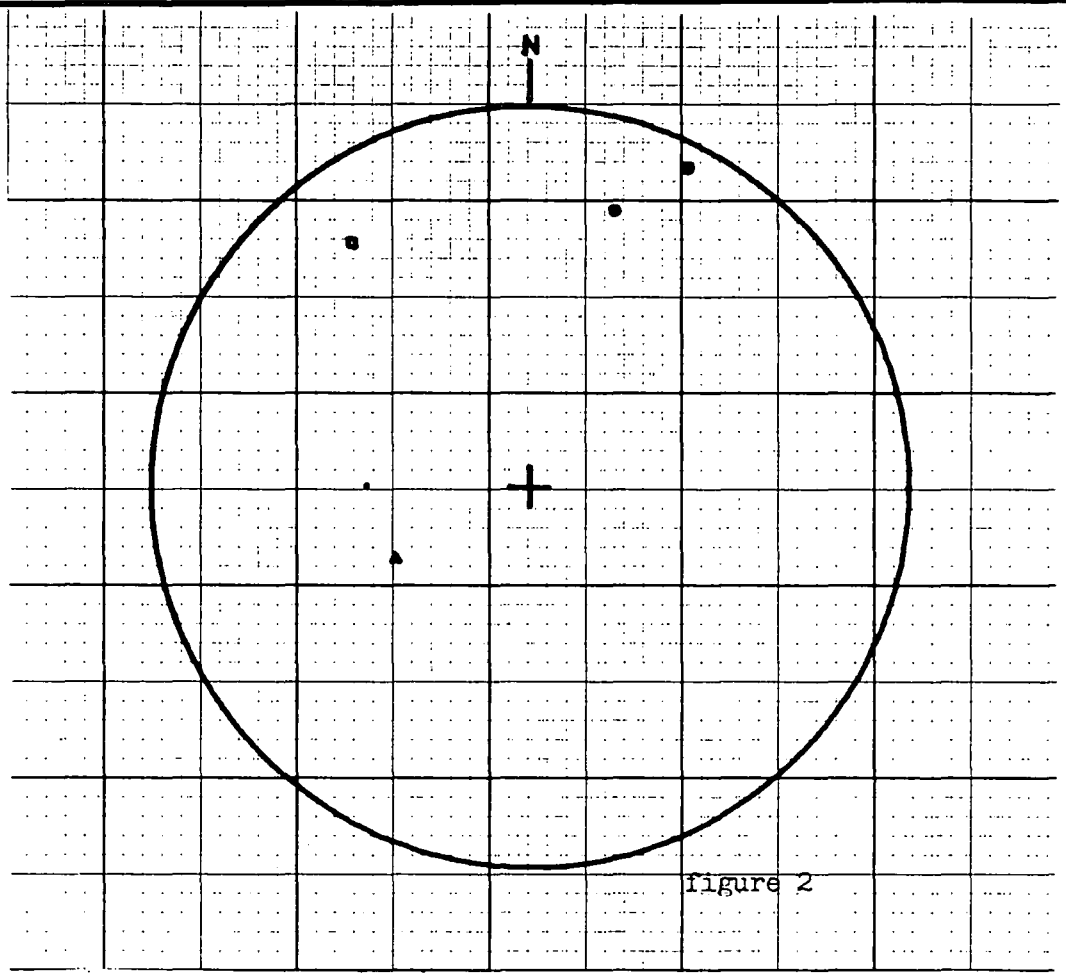


figure 2

Note, the data listings on pages 65 and 67 are examples of the listing option in the data editing program. The printer is conected to the RS-232 interface or the GPIB buss. The address for the printer is 40. The data editing program will ask for the binary data file number and which option. The options are listed on the screen. These options alow the user to delete data, add data, change data, or list the data.

The vector mean / best fit pole program ask for the binary data file number and then lists options. The options are: vector mean, best fit pole,exit, and output data to tape. The options are displayed on the screen. The plotting symbol for the vector mean is a cross (+), and a # number sign for the best fit pole. Figure 3 is a full size plot of pole to plane data (symbol 1) with the vector mean and the best fit pole. Notice that the further the data point is from the cyclographic trace, the less weighted it is in the calculation.

TITLE
Stereonet Plot (Equal Area)

ABSTRACT NUMBER

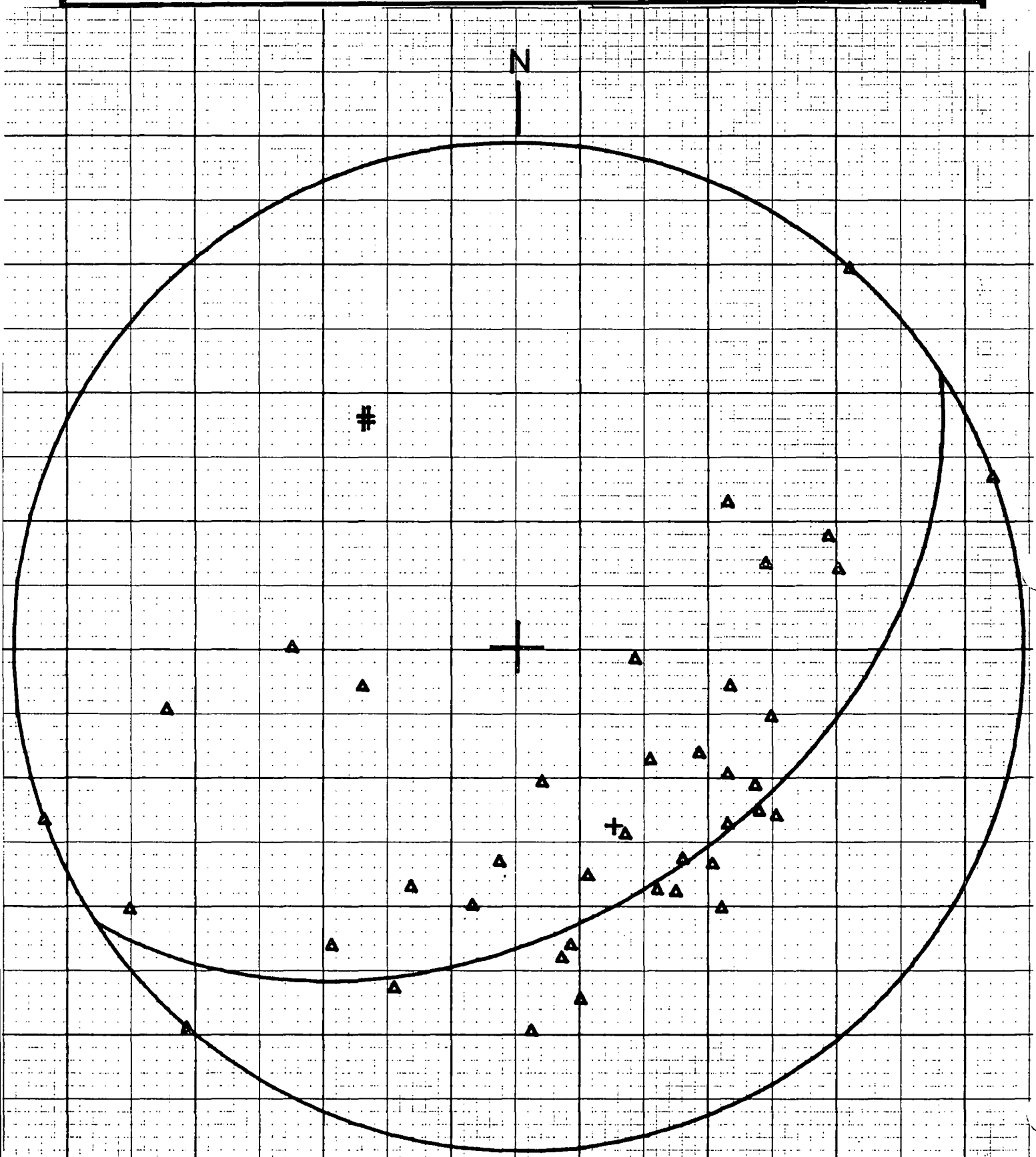


figure 3.

TITLE

Stereonet Plot (Equal Area)

ABSTRACT NUMBER

For report ready stereonets used TEK plotting pens (ink). When making half size stereonets use the same pens as above (not felt tip !!!). For centering paper on the plotter, the center of the stereonet is located at 75,150 (GDU). To make half size stereonets set the upper right corner to 75,150 on the plotter. Note, due to the window dimensions(ie; VIEWPORT) the stereonet appears oval on the screen. It is round on the plotter.

References

Ramsay, J.G. (1967) Folding and Fracturing of Rocks:New York, McGraw-Hill, p. 14-22.

Schmidt, W. (1925) Gefügestatistik: Tschermaks Mineralog. Mitt., v. 38, p. 392-423.

Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
TM5000 Instrument Checkout	062-7456-01 - Program 10
ORIGINAL DATE 10-26-82 / REVISION DATE	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM) 32K Optional-464X Printer TM 5000 Series Instrument
AUTHOR Jim Ormond & Bill Vesser, Tektronix, Inc., Beaverton, OR	

ABSTRACT

Files: 16 ASCII Program
Requires dedicated tape

Statement: 4071

A collection of BASIC programs checks out the TM5000 Series of Programmable instruments for all GPIB functions. It checks for compliance to the reference library of commands and Tek Codes and Formats for GPIB instruments.

The instruments are:

DC5009 Programmable Universal Counter/Timer
DC5010 Programmable Universal Counter/Timer
DM5010 Programmable Digital Multimeter
PS5010 Programmable Triple Power Supply
FG5010 Programmable 20 MHz Function Generator
SI5010 Programmable R.F. Scanner
MI5010 Programmable MultiFunction Interface

This verification software provides a go/no-go testing of the firmware command set used during normal operation of the TM5000 instruments. It also verifies that the proper error codes are returned for specific conditions created by the software.

Operator interaction is minimal. Once a test or test sequence has been selected, no additional settings or display screen pagings are required. The exception is testing the error code generation. In this mode, the operator is required to interact periodically by pressing the front panel ID button on an instrument, connecting a signal to the instrument, etc.

TABLE OF CONTENTS

PRELIMINARY OPERATING INSTRUCTIONS

Using the instructions on page ii, transfer the 16 files comprising this program to a separate tape.

From TEKNIQUES VOL. 7 NO. 4 T2 tape			To TM5000 Instrument Checkout tape		
File #	Bytes	Type	File #	Bytes	Type
24	16128	ASCII Program	1	16128	ASCII Prog
25	4864	" "	2	4864	" "
26	6144	" "	3	6144	" "
27	6656	" "	4	6656	" "
28	6656	" "	5	6656	" "
29	6144	" "	6	6144	" "
30	6400	" "	7	6400	" "
31	8704	" "	8	8704	" "
32	7424	" "	9	7424	" "
33	5120	" "	10	5120	" "
34	5376	" "	11	5376	" "
35	4352	" "	12	4352	" "
36	5632	" "	13	5632	" "
37	3584	" "	14	3584	" "
38	7168	" "	15	7168	" "
39	1792	" "	16	1792	" "

OPERATING INSTRUCTIONS

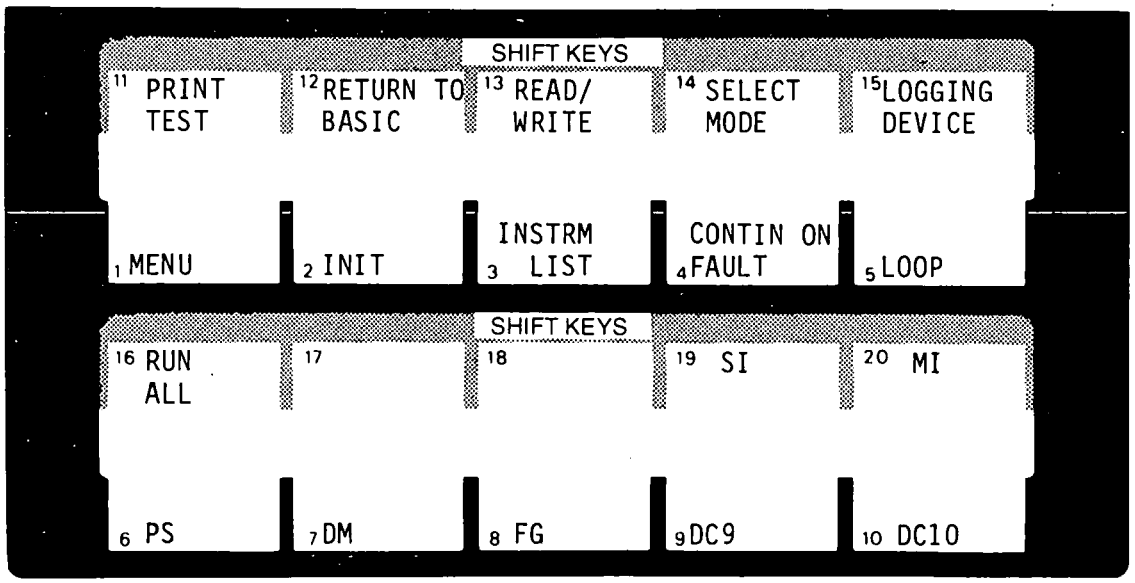
Once the files have been transferred, simply insert the TM5000 Instrument Checkout tape into the 4050 Tape Drive and press AUTOLOAD. The following dialog will appear on the screen.

```
Menu — User Definable Keys:

1..Menu              6...PS5010      11..Print Test Info.
2..ReInitialize      7...DM5010      12..Return to 4050 BASIC
3..Instrument List   8...FG5010      13..Read/Write Immediate
4..Cont. on Fault    9...DC5009      14..Select Test Mode
5..Loop on Test      10..DC5010      15..Select Logging Device
                     19..SI5010      16..Run ALL Tests
                     20..MI5010

Press USER KEY:*
```

This message indicates that the verification software has been loaded. You can now press any of the user-keys listed above to select the desired routines. These routines are explained next.



1.3 Descriptions of Verification Routines

The 4050/TM5000 Verification Software has 18 routines that are callAble from the 4050 user-definable keys. These keys are located to the upper left of the main keyboard.

Most of the keys have two routines associated with them. In the discussion below, the number of the user key which calls that routine appears just ahead of the routine name. To select any test with a key number between 1 and 10, just press that key. If the number of the routine is between 11 and 20, first depress and hold the SHIFT key (on the main keyboard) then press the appropriate user key. For example, pressing key 1 selects the Menu routine, but pressing SHIFT-Key 1 selects The PRINT TEST INFO routine.

The software is designed in modules so that it can either be executed all at once (by pressing the ALL key) or executed in selected segments (by selecting certain other keys such as PS5010 and DC5009). The following paragraphs explain each of the 18 routines in greater detail. The number preceding each routine name refers to the key that must be pressed to execute that particular routine.

1. Menu

The Menu routine prints the complete routine list along with the user key that selects each one. (This is the same menu that is printed when the software is first loaded.) After examining the Menu you can press a user key to execute one of the 18 routines listed.

2. Re-Initialize

The Re-Initialize routine resets all parameters to their start-up values:

A. No print out of test information.
B. Stop on error.
C. Loop on each test once.
D. Primary Address = 22
E. Intrument Type = PS5010
F. No other instruments on the list.
G. Page full = Blinking 'F'.

If the Re-Initialize key is pressed, the above parameters are selected. If other parameters are desired, each one must be selected and set.

3. Instrument List

This routine allows you to specify the addresses of TM5000 instruments to be tested. In addition, you also specify the type of instrument at each address so the program will know which test is valid for that address. Up to 15 instruments may be on the GPIB (IEEE 488 bus) at one time. The instrument to be tested is selected as a part of this routine.

The Instrument List routine begins by listing the current instrument types and their addresses, as shown by the following print out. Notice that the list shows the initial (default) type and address assumed by the software:

```
Changes are made to the Instrument list by entering the
INSTRUMENT TYPE (T) and PRIMARY ADDRESS (P) as follows:

To add an instrument, type:      TəP
To delete an instrument, type:   DəP
To select an instrument, type:   SəP

To make multiple changes with one entry, place a
semi-colon (;) between commands.

The current Instrument List is:

     PS5010 ə 22

The currently selected Intsrument is the PS5010 ə Add. 22

Do you want to change the list ?Y
```

In response to the question, type Y followed by a carriage return. This prints the message:

```
Instrument types available:

     1..PS5010
     2..DM5010
     3..FG5010
     4..DC5009
     5..DC5010
     6..SI5010
     7..MI5010

Enter changes: 3ə24;6ə23;7ə26;Sə24;Də22
```

In response to this statement, enter a new instrument type (1 - 7), the 'ə' symbol, and the primary address of the Intsrument. If you wish to select an instrument already on the list for testing, enter 'S' (for select), the 'ə' symbol, and

the instrument's Primary Address. To delete an instrument from the list, enter 'D' (for delete), the ',', and the Primary Address of the instrument to be deleted. To manipulate more than one instrument list parameter in a single entry, as is shown in the example above, simply use a semi-colon (;) as the command delimiter.

Follow along the example entry above. In this case, the user manipulated 5 parameters in the list. The first three commands added an FG5010 at address 24, an SI5010 at address 23, and an MI5010 at address 26. Next, the instrument at address 24 (The F5010, in this case) is selected for testing. Finally, the instrument at address 22 (the PS5010) is deleted from the list.

After your response has been entered, the screen pages and the new list is printed along with the question:

Do you want to change the list?

If you are satisfied with the selections you have made, type an N followed by RETURN. The Instrument List is now exited as indicated by the prompt:

Press USER KEY:

4. Continue on Fault

This routine causes the checkout software to report any non-fatal faults detected, but directs it to continue testing. Fatal faults are those which preclude further testing (e.g., no response at a selected bus address). Non-fatal faults are those that are specific to a portion of the instrument under test. When a non-fatal fault is detected, an error message is printed and the remainder of the test is completed.

Repeatedly pressing the Continue on Fault key switches the software back and forth between "continue" and "halt" modes. For example, if the previous mode was "continue" and then the Continue on Fault key is pressed, the following statement is printed:

Testing will now - HALT - when an error is detected.

If the previous mode was "halt", then the statement is:

Testing will now - CONTINUE - when an error is detected.

5. Loop on Test

The Loop on Test routine allows you to specify the number of times that each test is to be executed. If Loop on Test is not executed, each test is performed once.

When the Loop on Test routine is executed, it prints:

Enter the number of times to loop on all tests.

Simply enter a number specifying the number of times each test is to be executed and press RETURN.

11. Print Test Information

This routine allows you to print information about the test in progress. Examples of the types of information printed are:

1) Current instrument settings.
2) Responses to queries from the controller.
3) Status information received after a GPIB POLL.

When the Print Test Information routine is executed, it prints:

Information about what data is sent, what data is received, and other pertinent information about the test can be printed on the terminal screen.

Do you want the test information printed?Y

If you want the test information printed on each test, simply type Y and press the RETURN key. Assuming that a Y has been entered, the following print out occurs:

Because of the numerous print outs when the flag is set, you may want to have the screen paged automatically and/or copies made automatically. If so, enter:

0 -- No PAGE, no COPY (blinking F)
2 -- PAGE, no COPY
3-- PAGE and COPY

The first case (0) is the default condition and is selected by entering 0 and pressing RETURN. If you want the screen erased after each full page, but do not want a hard copy, enter 2 followed by RETURN. If you want a hard copy generated after each full page, enter 3 followed by RETURN.

12. Return to 4050 BASIC

The Return to 4050 BASIC routine halts execution of the verification software and returns control to the 4050-series controller. It also sets the TM5000 instruments currently in the Instrument List to their power-on states, sends the GPIB DCL (Device CLear) message, and unconfigures the bus (sends UNT and UNL). However, it cannot be used to recover from a condition in which the program hangs without further execution. If this happens, you can recover by pressing the BREAK key twice in succession. After the Return to 4050 Basic key has been pressed, type RUN to re-execute the software.

13. Read/Write Immediate

Read/Write Immediate is a "Talker/Listener" routine which allows you to send commands to the instrument selected for test by the Instrument List and receive responses from the instrument, when appropriate.

Upon pressing User Key 13, the following appears on the screen:

```
This routine will allow you to enter Instrument Commands
directly from the keyboard.

The instrument to which commands are directed is
selected in the INSTRUMENT LIST.

You will be prompted for input with "??"

To get out of the READ/WRITE IMMEDIATE loop, type "EXIT".

To re-send the last command line you entered, hit RETURN
(with no text) in response to the "??" prompt.

??
```

At this point you may enter any command the instrument will accept. (You could, of course, enter an illegal command and get the response to SRQ). The command string entered is checked for the presence of either '?' or 'SEND' and, if present, input from the instrument is solicited.

If you wish to repeat the last command string entered, simply hit RETURN and the string will be re-sent to the instrument. To get out of the loop, enter 'EXIT' and RETURN. At this point the program will respond with:

```
Press USER KEY:
```

14. Select Test Mode

Selecting this routine allows you to choose between the 'Legal Mode' tests and the 'Illegal Mode' tests. Legal Mode tests are those which are traditional for checkout softwares. They test the firmware command set of an instrument for recognition of commands sent in a 'normal' manner. The test files numbered 2 through 8 of this package are the Legal Mode tests. Illegal Mode tests, on the other hand, are those which send an instrument commands which are specifically wrong in certain ways. The purpose of these tests is to cause the firmware to generate SRQ's and the error codes which indicate what was wrong with the last command string. Test files 9 through 15 of this package are the Illegal Mode tests.

Upon entering the Select Test Mode routine the following is printed:

> You have the option of testing the instruments for proper recognition of "legal" commands (the normal mode of this checkout software), or you may test for proper error code generation.
>
> Enter "0" for normal mode or "7" for Error Code tests:

The "0" and "7" are used by the FIND command in the program to indicate if the program is to FIND files in the 2 - 7 range or the 9 - 15 range. (FIND F+M0, where M0 is what you are setting here).

15. Select Logging Device

The Select Logging Device routine allows you to steer the output generated when you have selected to Print Test Information (Key 11). Terminal dialogue in this routine is as follows:

> You have the choice of logging test names and error Messages to the terminal screen or to a line printer.
>
> Enter the number which represents your choice:
>
> 32...Log data to terminal screen
> XX...Log data to line printer - XX = ROM Pack address.
> (41, 51, etc.)
>
> Enter choice ->

Valid entries are limited to 32, 41, 51, 61, and 71. The value entered is checked against these and the routine is exited if a

match is found.

16. Run all tests

The ALL routine causes all verification tests to be executed for the current Instrument List in the following order:

PS5010
DM5010
FG5010
DC5009
DC5010
SI5010
MI5010

The routine goes down the above list of tests, pointing to each test in turn and checks the Instrument List for any instruments of that type. If a match is found, then that test is brought into the 4050 and executed. Notice that the sequence corresponds to pressing user keys 6 - 10, 19, and 20 in succession.

If the Continue on Fault routine (user key 4) has been executed prior to executing ALL, each test prints out the appropriate comments when an error is detected and then continues with the test.

NOTE

The remainder of the routines execute specific checkout tests. As previously noted, these tests are automatically executed as part of the ALL routine.

6 - 10, 19, 20. Verification tests

Keys 6 through 10, 19, and 20 cause the various test files to be overlayed into 4050 memory and executed. In order to run a specific test, the type of instrument the test is designed to check must be in the Instrument List and selected for testing. Upon pressing any of these keys, the unit selected for test is verified to be of the correct type. If the test you selected does not match the type of instrument currently selected in the Instrument List the following message is displayed:

The <inst type> is not selected for test in the Instrument List

where <inst type> is the name of the type of instrument the test you chose would check out.

1.4 Using the Verification Software

The following is a typical operating sequence using the 4050/TM5000 Verification Software. It is assumed that the TM5000 instruments and the 4050 series controller are connected together as a system.

1. Turn on the 4050-series Controller.

2. Insert the Verification Software tape cartridge into the 4050 and press the AUTOLOAD button. The tape contents will be read into memory and, when finished, the menu of routines is printed on the display screen. A blinking asterisk appears after the "Press USER KEY:" prompt. This indicates that the program is in IDLE or MONITOR mode and waiting for an input. Press a user key to select the desired routine.

NOTE

Due to the nature of 4050-series Controller's SRQ handling, it is recommended that the TM5000 instruments not be turned on until after you have built the Instrument List.

3. Press user key 3 to check the Instrument List. Make changes to the list to reflect the addresses and types of instruments connected to the GPIB.

NOTE

Do not use GPIB Primary Address 0 for any instruments on the bus. This is the address the 4050 assumes for itself.

4. Power up the TM5000 instruments and any associated equipment. Allow an adequate warm-up time (at least 10 minutes).

NOTE

It is recommended, through experience, that you select Read/Write Immediate (key 13) before powering up the TM5000 instruments. This allows a friendlier environment (for you) in which to field the power-up SRQ's. After turning the instruments on, wait a few seconds and enter any command (ID? or ERR? does nicely) to allow the 4050 to handle the SRQ's. Type "EXIT" to get back to IDLE.

5. Select Continue on Fault (key 4), Loop on Test (key 5), Print Test Info (key 11), and/or Select Test Mode (key 14), if desired.

6. To execute the verification tests, run the ALL routine (key 16), or any of the specific tests included in the ALL routine (callable by keys 6 through 10, 19, and 20).

7. To halt execution of the verification software, press the BREAK key and then press user key 12 (Return to 4050 BASIC). This returns the 4050 to the idle mode and resets the TM5000 instruments.

2.1 Mainstream Program

The 4050/TM5000 Verification Software is modeled after the SPS Checkout package.
Some expansions were made in the mainstream to allow for multiple instrument types and addresses and there are some other added features, but the basic structure is maintained.

The form of this discussion will be of a blow-by-blow nature, where needed, divided into sections according to the program structure.

As this discussion will reference specific line numbers from time to time, it is important that you have a program listing along side this document. As of this writing, the current version number and date of the software is V99-95, 8-APR-81.

User Key Definitions

Lines 3 - 82 serve to define the user keys. The variable F0 is given values to represent functions to be performed in the mainstream. Variable F is given values to represent file numbers to be overlayed into 4050 memory when an instrument test is selected. When the user presses a user key, the 4050 vectors to the line number equal to the key's value multiplied by 4 in an implied GOSUB. For example, if the user presses user key 4, the 4050 performs a GOSUB 16.

Wait Loop

Lines 100 - 215 comprise the Wait Loop. This routine starts by setting the function variable, F0, to 0 and performs a subroutine at line 900 to initialize all global variables and a routine at line 2300 which prints the title and menu.

After the two subroutines are finished, the GOSUB statements at lines 125 and 130 are DELETEd to prevent gratuitous re-initialization on later passes through this portion of the program.

Line 145 makes a call to the subroutine at line 180 to link and enable the SRQ handler and print a prompt to the operator on the screen. Upon RETURN, the routine prints a blinking, non-stored asterisk to indicate to the user that the program is waiting for user key input.

When the operator presses a user key, the 4050 performs its implied GOSUB to line number <key value * 4>.

The routines that the Key Handler calls end with GOTO 180. The section of code at line 180 assures linkage with the SRQ handler and prints the prompt. The RETURN statement at line 200 terminates that implied GOSUB caused when the user key was pressed. It returns the program to the 'Blinking Asterisk' loop, above.

Run Program Control

The routine from Lines 300 - 335 is the Run Program Control routine and is entered from the Key Handler. It disables further interrupts from the user keys and performs a computed GOSUB based on the value of function pointer, F0, given by the Key Handler.

Note that the routine terminates with GOTO 180, the reason for which is discussed under "Wait Loop", above.

Overlay Tests

Lines 400 - 490, the Overlay Test Program routine, is also entered from the Key Handler. In this routine, the user keys are, again, disabled and a 'junk string' is loaded with the name of the instrument selected for testing by the Instrument List routine. T$ contains the names of all the instruments this software will test. F was set in the Key Handler to point to the test file.

Next, The element of array T (defined in the Instrument List routine) pointed to by N3 (a pointer to the instrument selected for test) is checked to see if the test desired is for the type of instrument currently selected for testing.

The elements of array T are assigned values in the Instrument List routine. These values signify instrument types by number (1=PS5010, 2=DM5010, etc.). The elements of this array have a one-to-one relationship with the elements of array P, the Primary Address array, also assigned values in the Instrument List routine. Variable N3 is a pointer to the unit under test. This variable also receives its value from the Instrument List routine.

If the test at line 425 determines that the instrument type does match the test selected, then a test is made to see if that file is already in 4050 memory. This is done by comparing the Junk string, which was loaded at line 420, to F$ which contains the name of the last test run. If the test is true, then a branch to the test is made without re-appending the file. Otherwise, the portion of memory reserved for test overlays is cleared and the new test file is searched for. Note that line 455 says FIND F+M0. M0 is an offset that determines whether the test is to be a Legal Mode test or an Illegal Mode test. Its value is determined by the routine at line 850, which is to be discussed later.

Line 460 compresses memory.

The test file is then appended and branched to. N4 is a copy of N3, carried into the test overlay. Note that this routine, like the one at line 300, ends with GOTO 180.

Write to GPIB/Read from GPIB

These two routines are the global GPIB I/O routines. Lines 500 - 530 are the Write routine, and Lines 600 - 630 are the Read routine. These routines are used by the test overlays and the immediate mode I/O routine, discussed later.

F1, tested at lines 515 and 615 is a flag to determine whether to display the messages passing between the 4050 and TM5000 instruments. D0 is given a value in another routine and steers the printing to the terminal screen or another data logging device, such as a line printer.

P(N4) is the primary Address of the currently selected instrument under test. A$ is the global GPIB output string and R$ is the global input from GPIB.

Read/Write Immediate

Lines 700 - 845 make up what manuals folks like to call a Talker/Listener Program.

The routine starts out by saving the state of print flag F1 in X and then re-assings a value of 1 (print all I/O) to F1. Next, a prompt is displayed and input is made to C$.

First, C$ is tested to see if the user wants to leave the routine and, if so, the previous state of F1 is restored and the routine is exited. If not, then C$ is tested to see if it is null. If it is null, then the previously sent command is sent again. This allows one to repeatedly send the same command string without having to re-type it.

If C$ contains something other than 'EXIT' or null, then that new C$ is copied into the global output string, A$, and sent out.

Line 805 tests A$ for the presence of a '?' or the 'SEND' command. Either of these two in a string signifies that a command was sent which warrants an input from the instrument, so a GOSUB to the Read routine is made.

Test Mode

Since the TM5000 Verification Software includes files for both 'Legal' mode and 'Illegal' mode (where error codes are tested), this routine, from Lines 850 - 898, provides for setting variable M0 to offset the tape search by 0 or 7. The Legal mode tests are in files 2 - 8 and the Illegal mode tests are in files 9 - 15.

Note that F$, which contains the name of the last test run, is cleared so the Overlay Test routine won't find an instrument name match in it and ignore appending the new file.

Initialize

This routine, from Lines 900 - 1085, sets all variables to their default states, assures linkage to the SRQ handler, sets the terminal screen page-full parameter to 'Blinking F', and, if this is not the first time through, prints a message on the screen.

Instrument List

The Instrument List starts out by printing some instructions on the screen, then, at line 1180, a test is made of Instrument Type array T. Array T contains numbers in the range 1 - 7 corresponding to the instrument types currently associated with Primary Addresses in the list. If array T is empty, there are no instruments in the list and a branch is made to line 1615 where a message to that effect is printed, after which you are solicited for input.

If array T does contain instrument types (the normal condition), a FOR/NEXT loop is entered. The variable N1 contains the number of instruments on the list. T$, which holds the names of the instruments this software will check out, is segmented out using array T indexed by I. For each element of T, the appropriate instrument name is drawn from T$ and printed along with the Ith element of P, the Primary Address array.

Next, at line 1210, variable N3, a pointer into arrays T and P, is tested for non-zero. N3 points to the unit selected for test and if it is 0, there are no instruments selected so a branch is made to line 1595 where an appropriate message is displayed, after which another branch is made to where you are asked for input.

If, however, N3 is not zero (the normal condition), then the unit selected and its Primary address are displayed by getting the segment of T$ pointed to by T(N3) and printing it with P(N3).

At this point, line 1230 askes if any changes are to be made to the list and a computed branch is made accordingly. If you answered 'N', the program branches to line 1585 where N1 and N3 are tested to see that you have at least one instrument on the list and one is selected for testing. If either variable is 0, you must do something about it and are taken to the section that askes for input.

If, at line 1240, it is determined that you said 'Y', control goes to the routine between lines 1245 and 1275. This routine simply steps through T$ indexed by I and prints a menu of instrument types and their numbers from which to choose your input.

Line 1290 inputs the changes into O$. The syntax of filling O$ is covered in the operator's section of this document. After inputting to O$, a subroutine at line 1625 is called which goes through O$ to determine (1), how many changes are specified in O$ and, (2), at what character positions are the delimiters (";").

The subroutine starts out by initializing two variables, C1 and C2, to 0. C1 is the count of how many changes (commands) are in Q$ and C2 is the count of how many passes have been made through Q$. This routine makes 2 passes.

On the first pass Q$ is stepped through one character at a time. Whenever a semicolon is found, the pass count in C2 is checked and, if C2 shows this to be the first pass, the command count, C1, is incremented. When the end of Q$ is found, C2 is set to 1, indicating that one pass has been made, delimiter pointer array P1 is cleared and diminsioned to the number of delimiters + 1, and the delimiter count, C1, is initialized to 1.

Now the second pass starts. This time, whenever a semicolon is found, delimiter pointer array element P1(C1) receives the value of I, which is the character position of the delimiter, then the delimiter count, C1, is incremented. When the end of Q$ is reached, the pass count is checked and, since this is the second pass, the RETURN statement at line 1670 is executed and control goes to line 1300.

At this point, array P1 contains the character positions of the semicolons in Q$ and C1 holds a count of them.

Now starts the complicated section of the Instrument List routine. Lines 1300 to 1410 are a FOR/NEXT loop in which variable K is stepped from 1 to the number of commands found in Q$. K will be pointing to the elements of array P1. N2 becomes the character position of the '@' symbol 2 or 3 characters before the delimiter pointed to by P1(K). N2 is checked in line 1320 to make sure it is in that range. X$ becomes the character(s) between '@' and ';', then P2 is given the value represented by X$. This is the Primary Address to be operated on and is checked for legal range of value.

Next, X$ becomes the character before '@'. It is tested to see if it is a 'D' (for DELETE), or an 'S' (for SELECT). If it is neither of these, then it must be a number from 1 - 7 indicating the instrument type to ADD to the list with an address of P2. Element N1+1 of array T takes the value of X$ and is checked for correct range of value. The reason the new instrument type is placed in element N1+1 of T is that N1 holds a count of the number of instruments currently assigned in the list and we are adding an instrument one element beyond that count. N1 is checked at line 1365 to be greater than 0. This determines the manner in which P2 is placed into array P. If N1 is 0, P2 is put into element N1+1 of array P and then N1 is incremented to indicate the new count of instruments in the list. Otherwise, if N1>0, the primary address list is stepped

Line 1290 inputs the changes into Q$. The syntax of filling $ is covered in the operator's section of this document. fter inputting to Q$, a subroutine at line 1625 is called hich goes through Q$ to determine (1), how many changes are pecified in Q$ and, (2), at what character positions are the eliniters (";").

The subroutine starts out by initializing two variables, C1 nd C2, to 0. C1 is the count of how many changes (commands) re in Q$ and C2 is the count of how many passes have been made hrough Q$. This routine makes 2 passes.

On the first pass Q$ is stepped through one character at a time. Whenever a semicolon is found, the pass count in C2 is checked and, if C2 shows this to be the first pass, the command count, C1, is incremented. When the end of Q$ is found, C2 is set to 1, indicating that one pass has been made, delimiter pointer array P1 is cleared and diminsioned to the number of delimiters + 1, and the delimiter count, C1, is initialized to 1.

Now the second pass starts. This time, whenever a semicolon is found, delimiter pointer array element P1(C1) receives the value of I, which is the character position of the delimiter, then the delimiter count, C1, is incremented. When the end of Primary Address array is stepped through to find the address to be deleted. If not found, you are told so (sort of), but if P2 is found in P, then a check is made to see if the instrument you are deleting is the one selected for testing. If it is, then N3 is zeroed and dealt with later.

If the unit being deleted is not the one selected for test, then the instrument count is checked to make sure you are not deleting the last instrument on the list (i.e. if you currently have 10 instruments on the list, you are deleting the tenth one rather than the 7th). If it is the last one, then N1 is decremented to show one less instrument and that is that. Otherwise, if the instrument to be deleted is in the middle of the list, N3 must be checked to see if it is pointing at or belo. the instrument to be deleted. If it is, don't bother with it. Next, the instrument count is decremented and the elements of arrays T and P above the deleted instrument are shifted down one notch to fill in the void.

Now for the SELECT routine. It starts off at line 1535 checking the instrument count. If N1=0 nothing is there to select, so a branch is made to print an error message. If N1>1 then array P is searched to make sure the address in P2 exists in P. When The element in P which equals P2 is found, a jump is made to 1560 where the unit selected pointer (N3) is set to point at that element and the SELECT routine is exited.

Return to 4050 BASIC (Done routine)

Lines 1800 - 1895 are the Return to 4050 BASIC routine. This performs some general clean-up before program execution is stopped. It can also be used after hitting the BREAK key to make sure things are in a known state.

First, GPIB commands <DCL>, <UNL>, and <UNT> are sent over the bus. Next, A$ is loaded with the command string to initialize the instrument(s) on the bus (a carry-over from CP56006), and array P is stepped through to send A$ to valid addresses as indicated when any non-zero address is found in the array.

Finally, the 4050 page full parameter is set to 'blinking F', a superfluous SET NOKEY is executed, and the program stops.

Continue on Error

In this routine, which resides at Lines 1900 - 1945, the variable R is set or cleared depending on its state prior to the execution of this routine. Successive executions of Continue on Error, will toggle R to its opposite state. R is referenced in the Error Handler routine to determine whether program execution should halt or continue.

Loop on Test

Loop on Test, from Lines 2000 - 2035, assigns a value to variable L which is used in the test overlays to determine how many times the test is to be performed before returning to the mainstream monitor.

Set Print Flag

Lines 2100 - 2245. Flag F1 is cleared or set to indicate whether or not to print A$ on the logging device before sending it out over the GPIB and B$ upon receipt from the bus. Also, the 4050 page full parameter is chosen.

Print Title and Copyright Notice

This routine, from Lines 2300 - 2390, calls a subroutine at line 2370 to print "-"s across the screen, then the text from 2325 to 2355 is displayed and, again, the subroutine at 2370 is called to print another row of "-"s. A jump is then made to the Menu routine, discussed next.

Print Menu

Lines 2400 - 2495 make up the Menu routine. It begins by restoring the data pointer to line 2450 and goes through a Rather straight-forward READ/PRINT loop which produces a menu printed in three columns. Modifications to this routine call for keeping in mind that you must maintain the 3-column approach in the format of the DATA statements since the READ and PRINT operations work on three data items at a time.

Error Handler

When a test overlay discovers an error, it makes a call to this subroutine between Lines 2500 - 2635 which determines the disposition of program flow.

The variable R, set or cleared in the Continue on Error routine, determines whether the program will return to the test overlay or stop. If R=0 (halt mode), then the dash-printing routine at line 2370 in the Print Title section is called, the error message is printed at device 00 (explained later), another row of dashes is printed, and the program stops.

If R=1 (continue mode), no dashes are printed, but the error message is, and control is returned to the test overlay.

The variables A$, B$, E$, F$, and I$ are carried in from the test overlay, the contents of which are discussed in the error handling portion of the Test Overlays section of this chapter.

Query and Check ID

Lines 2700 - 2765 make up this routine which, when called by the test overlay, sends out the ID? query and checks the first 15 characters of the instrument's response in B$ against the expected response in I$, which was carried into this routine by A$.

A new version of this software could use a variable brought in from th test overlay to be used in line 2735 to replace the "15". This would allow for instruments of various length ID? responses to be checked. The value assigned this variable would be the length of the response less the firmware version number, as the version may not stay the same , but the rest of the ID would.

ALL Tests

The ALL routine, from Lines 2800 - 2890, allows all the instruments on the list to be tested in instrument type order.

First, the number of loops of ALL is assigned to L8 from the keyboard and the loop is started. A message is displayed on device D0 and another loop is started. F2 is the file pointer, the value of which ranges from 2 - 8 or 9 - 15 depending on the value of M0, the 'Test Mode' offset. (M0 will be either 0 or 7, depending on whether you are running Legal or Illegal mode tests).

Line 2825 initializes a flag, F3, which indicates in the next loop whether or not the file of the type of instrument to be tested is already appended. This saves program execution time as well as extra wear on the tape by reducing the need to FIND and APPEND files.

Next, we enter a third loop where a 'Unit Selected' pointer is stepped through the number of instruments in the Instrument List. If the instrument type pointed to by N4 is not the type for the file selected by F2, then the next instrument type is looked at. When the instrument type equals the file type, F3 is checked to see if the file has been appended and, if so, a GOSUB is made to the test. If not, the memory space to be occupied by the test overlay is cleared, a tape search is made for the file, memory is compressed, and the file is appended. Now, F3 is set to indicate file F2+M0 has been appended and a GOSUB is made to the test.

The net result of all this is that the instruments are tested in order of test type by file number rather than serially through the Instrument List in order to reduce tape activity. Note that variable L, which indicates within the test overlays the number of loops they are to perform, is not touched in this routine. Therefore, one could have each test loop a number of times on itself before returning to the ALL routine for the next N4, F2, or L8.

Serial POLL routine

Lines 2900 - 3215 are the Serial POLL routine. The object of the routine is to determine a valid address to POLL, bit-test the resulting status byte to determine its meaning, get the error code from the instrument, and finally display all this in a formatted PRINT statement. This process is repeated for each instrument in the Instrument List.

Starting near the top, at line 2935, S$ is 'cleared' for later use, then a loop is entered in which I7 is the pointer into arrays T and P and is stepped from 1 through the number of instruments on the list. Type array T is used to determine valid elements of the list. If an element of T is 0, there is no valid instrument type at that element. After POLLing address P(I7), the ERR? query is sent to that instrument and its response is recorded in E$.

After receiving the ERR? response, flag F1 is checked to see if the user desires to have program messages displayed. If so, the status byte in S9 is saved in S7, X$ gets the instrument name from T$ and D is checked to see if the unit just POLLed had SRQ asserted. D=1 if it had, so Z$, which is used in the final PRINT statement as a visual indication to the user as to which instruments had SRQ asserted, is made to reflect that.

Next, at line 3000, array S, which was diminsioned to 8 in the Initialize routine, is cleared to all 0's. Now a loop begins where each bit of the status byte, S9, is checked and the elements of S which correspond to bits of S9 that are set (1's) are also set to 1.

When this loop is done, variable S9 will be reduced to 0, so it will be partially built up again in the next loop by checking the lower 4 elements of array S.

After this is done, S(8) (bit 8 of the status byte) is tested to see if the instrument is reporting device-dependant (=1), or system (=0) status. Device-dependant status is that which can only be applicable for a given instrument. An example would be The Phase Lock Interrupt from the FG5010.

System status, on the other hand, is that status which could be reported by any of the instruments, such as reporting 'Command Error', etc.

In the case of this routine, there is no section written to fill S$ with device-dependant messages as this would be too cumbersome with mutiple instrument types, each of which would have to have its own section of device-dependant messages.

Therefore, if S(8)=1, we skip on down and report as much as is known at this point. If S(8)=0, though, then bit 6 is tested to see if the status being reported reflects an Abnormal (=1), or a Normal (=0) condition. Depending on that, the routine either falls through to System Normal Status or jumps to System Abnormal Status.

At this point, in either section, the value of S9 (the lower 4 bits of the status byte) is used in a computed GOTO to assign the proper message to S$.

At line 3195, flag F1 is checked again and, if set, a formatted PRINT is done to device D0. The contents of the PRINT statement are:

Z$="<<SRQ>>" if D=1 after POLL (That device asserted SRQ)
X$="<Instrument name>" of the instrument POLLed
P(17)=<Primary Address> of the instrument POLLed
S7=S9 saved before S9 was torn apart looking for set bits.
S$="<message>" which conveys meaning of lower 4 bits of status
F$="ERR <num>;" ERR? response

Data Logging Select

The purpose of the Data Logging Select routine from Lines 3300 - 3390 is to assign the address of the destination of the PRINT statements throughout the software which output information you selected to print in the Print Test Information routine (user key 11).

If you want to log data to a 4924 tape drive, the test for range of address values at line 3370 will have to include the GPIB address of the 4924. The values which are legal here are the 9050 terminal screen (32) and the ROM pack slots of a 4052 (41, 51, 61, 71). The slots of a 4051 are 41 and 51.

2.2 Test Overlays

The test overlays are appended to the mainstream starting at line 4000 and are accessed by the program via GOSUB 4000 statements. All of test overlays follow the same general formats and conventions, so for this discussion, the FG5010 Legal Mode Settings Test will be used as the example. It resides in file 4 and is very representative of the other tests.

Example Test Overlay - FG5010

At the beginning of the test, F$ is is set to the test name and is use when printing the test title, reporting errors and SRQ's, and is referenced before overlaying tests to determine that the present test is valid for the instrument type to be tested.

Next, the test loop is entered. The variable L, assigned a value in the Loop on Test routine (user key 5), tells how many times to execute this test before returning to the mainstream monitor loop. A message is then displayed which identifies the test, loop count, and Primary Address of the unit under test.

The first thing to verify is that the instrument responds to the ID? query properly, so A$ is loaded with the 1st 15 characters of the expected response and a call is made to the ID Query routine in the mainstream.

Next, the default settings are verified by sending INIT and SET? to the instrument using the global GPIB Put and Get routines. I$ is loaded with the expected reponse to SET?. If the global GPIB input variable, B$, matches I$, then control is passed to the next portion of the test. Otherwise, a subroutine is called which builds an error message and calls the global error reporting routine.

Upon either receiving the correct settings information or return from the error reporting routine, the TEST command is sent to the instrument, time is given for the instrument to complete it, and the results are obtained and verified against the expected response in I$. If the response is not correct, a low-level POLL is performed. The status byte (X) will only be 65 on the first POLL after a true power-up. The test command will not cause it. It is helpful, though, to know what the status byte is.

After TEST comes a verification of all command headers for which the arguments are ON and OFF. The data pointer is RESTOREd to a DATA statement which contains a list of these headers. Variable V, which signifies whether the argument type is Alpha (0), or Numeric (1), is set here to Alpha. In some test overlays there is also a variable O which, when O, means that the command is a Set-only type and cannot be queried. When O=1, that would indicate the command is of a type that can be queried.

Next, at line 4425, A$ is loaded with a couple of commands to set the FG5010 to a state in which all of these headers will be valid. A$ is sent, and a nested loop is entered where each header will be sent and verified with the arguments OFF, ON, OFF. The I loop counts headers and the J loop points to the HEADER OFF or HEADER ON routines.

After this set of commands are verified, 4 more are checked. In line 4415, V is set to Alpha (0) and the data pointer is RESTOREd to the next set of DATA statements, each of which contain a header, the number of arguments for that header, and the arguments. A nested loop is entered at this point. The outer loop reads a header and the number of arguments to be read by the inner loop. In the inner loop, arguments are read and control is passed to a subroutine which puts the headers and arguments together, sends them, and verifies them.

The next section tests a group of 6 headers which take numeric arguments of varying range and step sizes. For this, the DATA statements, starting at line 5970, must contain the command header, starting value, ending value, and step size. These are read into H$, K1, K2, and K3, respectively on each READ. K1, K2, and K3 map directly into the FOR statement in setting up the J loop at line 4640. The J loop's function is to call a subroutine that puts together the header in H$ with the ASCII representation of the value of J, sends it out, and verifies the results. At line 4637, A$ is loaded with the FG5010 DISPLAY command, which, when coupled with the header in H$, tells the FG5010 to display that function on its front panel. At the end of this section, in line 4660, the FG5010 is told to display the current Frequency setting. The reason for this is mainly asthetic - the routine leaves the FG looking 'normal'.

The next section verifies that settings can be stored in and recalled from the storage registers internal to the FG5010.

After starting the I loop, the instrument is INITialized to default settings, then a set of non-default settings is obtained from the DATA statements between 6015 and 6025. A$ is built up with these and sent to the instrument. Next, a SET? query is performed and the results are stored in I$. The reason

for obtaining the 'Expected Results' string in this manner is to make sure the settings in I$ are in the format the instrument will send later. Now, A$ becomes "STORE <I>;INIT;RECALL <I>;SET?;" where I is decremented from 9 down to 1. A$ is sent, B$ is received and compared to I$ The INIT between STORE and RECALL assures that the instrument must actually have STOREd and RECALLed the settings in order to match I$.

The next thing that is checked is the Low Level SETtings feature of the FG5010. It begins by initializing the instrument and sends LLSET? (the Low Level SETtings query). Variables K1 and K2 are deleted, and E is initialized to 0. Now, K1 and K2 become arrays of 50 elements each. 50 is the number of bytes, including header, required to send or receive the binary block.

Continuing on at line 4940, the FG5010 is made a talker. Lines 4941, 4942, and 4947 are remnants of when the byte count in the binary block was dynamic from firmware version to version. Line 4945 reads in 50 bytes from the bus and line 4950 sends UNT and UNL to unconfigure the GPIB.

Next, the data pointer is set to line 6045 where the 50 data items which are the expected response reside. These items are read into K2 a in line 4962, a fast comparison of the two arrays is made. The SUM function is used here under the assumption that if every element of K1 equals every element of K2, the algebraic sums will be equal. It is unlikely that if an element of K1 is off by X that another element of K1 would be off by -X and cancel the error in SUM.

If the sums match, the program skips and jumps on down to check the ERR? query. If not, then an element-by-element comparison is made of the two arrays and an error message is displayed for each mis-match found.

Note the use of variable E in the error message section from 4990 to 5010. It is a flag to print lines 4995 - 5002 only on the 1st error as a heading to the list of bad bytes. The code from 5050 to 5095 is now un-reachable as a result of REMarking out line 4941.

Line 5100 starts the verification of the ERR? query. Since the only predictable error code is ERR 0;, and since there could be some other error code next in line to be sent, the ERR? query is sent twice in order to flush out any pending error code before getting the ERR? response to be verified.

After testing ERR?, the loop count, L1, is updated and the test either runs again or RETURNs to the mainstream.

Now we will look at the various subroutines within the average test overlay.

First, starting at line 5200, there are two routines that take a command header in H$ and append "OFF" or "ON", depending on which routine you are in. The next subroutine appends the argument in S$ to the header in H$ and the fourth subroutine appends the ASCII representation of the value of J to H$. All 4 subroutines pass control to the routine at 5600 which takes A$, appends a semicolon to it, and stores it in I$ to compare with B$ later. Next, the header in H$ is tacked onto the end of A$ and made into a query by the addition of a "?", and then the final semicolon is attached. The final product is in the form of: HEADER ARGUMENT;HEADER?;.

At this point, the next subroutine down the road is called. This one calls the global Put and Get routines and compares I$ to B$ according to argument type. If V is 0, it is an Alpha type argument and can be compared directly. If V=1, though, it is a numeric type argument. This means the charcter position of the argument in B$ must be located and the argument segmented out. X1 points to the space before the argument and X2 points to the semicolon following it. Z$ becomes the header portion of B$ and X$ becomes the argument portion.

Now, the value represented by X$ goes into X and a reverse of the above process is performed: X$ becomes the ASCII representation of X, and B$ is rebuilt, including the final semicolon. The reason for all this is that sometimes there is one space between a header and its numeric argument and other times there are two spaces. This routine assures that, at this point, B$ will always have only one space and can be compared to I$.

Another method of comparing I$ to B$ for numeric argument commands would be to segment out the header (which is what Z$ is at line 5740) and compare that with H$. If they match, then proceed with IF VAL(I$)=VAL(B$) THEN 5785. This is a legal 4050 statement and works quite well. You could shave 3 or 4 lines of code from this section and probably make it an easier reading piece of code as a bonus.

The next routine is used whenever B$ and I$ don't match. It builds an error message in E$ and goes to the Error Handler in the Mainstream.

The last section of each test overlay is the data list. It typically starts out with the headers which take ON and OFF as arguments, usually followed by the headers that take other alpha arguments. Then, the data list covers the headers that take numeric arguments. More often than not, the numeric argument values following the command header will map dirctly into the FOR statement of a FOR/NEXT loop. Finally, for instruments that have a Low Level Settings feature, you will find those data items bringing up the rear.

Examples

```
------------------------------------------------------------------------
                  4050/TM5000 Evaluation Software

                     Version 99-95    8 APR 81

                  Copyright 1980   Tektronix, Inc.

                        ALL RIGHTS RESERVED
------------------------------------------------------------------------


                     Menu -- User Definable Keys:


1..Menu                    6...PS5010          11..Print Test Info.
2..ReInitialize            7...DM5010          12..Return to 4050 BASIC
3..Instrument List         8...FG5010          13..Read/Write Immediate
4..Cont. on Fault          9...DC5009          14..Select Test Mode
5..Loop on Test            10..DC5010          15..Select Logging Device
                           19..SI5010          16..Run ALL Tests
                           20..MI5010

Press USER KEY:



Changes are made to the Instrument List by entering the
INSTRUMENT TYPE (T) and PRIMARY ADDRESS (P) as follows:

To add an instrument, type:      T@P
To delete an instrument, type:   D@P
To select an instrument, type:   S@P

To make multiple changes with one entry, place a
semi-colon (;) between commands.

The current Instrument List is:

     DC5010 @ 20
     DM5010 @ 16
     FG5010 @ 24

The currently selected instrument is the DC5010 @ Add. 20

Do you want to change the list ?N
```

<<FG5010 Error Code Test>> Loop 1 of 1 Unit 24

Please attempt to change settings on front-panel.

Press Front-Panel ID Button.

Connect a cable from the signal OUTPUT (grey outline area) to the TRIG/GATE IN (green outline area) of the FG5010.
Hit RETURN when done.

You may now disconnect the cable from the FG5010.
Hit RETURN when done.

<<DC5010 Error Code Test>> Loop 1 of 1 Unit 20

Connect a fast signal (around 20 MHz will do) to the CHA A input of the DC5010.
Hit RETURN when ready to procede.

Please attempt to change settings on front-panel.

<<DC5009 Error Code Test>> Loop 1 of 1 Unit 18

Connect a fast signal (around 20 MHz will do) to the CHA A input of the DC5009.
Hit RETURN when ready to procede.

Please attempt to change settings on front-panel.
Press Front-Panel ID Button.

Press USER KEY:

NOTE:Only the auto-trigger button met the front panel button criteria.

```
<<MI5010 Settings Test>>  Loop 1 of 1     Unit 23
   Mainframe Test
  ****  ERROR  ****
ERROR - ID incorrect (ID TEK/MI5010,V) recd. During MI5010 Settings Test
          Sent: ID?
      Received: ID TEK/SI5010,V
      Expected: ID TEK/MI5010,V
  ****  ERROR  ****
Incorrect response to INIT;SET? During MI5010 Settings Test
          Sent: INIT;SET?
      Received: ;OPE 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16;CLO 0;ARM OF
F;CONF 4,4,4,4;SCAN 0;OPC OFF;USER OFF;RQS ON;
      Expected: OPC OFF;USER OFF;RQS ON;

Press USER KEY:
```

Example of wrong instrument: SI instead of MI. Failed ID? test and SET? test.

```
<<PS5010 Settings Test>>  Loop 1 of 1     Unit 22

Press USER KEY:
INVALID FUNCTION ARGUMENT IN LINE 37888 - MESSAGE NUMBER 27
LIS 37888

LIS 10000,20000

LIS 3780,3900

LIS 4000,4200
4000 REM **** FILE 2 - PS5010 ****
4005 F$="PS5010 Settings Test"
4010 FOR L1=1 TO L
4015     PRINT @D0:"<<";F$;">>  Loop ";L1;" of ";L;"     Unit ";P(N4)
4100     REM
4105     REM                                                  *** VERIFY ID
         ***
4110     REM
4115     A$="ID TEK/PS5010,V"
4120     GOSUB 2700
4200     REM
```

Ecample of non-existent line number.

```
<<PS5010 Error Code Test>>  Loop 1 of 1     Unit 22
  ****  ERROR  ****
Incorrect Error Code or Status returned after VPOS 20;IPOS .85; During P
S5010 Error Code Test
          Sent: VPOS 20;IPOS .85;
      Received: Status 98, Error Code 205
      Expected: Status 98, Error Code 204
Please attempt to change settings on front-panel.

Press Front-Panel ID Button.

This portion of the test will verify the Supply Regulation
Interrupts and requires you to have a 300 Ohm 1W resistor,
a 10 Ohm 10 Watt resistor, and a large capacitor (around
1400 MFD, 200 VDC) handy.

Please hit RETURN when ready.

Connect the 300 Ohm resistor across the COMMON (white) and
Negative (black) terminals of the Floating Supply.

Hit RETURN when ready.

Move the "hot" end of the resistor from the Negative
(black) terminal to the Positive (red) terminal of the
Floating Supply.

Hit RETURN when ready.

Dis-connect the 300 ohm resistor from the Floating Supply
and connect the 10 Ohm resistor across the terminals
of the Logic Supply.
Hit RETURN when ready.

Dis-connect the resistor from the Logic Supply and set
it aside.  Instal the large capacitor across the Logic
Supply.

                        CAUTION
                        -------
                  MAKE SURE THAT YOU
                OBSERVE PROPER POLARITY
                     OF CAPACITOR.
                  DAMAGE MAY OCCUR IF
                 CONNECTED IMPROPERLY.

Hit RETURN when ready.

Remove capacitor from Logic Supply and connect it to the
Positive Supply, heeding the caution above.
Hit RETURN when ready.

Dis-connect capacitor from Positive Supply and connect it
across the Negative Supply.  Heed above caution.
Hit RETURN when ready.

Dis-connect capacitor from Negative Supply.
Hit RETURN when ready.

Press USER KEY:
```

Tektronix®
COMMITTED TO EXCELLENCE

INFORMATION DISPLAY DIVISION PROGRAM EXCHANGE

TITLE	PART NUMBER
4051 Assembler	062-7456-01 - Program 11
ORIGINAL DATE: October 1983 / REVISION DATE:	EQUIPMENT, OPTIONS AND SOFTWARE REQUIRED (INCLUDING PERIPHERALS AND HOST SYSTEM): 4051 - 16K
AUTHOR: Carl Hovey & Steve Tuttle — Tektronix, Inc. Wilsonville, OR	

ABSTRACT

Files: 1 ASCII Program
1 ASCII Data
3 ASCII Text
(transfer to separate tape)

Statements: N/A (In Call Execute Format)

This program is an interactive assembly and debugging tool for the 4051. IT provides the 4051 programmer with a versatile system for creating and debugging programs in 6800 machine code. The one pass assembler allows the user to examine memory locations and registers.

To create source files the Editor ROM or any other text processing program for the 4051 can be used. Also source can be entered directly into the program.

PRELIMINARY OPERATING INSTRUCTIONS

The five files comprising this program must be transferred to a separate tape.

Using the instructions on page ii, transfer the following:

From TEKNIQUES VOL. 7 NO. 4 T2 tape			TO 4051 Assembler tape		
File #	Bytes	Type	File #	Bytes	Type
40	768	ASCII Program	1	768	ASCII Program
41	11520	ASCII Data	2	11520	ASCII Data
42	4608	ASCII Text	3	4608	ASCII Text
43	3328	ASCII Text	4	3328	ASCII Text
44	768	ASCII Text	5	768	ASCII Text

```
 DDDDDD      DDDDDD      TTTTTTTT
 DDDDDDD     DDDDDDD     TTTTTTTT
 DD    DD    DD    DD       TT
 DD     DD   DD     DD      TT
 DD     DD   DD     DD      TT
 DD     DD   DD     DD      TT
 DD     DD   DD     DD      TT
 DD    DD    DD    DD       TT
 DDDDDDD     DDDDDDD        TT
 DDDDDD      DDDDDD         TT
```

```
VV     VV   555555555    11                  77777777
VV     VV   555555555   111                      77
VV     VV   55           11                     77
 VV   VV    55           11                    77
 VV   VV    5555555      11                   77
 VV   VV    55555555     11                  77
  VV VV             55   11                 77
  VV VV      55     55   11      ....      77
   VVV        5555555    11      ......    77
    V          55555     1111    ....      77
```

DDT V51.7 IS AN INTERACTIVE DISSASSEMBLY AND DEBUGGING TOOL FOR THE 4051. ITS PURPOSE IS TO PROVIDE THE 4051 SYSTEMS PROGRAMMER AND ROMPACK WRITER WITH A VERSATILE SYSTEM FOR CREATING AND DEBUGGING HIS 6800/4051 CODE. DDT V51.7 CONSISTS OF TWO PROGRAMS ON 4051 MAG TAPE

DEBUGGER EXPLAINATION

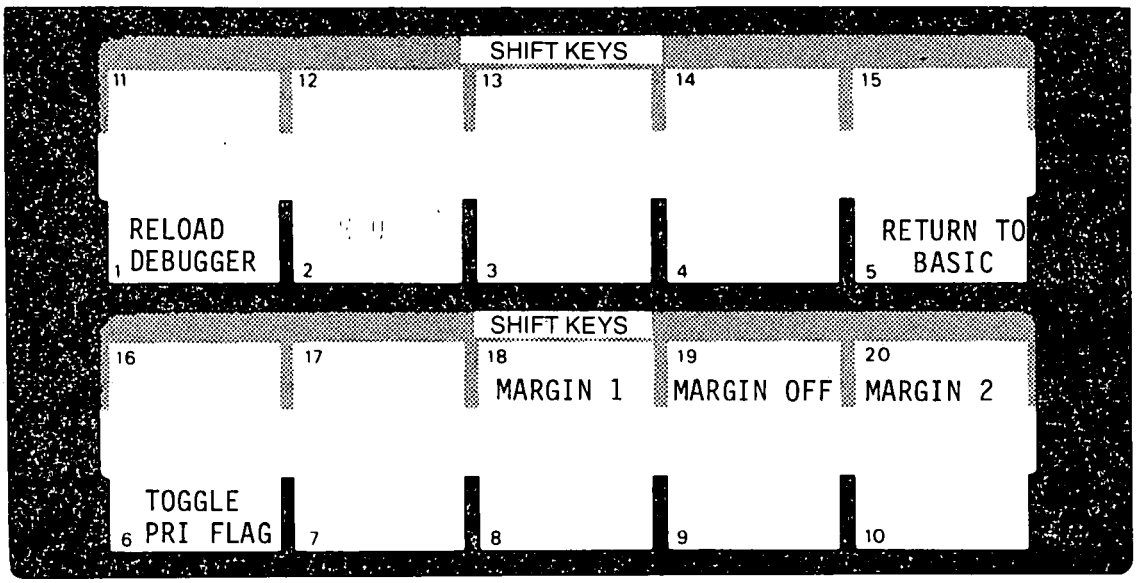
THE 4051 RESIDENT DEBUGGER IS LOADED INTO MEMORY BY A SYSTEM INVOLVING THREE CHARACTER STRINGS. ONE CHARACTER STRING CONTAINS THE DEBUGGER IN A RELOCATABLE ASCII HEX FORMAT. THE SECOND STRING IS A RELOCATING LOADER. THE LOADER IS IN CALL "EXEC" FORMAT. THE THIRD STRING IS THE "DESTINATION" STRING. IN OTHER WORDS THE LOADER LOADS THE DEBUGGER INTO THE THIRD STRING. THE LOADER BUILDS THE THIRD STRING FROM THE FIRST STRING BY TAKING THE ASCII HEX CHARACTERS OF THE FIRST STRING AND BUILDING EIGHT BIT MACHINE CODE BYTES AND STORING THEM IN THE THIRD STRING. CHARACTERS OF A VALID CHARACTER STRING MAY **NOT** HAVE THE EIGHT BIT SET SO THE THIRD CHARACTER STRING OF THIS SETUP IS INVALID. HOWEVER, THERE IS A TRICK. THE THIRD STRING IS DIMENSIONED LARGE ENOUGH TO HOLD THE BINARY VERSION OF THE DEBUGGER BUT WHEN THE LOADER BUILDS THE THIRD STRING IT DOES NOT CHANGE THE LENGTH VALUE FOR THAT STRING. THUS THE 4051 DOES NOT REALIZE THERE IS A STRING WHICH IS FULL OF INVALID CHARACTERS.

THE USER MAY SPECIFY THE ABSOLUTE MEMORY ADDRESS INTO WHICH THE DEBUGGER IS TO BE LOADED BY DEFINING THE THIRD STRING (DESTINATION STRING) TO CONTAIN THE ABSOLUTE HEX ADDRESS FOR THE BEGINNING OF THE DEBUGGER. IF A LOAD ADDRESS IS NOT SPECIFIED BY THE USER, THE DEBUGGER IS LOADED INTO THE SPACE ALLOCATED BY THE 4051 FOR THE THIRD STRING. A USER MUST BE VERY CAREFUL ABOUT WHERE HE LOADS THE DEBUGGER BECAUSE IT IS POSSIBLE TO LOAD IT RIGHT OVER TOP OF PERTINENT DATA ALREADY IN RAM. IF SOME RAM IS CLOBBERED, NOTHING WILL GO WRONG UNTIL THE USER TRIES TO RETURN TO BASIC. ON A RETURN TO BASIC WITH CLOBBERED RAM A SYSTEM ERROR IS ENCOUNTED AND **ALL** DATA IN MEMORY IS WIPED OUT.

IF NO RAM IS CLOBBERED WHEN THE DEBUGGER IS LOADED IT IS **POSSIBLE** TO RUN THE DEBUGGER AND RETURN TO BASIC, RUN BASIC AND THEN RETURN TO THE DEBUGGER ANY NUMBER OF TIMES, WITH NO ILL EFFECTS ON EITHER THE USER OR THE SYSTEM.

A NOTE OF CAUTION: WHEN YOU ARE RUNNING THE DEBUGGER YOU HAVE ACCESS TO THE VERY GUTS OF THE 4051 OPERATING SYSTEM AND IT IS <u>VERY, VERY EASY</u> TO STEP ON SOMETHING AND THEN THE WHOLE SYSTEM <u>WILL CRASH</u>.

TITLE DDT V51.7 TAPE # FILE #



DDT V51.7 USER DEFINABLE KEYS

KEY #1: RELOAD DEBUGGER PROGRAM.
THIS KEY WILL RELOAD THE DEBUGGER PROGRAM'S MACHINE CODE FROM A SOURCE STRING INTO A DESTINATION STRING; CONTROL IS THEN TRANSFERRED TO THE DEBUGGER. THIS KEY FUNCTION ONLY WORKS AFTER THE DEBUGGER'S LOADER PROGRAM HAS DEFINED ALL OF THE NECESSARY STRINGS.

KEY #5: RETURN TO BASIC.
THIS KEY WILL TRANSFER CONTROL BACK TO THE 4051'S BASIC OPERATING SYSTEM. ALL SYSTEM VARIABLES CLOBBERED BY THE DEBUGGER ARE RESTORED. SYSTEM VARIABLES ALTERED BY THE USER MAY RESULT IN A FATAL SYSTEM ERROR. THIS KEY FUNCTION WILL NOT WORK IF CONTROL WAS TRANSFERRED TO THE DEBUGGER PROGRAM VIA A BREAK POINT (USE G COMMAND INSTEAD).

KEY #6: TOGGLE PRINT FLAG.
THIS KEY TOGGLES A FLAG WHICH WILL SUPPRESS THE AUTOMATIC ECHO ON A DEPOSIT INSTRUCTION, DEPOSIT HEX, OR DEPOSIT ASCII COMMAND. MOSTLY USED TO SPEED UP THE ASSEMBLER. ON ENTRY TO THE DEBUGGER, THIS FLAG DEFAULTS TO 'ECHO ON'.

THE FOLLOWING THREE USER DEFINABLE KEYS CONTROL THE CRT DISPLAY'S MARGIN MODES.

KEY #18: MARGIN 1.
THIS KEY WILL FORCE THE DEBUGGER PROGRAM TO BLINK THE "F" AND WAIT FOR THE USER TO PAGE THE SCREEN AFTER ONE COLUMN OF DATA HAS BEEN PRINTED.

KEY #19: MARGIN OFF.
THIS KEY WILL FORCE THE DEBUGGER TO IGNORE ALL PAGE FULL CONDITIONS. AFTER TWO COLUMNS HAVE BEEN PRINTED THE DEBUGGER WILL WRITE OVER COLUMN ONE.

KEY #20: MARGIN 2.
THIS KEY WILL FORCE THE DEBUGGER PROGRAM TO BLINK THE "F" AND WAIT FOR THE USER TO PAGE THE SCREEN AFTER TWO COLUMNS OF DATA HAVE BEEN PRINTED. THE DEBUGGER DEFAULTS TO THIS MODE

NOTE: THE DATA COMMUNICATIONS OVERLAY MAY HELP TO REMEMBER THE KEY NUMBERS

RESIDENT ASSEMBLER

LABELS

ALL LABELS BEGIN WITH THE LETTER "Z".

LABELS MAY END WITH A <CARRIAGE RETURN>, A <PERIOD> OR A <SPACE>. IF THE LABEL ENDS WITH A SPACE, A 6800 OP CODE MNEMONIC, OR AN "*" WITH A COMMENT SHOULD FOLLOW.

EXAMPLE:

```
INPUT NEXT VALID CHARACTER, IGNORE CHARACTERS IN ZTABLE
ZGETCHR                    * EQUATE THE LABEL GETCHR TO THE VALUE $57C9
$6000                      * ORG $6000
ZSTART   LDX    #ZTABLE    * LOAD POINTER TO TABLE
ZANOTHER JSR     ZGETCHR   * REGISTER A GETS NEXT CHARACTER
         JSR     ZSCAN     * SCAN FOR A MATCH
         BEQ     ZANOTHER  * IF MATCH, GET ANOTHER CHARACTER
         RTS
ZSCAN    CMP A   0,X
         BEQ     ZDONE     * IF A MATCHES THEN RETURN WITH EQUAL
         INX
         TST     0,X       * MSB SET MEANS END OF TABLE
         BPL     ZSCAN
ZDONE    RTS
$7000                      * ORG TABLE AT $7000
ZTABLE                     * CHARACTERS TO IGNORE
"$07."$19."$53."$61."$69."$72."$FF.
```

LABELS MAY BE OF ANY LENGTH AND MAY CONTAIN ANY CHARACTER EXCEPT FOR A <PERIOD>, <CARRIAGE RETURN>, OR <SPACE>.

THERE IS A WAY TO CLEAR THE SYMBOL TABLE: RELOADING THE DEBUGGER IS PROBABLY THE EASIEST WAY. THERE ARE NO CHECKS FOR SYMBOL TABLE OVERFLOW.

COMMENTS

COMMENTS MAY APPEAR IN THE SOURCE CODE AS SHOWN IN THE EXAMPLE ABOVE. COMMENTS BEGIN WITH "*" AND END WITH <CARRIAGE-RETURN>.

$HEX

IN ORDER TO EASE THE TRANSFER OF SOURCE CODE FROM THE DEBUGGER UP TO THE CYBER WITH A MINIMUM OF TRANSLATION, THE DEBUGGER WILL ALLOW A "$" BEFORE HEXIDECIMAL NUMERALS.

UNRESOLVED REFERENCES

IF ONE EXECUTES A JSR TO A LABEL THAT IS NEVER DEFINED, THE STACK WILL GROW UNTIL THE SYSTEM BLOWS UP. SO MAKE SURE ALL LABELS ARE DEFINED BEFORE RUNNING A PROGRAM. THIS PROBLEM IS A SIDE EFFECT OF THE WAY THE ONE PASS ASSEMELER HANDLES FOWARD REFERENCES.

"WHAT" ERROR MESSAGES

THE "WHAT" ERROR MESSAGE GIVES THE FOLLOWING INFORMATION:
ADDRESS IN LINE BUFFER OF THE CHARACTER CAUSING THE ERROR
ASCII CHARACTER CAUSING THE ERROR
HEXIDECIMAL EQUIVALENT OF THE ASCII CHARACTER CAUSING THE ERROR

EXAMPLE
0224 V 56 WHAT?

MACROS ON TAPE

ONE MAY SAVE A SEQUENCE OF DEBUGGER COMMANDS ON TAPE. THE SEQUENCE MAY CONTAIN ANY DEBUGGER COMMAND, BUT SOME OF THEM MAY GET YOU INTO TROUBLE (LIKE "J"). THE INTENT IS TO ALLOW THE ASSEMBLY OF A FILE OF SOURCE CODE, OR TO ALLOW ONE TO EXECUTE A COMMON SEQUENCE OF COMMANDS. EXECUTION OF THE MACRO ENDS WITH AN END OF FILE MARK ($FF) AS PLACED ON THE
TAPE BY A CLOSE STATEMENT IN BASIC. IF TWO BACK-TO-BACK CARRIAGE-RETURNS ARE ENCOUNTERED, EXECUTION OF THE MACRO IS SUSPENDED BUT MAY BE RESUMED LATER.

TO EXECUTE THE MACRO, FIND THE FILE (SEE THE "H" COMMAND) AND HIT RIGHT BRACE <SHIFT-]> (THE KEY NEXT TO <BACK SPACE>) AND RETURN.

USING INTERRUPTS
COMMAND SYNTAX NOTATION

COMMAND SYNTAX NOTATION

NOTATION CONVENTIONS USED IN COMMAND SPECIFICATIONS AND EXAMPLES THROUGHOUT THIS MANUAL ARE LISTED BELOW.

NOTATION	DESCRIPTION
BOLDFACE CHARACTERS	BOLDFACE CHARACTERS MUST BE ENTERED EXACTLY AS SHOWN.
ORDINARY CHARACTERS	ORDINARY CHARACTERS (I.E., NOT BOLDFACE) IDENTIFY ELEMENTS THAT MUST BE REPLACED WITH USER-SELECTED VALUES.
$	WHEN NOT USED IN THE CONTEXT OF A DEBUGGER COMMAND, THE DOLLAR SIGN INDICATES A HEXADECIMAL STRING. THE LEGAL DIGITS ARE 0-9 AND A-F.
< >	A WORD OR WORDS ENCLOSED IN THE SYMBOL PAIR <> STANDS FOR AN ENTITY THAT CANNOT BE PRINTED DIRECTLY, OR WHICH IS OTHERWISE REPRESENTABLE ONLY BY A PHRASE. (I.E., THE SYMBOL **<CR>** REPLACES THE PHRASE "CARRIAGE RETURN," AND STANDS FOR THE ASCII CHARACTER WHOSE HEXADECIMAL VALUE IS $0D.)
[	THIS IS HOW A SQUARE BRACKET LOOKS ON THIS PRINTER
[ ]	AN ELEMENT ENCLOSED IN ORDINARY BRACKETS (NOT BOLDFACE) IS OPTIONAL. IF BRACKETS ARE NESTED, ELEMENTS IN INNER LEVELS MAY NOT BE SPECIFIED UNLESS THOSE IN THE OUTER LEVELS ARE ALSO SPECIFIED. FOR EXAMPLE, THE SPECIFICATION [<ITEM #1>[<ITEM #2>], MEANS THAT THE ONLY LEGAL CHOICES ARE: (1) NEITHER ITEM; (2) ITEM #1 ONLY; OR, (3) ITEMS #1 AND #2. ITEM #2 MAY NOT BE SELECTED ALONE.
<CONTROL-I>	THE CHARACTER PRODUCED BY HOLDING THE "CONTROL" KEY DOWN WHILE TYPING AN "I" (ASCII $09).
<CONTROL-B>	THE CHARACTER PRODUCED BY HOLDING THE "CONTROL" KEY DOWN WHILE TYPING A "B" (ASCII $02).
<CONTROL-Y>	THE CHARACTER PRODUCED BY HOLDING THE "CONTROL" KEY DOWN WHILE TYPING A "Y" (ASCII $19).

<CR>	THE CARRIAGE RETURN (ASCII $0D).
<ESC>	THE CHARACTER PRODUCED BY THE "ESCAPE" KEY ON THE TERMINAL (ASCII $1B).
<LF>	THE LINE FEED (ASCII $0A).
<RUBOUT>	THE RUBOUT CHARACTER (ASCII $7F).

ODT V51.7 OPERATIONS

STRINGING COMMANDS

COMMANDS MAY BE STRUNG TOGETHER ON A LINE, BY THEMSELVES OR INSIDE A REPEAT LOOP. ANY COMMAND THAT MAY BE TERMINATED BY A PERIOD (*, ", AND M COMMANDS) MUST BE THUS TERMINATED UNLESS IT IS THE LAST COMMAND ON A LINE.

REGAINING CONTROL

REGAINING CONTROL

ODT V51.7 HAS A COMPACT COMMAND NOTATION, AND WILL INTERPRET WHAT YOU TYPE DISCONCERTINGLY LITERALLY. THUS, IT IS SOMETIMES NECESSARY TO RECOVER FROM PROBLEMS THAT TYPING ERRORS OR HASTY ENTRIES HAVE CAUSED. THERE ARE TWO METHODS FOR ASSERTING CONTROL, LISTED IN ORDER OF ASCENDING POWER AND DESCENDING COUTH.

BREAK KEY

HIT THE GOLD BREAK KEY TWICE AND HOPE YOU GET BACK TO BASIC. ALSO HOPE THAT THE CONTENTS OF MEMORY ARE STILL INTACT.

THE LAST RESORT

TURN OFF THE POWER. (CONDOLENCES)

DEBUGGER INPUT CONTROL

INPUT LINE TERMINATION

<CR>

THE CARRIAGE RETURN (**<CR>**) IS THE UNIVERSAL TRAILING DELIMITER AND LINE TERMINATOR. ANY INPUT STRING, LEGAL OR ILLEGAL, MAY BE TERMINATED BY A **<CR>** (THOUGH NOT NECESSARILY WITHOUT ERROR).

THE **<CR>** IS A LEGAL TERMINATOR FOR THE ' COMMAND (INSTRUCTION ENTRY) AND THE " COMMAND (HEXADECIMAL DIGIT ENTRY). THE **<CR>** WILL CORRECTLY TERMINATE THESE COMMANDS ONLY AT THE POINTS WITHIN THE COMMAND SYNTAX AT WHICH A TERMINATOR IS LEGAL. AN ERROR MESSAGE (DEPENDENT UPON THE COMMAND INVOLVED) WILL BE GENERATED IF THE **<CR>** IS USED IN OTHER PLACES.

<LF>

THE LINE FEED (**<LF>**) MAY **NOT** BE SUBSTITUTED FOR THE CARRIAGE RETURN (**<CR>**) FOR THIS VERSION OF DDT.

CORRECTING TYPING ERRORS

TO CORRECT TYPING ERRORS USE THE 4051 LINE EDITOR KEYS AS DESCRIBED IN THE 4051 MANUAL.

CAUTION: THE LINE EDITOR KEYS CLOBBER THE 4051 PSEUDO REGISTER R0.

ODT V51.7 OPERATIONS
REPEATING DEBUGGER COMMANDS

REPEATING DEBUGGER COMMANDS

MANY TIMES IT IS DESIRABLE TO PERFORM AN OPERATION REPETITIVELY, AS WHEN ONE WISHES TO INITIALIZE MEMORY TO A SPECIFIC VALUE. TWO COMMANDS ARE PROVIDED THAT ALLOW COMMANDS OR COMMAND SEQUENCES TO BE EXECUTED REPETITIVELY.

REPEAT LOOP DEFINITION

[M];N[

THE [COMMAND SETS UP THE PARAMETERS FOR A REPEAT LOOP. THE FORMAT MAY BE ;N[OR M;N[, WHERE N IS THE REPEAT COUNT AND M IS AN OPTIONAL STARTING ADDRESS FOR THE OPERATIONS CONTAINED IN THE REPEAT LOOP. THE MAXIMUM VALUE FOR N IS $FFFF (DECIMAL 65,535). NESTING OF REPEAT LOOPS IS NOT PERMITTED. REPEAT LOOPS ARE TERMINATED BY THE] COMMAND, OR BY THE END OF THE LINE. WHEN THE END OF THE LINE IS THE TERMINATOR FOR A REPEAT LOOP, THE LOOP IS EXECUTED ONLY ONCE. THIS IS NOT CONSIDERED AN ERROR.

REPEAT LOOP TERMINATION

]

THE] COMMAND MARKS THE END OF A REPEAT LOOP. NO PARAMETERS ARE ALLOWED. A [COMMAND NOT TERMINATED BY A] COMMAND IS NOT TREATED AS AN ERROR, BUT EXECUTES ONLY ONCE. A] COMMAND NOT PRECEDED BY A [COMMAND IS IGNORED.

MEMORY DISPLAY, MODIFICATION, AND SEARCH

DISPLAYING MEMORY IN INSTRUCTION FORMAT

[M][;[N]]/

MEMORY DUMP (INVOKED BY / AND IMPLIED BY >, <, AND ^), USES THE CURRENT ADDRESS (CURADR) AS A POINTER TO THE NEXT "INSTRUCTION." IF THE BYTE CAN BE DECODED TO AN INSTRUCTION MNEMONIC, THIS ROUTINE PRINTS THE BYTE(S) IN HEX, THE MNEMONIC AND THE OPERAND. THE LENGTH LOCATION (LEN) IS SET TO THE NUMBER OF BYTES DUMPED.

IF M IS PRESENT, IT IS SUBSTITUTED FOR THE CURRENT ADDRESS BEFORE THE LOCATION IS DUMPED. IF THE SEMICOLON IS PRESENT (OR IF M IS PRESENT) AUTOMATIC ADDITION OF THE LENGTH OF THE PREVIOUS OPERATION IS SUPPRESSED. OTHERWISE, THE LENGTH FROM THE PREVIOUS STEP IS ADDED TO THE CURRENT ADDRESS TO DETERMINE THE LOCATION TO BE DISPLAYED. IF N IS NOT PRESENT, THE ASSUMPTION IS THAT ONE LOCATION IS TO BE DUMPED. IF N IS PRESENT, MORE THAN ONE LOCATION MAY BE DUMPED BY A SINGLE COMMAND. N IS TREATED MODULO 256, AND A VALUE OF ZERO IN THE LOWER 8 BITS CAUSES 256 "INSTRUCTIONS" TO BE DUMPED.

THE DUMP FORMAT IS SIMILAR TO AN ASSEMBLY LISTING. THE GENERAL FORMAT IS:

ADDR/ XX MNE R ARG,X <E.A.>

FIELD	MEANING
ADDR	ADDRESS OF THE FIRST BYTE OF THE "INSTRUCTION"
/	INDICATES THE DUMP WAS CAUSED BY A / COMMAND
XX	FIRST BYTE OF THE "INSTRUCTION"
MNE	THE INSTRUCTION MNEMONIC. PRESENT ONLY IF THE FIRST BYTE IS A LEGAL MC6800 INSTRUCTION OP CODE.
R	REGISTER DESIGNATOR. PRESENT ONLY FOR THOSE INSTRUCTIONS THAT REQUIRE IT.
ARG	ARGUMENT FIELD ENTRY. PRESENT ONLY FOR THOSE INSTRUCTIONS THAT REQUIRE AN ARGUMENT. THE ARGUMENT ENTRY MAY BE IMMEDIATE (SIGNALED BY A "#" PRECEEDING THE NUMBER), OR IT MAY BE AN ADDRESS. IN EITHER CASE, THE NUMBER MAY BE ONE OR TWO BYTES LONG.
,X	INDEXED OPERATION INDICATOR. PRESENT ONLY FOR INDEXED INSTRUCTIONS.
<E.A.>	EFFECTIVE ADDRESS. PRESENT FOR INDEXED AND BRANCH INSTRUCTIONS ONLY.

DISPLAYING THE REFERENCED ADDRESS

[M]>

THE '>' COMMAND ALLOWS ONE TO EXAMINE A LOCATION REFERENCED BY THE PREVIOUS INSTRUCTION EXAMINED. ONE MAY THEN RETURN TO THE ORIGINAL INSTRUCTION WITH THE < COMMAND. ALL THIS HOPPING AROUND IN MEMORY MAY BE DONE WITHOUT HAVING TO REMEMBER OF RETYPE ANY OF THE ADDRESSES AS THEY ARE SAVED FOR YOU BY THE DEBUGGER. ISN'T THAT NICE? IF M IS PRESENT, IT IS SUBSTITUTED FOR THE CURRENT ADDRESS BEFORE THE LOCATION IS DUMPED.

THE FORMAT OF THE DUMP IS SIMILAR TO THAT OF THE / COMMAND.

FORWARD STEPS ARE SAVED IN THE DEBUGGER'S STACK FOR LATER DISPLAY BY THE < COMMAND. BE CAREFUL NOT TO OVERFLOW THE STACK.

BACKTRACKING

<

THE < COMMAND BACKS UP TO THE INSTRUCTION EXAMINED BEFORE THE LAST > COMMAND. IF THERE ARE NO PREVIOUS REFERENCES TO BACK UP TO, THE ERROR MESSAGE < **WHAT?** IS OUTPUT.

DISPLAYING THE PRECEEDING ADDRESS

[M]^

THE ^ ("CARET" OR "UP ARROW") COMMAND EXAMINES THE CONTENTS OF THE PREVIOUS LOCATION. THE ^ COMMAND STEPS BACKWARD ONE BYTE AND DISPLAYS THE LOCATION. IF M IS PRESENT, IT IS SUBSTITUTED FOR THE CURRENT ADDRESS BEFORE THE LOCATION IS DUMPED.

MEMORY DUMP IN HEX AND ASCII (QUICK DUMP)

[M][;[N]]Q

THE Q COMMAND DUMPS MEMORY IN BOTH HEX AND ASCII. THE COMMAND FORMAT IS M;NQ, WHERE M AND N ARE OPTIONAL, AND THE SEMICOLON IS NECESSARY ONLY IF N IS SPECIFIED. IF M IS PRESENT, IT IS SUBSTITUTED FOR THE CURRENT ADDRESS BEFORE THE LOCATION IS DUMPED.

THE DUMP BEGINS AT THE CURRENT ADDRESS AND INCREMENTS THE CURRENT ADDRESS AS IT GOES, LEAVING IT SET ONE BYTE BEYOND THE LAST LOCATION DUMPED. THE NUMBER OF LINES TO DUMP IS SPECIFIED BY N (EACH LINE CONSISTS OF 16 BYTES). IF N IS NOT PRESENT, IT IS ASSUMED EQUAL TO 1. N IS TREATED MODULO 256, AND A ZERO IN THE LOW BYTE WILL CAUSE 256 LINES TO BE DUMPED.

THE FORMAT OF EACH LINE OF THE DUMP IS THE ADDRESS OF THE FIRST BYTE ON THE LINE (4 HEX DIGITS), FOLLOWED BY 8 GROUPS OF TWO BYTES (2 DIGITS PER BYTE, 4 DIGITS PER GROUP), FOLLOWED BY THE ASCII TRANSLATIONS OF THE BYTES. THE MOST SIGNIFICANT BIT IS IGNORED AND NO CHARACTERS ARE CONVERTED TO A PRINTABLE CHARACTER WITH AN UNDERSCORE.

PRINTING A FLOATING POINT VALUE

[M][;N] U

THE U COMMAND CONVERTS N 4051 FLOATING POINT NUMBERS FROM THE INTERNAL BINARY FORMAT TO ASCII DECIMAL FORMAT AND PRINTS THE VALUES. M POINTS TO THE FIRST BYTE OF AN 8 BYTE FLOATING POINT NUMBER.

ENTERING MC6800 INSTRUCTIONS

[M][;]'<INSTRUCTION>

THE ' COMMAND ALLOWS THE ENTRY OF INSTRUCTIONS IN MNEMONIC FORM. A <SPACE> MAY BE SUBSTITUTED FOR THE "'" IN ORDER TO MAKE THE INPUT LOOK MORE LIKE THE CYBER ASSEMBLER FORMAT. THERE ARE CERTAIN EXCEPTIONS TO THE ASSEMBLER INPUT FORMAT, THE MOST IMPORTANT BEING THAT ALL NUMBERS ARE ASSUMED TO BE HEXADECIMAL. A "$" IS PERMITTED BEFORE NUMBERS, BUT IS NOT REQUIRED. ALL LABELS MUST BEGIN WITH THE LETTER "Z". EACH INSTRUCTION MAY BE TERMINATED BY A <CR>, A SPACE (OR SPACES) WITH AN "*" FOLLOWED BY A COMMENT, OR A PERIOD (.). WHEN TERMINATED BY A PERIOD, ADDITIONAL DEBUGGER COMMANDS (INCLUDING INSTRUCTION ENTRIES) MAY BE PLACED ON THE SAME LINE.

THE SEMICOLON HAS NO EFFECT IF ENTERED WITH A VALUE FOR M. WHEN AN INSTRUCTION IS ENTERED WITH ONLY A SEMICOLON PRECEEDING THE ', AUTOMATIC ADDITION OF THE LENGTH OF THE PREVIOUS OPERATION IS SUPPRESSED. THIS ALLOWS (FOR EXAMPLE) AN INSTRUCTION TO BE DUMPED BY THE / COMMAND, AND THEN REPLACED BY ANOTHER INSTRUCTION WITHOUT THE NECESSITY OF TYPING THE ADDRESS AGAIN.

ENTERING HEXADECIMAL STRINGS

[M][;]"[XX]...

THE " COMMAND ALLOWS THE ENTRY OF UP TO THREE BYTES OF HEXADECIMAL DATA INTO MEMORY. M AND THE SEMICOLON ARE OPTIONAL AND THE XX ARE HEXADECIMAL DIGIT PAIRS (UP TO THREE PAIRS, AS INDICATED BY THE ...), WHICH MAY NOT BE SEPARATED BY SPACES. THE DIGITS OF A HEXADECIMAL DIGIT PAIR, WHICH CONSTITUTES ONE BYTE TO BE STORED INTO MEMORY, MAY NOT BE SEPARATED BY SPACES. A HEXADECIMAL STRING MAY BE TERMINATED BY A <CR>, OR A PERIOD (.). WHEN TERMINATED BY A PERIOD, ADDITIONAL DEBUGGER COMMANDS (INCLUDING HEXADECIMAL STRINGS) MAY BE PLACED ON THE SAME LINE.

IF M IS PRESENT, THE CURRENT ADDRESS IS REPLACED BY THE VALUE OF M BEFORE ANY BYTES ARE STORED. IF THE SEMICOLON IS PRESENT (OR IF M IS PRESENT) THE LENGTH FROM THE PREVIOUS OPERATION IS IGNORED. IF NEITHER M NOR THE SEMICOLON ARE PRESENT, THE LENGTH FROM THE PREVIOUS OPERATION IS ADDED TO THE CURRENT ADDRESS BEFORE ANY BYTES ARE STORED.

AFTERPWHEN A TERMINATOR IS ENCOUNTERED AFTER ONE OR MORE BYTES HAVE BEEN STORED, THE ROUTINE EXITS WITH THE CURRENT ADDRESS SET TO THE ADDRESS OF THE FIRST BYTE STORED, AND THE LENGTH SET TO THE NUMBER OF BYTES STORED.

ENTERING ASCII STRINGS INTO MEMORY

[M]<AMPERSAND>[<ASCII CHARACTERS>]

THE <AMPERSAND> TELLS THE DEBUGGER TO ENTER ASCII CHARACTERS INTO MEMORY. THIS COMMAND TERMINATES UPON A <CR>. THE <CR> IS NOT ENTERED INTO MEMORY; TO ENTER A <CR> INTO MEMORY, DEPOSITE AN HEXADECIMAL 0D.

ODT V51.7 OPERATIONS
MEMORY DISPLAY, MODIFICATION, AND SEARCH

MOVING BLOCKS OF MEMORY

S;EMD

THE M COMMAND ALLOWS BLOCKS OF DATA TO BE MOVED ABOUT IN MEMORY. THE S PARAMETER IS THE STARTING ADDRESS OF THE BLOCK TO BE MOVED, E IS THE END OF THE BLOCK TO BE MOVED (INCLUSIVE), AND D IS THE DESTINATION ADDRESS. ALL THREE ADDRESSES MUST BE PRESENT. EACH M COMMAND MAY BE TERMINATED BY A <CR>, OR A PERIOD (.). WHEN TERMINATED BY A PERIOD, ADDITIONAL DEBUGGER COMMANDS (INCLUDING ADDITIONAL M COMMANDS) MAY BE PLACED ON THE SAME LINE.

IF ANY OF THE ADDRESSES ARE MISSING, THE START ADDRESS FOLLOWS THE END ADDRESS, OR THE DESTINATION BLOCK WOULD WRAP OVER THE END OF MEMORY, NO BYTES ARE MOVED AND AN ERROR MESSAGE IS PRINTED.

A NOTE OF CAUTION IS WORTHWHILE AT THIS POINT. IF AN NMI INTERRUPT OCCURS WHILE THE M COMMAND IS EXECUTING, THE RESULTS WILL BE DISASTROUS. THE STACK POINTER IS USED AS ONE OF THE INDICES DURING EXECUTION OF THIS COMMAND, SO AN INTERRUPT WILL LIKELY AS NOT DESTROY THE DATA THAT IS TO BE MOVED.

SEARCHING MEMORY

(M);NL

THE L (LOOK) COMMAND SEARCHES MEMORY FROM ONE ADDRESS TO ANOTHER (HIGHER) ADDRESS FOR A SPECIFIED BIT PATTERN (IN THE "ONES" REGISTER) UNDER A MASK (IN THE "MASK" REGISTER). THE LENGTH OF THE SEARCH OBJECT MAY BE ONE, TWO, OR THREE BYTES (IMPLIED BY THE CONTENTS OF "MASK"). M IS OPTIONAL AND WILL DEFAULT TO THE CURRENT ADDRESS. M AND N ARE THE STARTING AND ENDING ADDRESSES, RESPECTIVELY. IF A MATCH IS FOUND, THE ADDRESS OF THE THREE-BYTE BLOCK THAT INCLUDES THE MATCHING LOCATIONS IS PRINTED IN THE FORMAT F XXXX (WHERE XXXX IS THE HEXADECIMAL ADDRESS). IF NO MATCH IS FOUND, THE DEBUGGER PROMPTS FOR THE NEXT COMMAND. ADDITIONAL L COMMANDS WITH NO ARGUMENTS WILL SEARCH FURTHER.

SEE REGISTER CHANGE OPERATIONS FOR SETTING THE "ONES" AND "MASK" REGISTERS.

REGISTER DISPLAY AND MODIFICATION

DISPLAYING ALL PROCESSOR REGISTERS

P

THE P COMMAND PRINTS THE CONTENTS OF THE USER'S REGISTERS IN THE FOLLOWING ORDER:

REGISTER	MNEMONIC
CONDITION CODES	C
ACCUMULATOR B	B
ACCUMULATOR A	A
INDEX REGISTER	X
PROGRAM COUNTER	P
STACK POINTER	S

NO PARAMETERS ARE LEGAL FOR THE "P" COMMAND.

SETTING REGISTERS

M=R
;(N)=R (ALTERNATE FORM)

THE = COMMAND CHANGES THE CONTENTS OF ANY ONE OF THE USER'S REGISTERS OR THE DEBUGGER REGISTERS MASK AND ONES. N IS OPTIONAL AND ASSUMED EQUAL TO ZERO IF NOT PRESENT, AND R IS A MNEMONIC WHICH DESIGNATES THE REGISTER TO BE SET. THE MNEMONICS ARE:

REGISTER	MNEMONIC
ACCUMULATOR A	A
ACCUMULATOR B	B
CONDITION CODES	C
DEBUGGER'S MASK REGISTER	M
DEBUGGER'S ONES REGISTER	O
PROGRAM COUNTER	P
STACK POINTER	S
INDEX REGISTER	X

DISPLAYING SINGLE REGISTERS

RR

AN "R" FOLLOWED BY A REGISTER MNEMONIC WILL DISPLAY THE CONTENTS OF THAT REGISTER.

EXECUTING SUBROUTINES

[M] J

THE J COMMAND TRANSFERS CONTROL FROM THE DEBUGGER TO THE SUBROUTINE STARTING AT M, OR IF M IS NOT SPECIFIED, THE CONTENTS OF REGISTER P IS USED. THE SUBROUTINE MAY RETURN CONTROL TO THE DUBUGGER WITH AN RTS. BREAKPOINTS ARE NOT ENABLED.

UPON RETURN TO THE DEBUGGER, ALL THE 6800'S REGISTERS EXCEPT FOR THE PROGRAM COUNTER ARE SAVED FOR EXAMINATION. (SEE DISPLAYING REGISTERS.)

TO ALLOW THE BREAK KEY TO WORK, THE J COMMAND CLEARS THE USER INTERRUPT MASK BEFORE PASSING CONTROL TO THE SUBROUTINE. THUS IF THE SUBROUTINE HANGS, A BREAK-BREAK MIGHT GET YOU BACK TO BASIC.

MAG TAPE OPERATIONS

THE 4051 MAG TAPE UNIT MAY BE USED BY THE RESIDENT DEBUGGER. FILES MAY BE FOUND, READ AND WRITTEN. FILES MUST BE PRE-MARKED BY BASIC. NOTE: FILES WRITTEN BY THE DEBUGGER MIGHT NOT BE READ BY THE 4051 BASIC O.S. UNLESS GREAT CARE IS TAKEN TO FOLLOW BASIC'S FILE STRUCTURE, ESPECIALLY WITH REGARD TO THE BLOCK LENGTH. AND FILES WRITTEN BY BASIC MAY BE READ BY THE DEBUGGER ONLY ONE RECORD PER T COMMAND.

THERE ARE FIVE DEBUGGER MAG TAPE COMMANDS.

H FIND HEADER OF TAPE FILE

W WRITE DATA ON TAPE

T READ DATA FROM TAPE

- BACK UP ONE RECORD (256 BYTES)

N BACK UP TO BEGINNING OF FILE, TO ALLOW REWRITING HEADERS.

THE SYNTAX OF EACH COMMAND IS AS FOLLOWS:

1. NH FIND THE NTH FILE ON THE TAPE AND READ THE HEADER INTO THE STANDARD BASIC TAPE BUFFER. THE N H COMMAND IS EQUIVALENT TO THE FIND N STATEMENT IN BASIC, EXCEPT THAT N IS HEXIDECIMAL. BE CAREFUL.

2. XXXX;XXXXW XXXX REPRESENT A HEX NUMBER. THE FIRST HEX NUMBER IS THE MEMORY ADDRESS AT WHICH TO START READING DATA AND THE SECOND HEX NUMBER IS THE MEMORY ADDRESS AT WHICH TO STOP READING DATA. IN OTHER WORDS, DATA IN AN AREA OF 4051 RAM DELIMITED BY THE TWO HEX ADDRESSES IS WRITTEN AS ONE CONTIGUOUS DATA BLOCK ON TO THE 4051 MAG TAPE. THE TAPE MUST BE PRE MARKED BY BASIC.

IF XXXX;XXXX IS NOT SPECIFIED, THE STANDARD TAPE BUFFER FROM $11F TO $21E IS USED.

3. XXXX;XXXXT

DATA IS READ FROM MAG TAPE AND LOADED INTO 4051 RAM STARTING AT THE HEX ADDRESS SPECIFIED IN THE COMMAND. NOTE: THE TAPE MUST BE POSITIONED AT THE BEGINNING OF A VALID DATA BLOCK. THE SIZE OF THE DATA (THE DIFFERENCE BETWEEN STARTING ADDRESS AND ENDING ADDRESS) SHOULD MATCH THAT WHICH WAS WRITTEN.

IF THE XXXX:XXXX IS NOT SPECIFIED, THE STANDARD TAPE BUFFER FROM $11F TO $21E IS USED.

HOST LINK OPERATIONS

HOST LINK OPERATIONS

HOST LINK/DOWNLOADER

K

THE K COMMAND ALLOWS THE USER TO LINK TO A HOST TIME-SHARING SYSTEM AND TO DOWNLOAD ASCII HEX INTO THE 4051'S MEMORY. THE FORMAT OF THE COMMAND IS K. NO PARAMETERS ARE VALID.

BEFORE USING THE K LINK IN THE DEBUGGER THE 4051 ENVIRONMENTAL PARAMETER "RATE" MUST BE SET. "RATE" IS THE BAUD RATE AT WHICH YOU WISH TO COMMUNICATE WITH A HOST COMPUTER.FOR EXAMPLE:

```
CALL "RATE",1200,0,2
```

REFER TO THE 4051 OPT 1 MANUAL FOR FURTHER DETAILS. ONCE THE RATE IS SET, CONTROL MAY BE PASSED TO THE DEBUGGER AND THE K LINK MAY BE USED. THE RATE PARAMETER MUST BE SET AFTER A SYSTEM RESET OR POWER UP. TO ENTER THE K LINK FROM THE DEBUGGER TYPE <K> FOLLOWED BY A CARRIAGE RETURN. IT IS NOW POSSIBLE TO COMMUNICATE WITH AND DOWNLOAD FROM THE HOST IN THE STANDARD FASHION. TO EXIT THE K LINK HIT USER KEY #5. THIS GETS YOU BACK INTO THE DEBUGGER COMMAND MODE.

COMMAND SUMMARY

REGAINING CONTROL

USER KEY #5	RETURN TO BASIC

DEBUGGER INPUT CONTROL

<CR>	TERMINATE INPUT LINE
<RUBOUT>	DELETE LAST CHARACTER OF CURRENT INPUT LINE

(SEE 4051 MANUAL FOR A DESCRIPTION OF THE LINE EDITOR KEYS)

REPEATING DEBUGGER COMMANDS

[M];N[	REPEAT LOOP DEFINITION
]	MARK THE END OF A REPEAT LOOP

MEMORY DISPLAY, MODIFICATION, AND SEARCH

[M][;[N]]/	DISPLAY MEMORY IN INSTRUCTION FORMAT
[M]>	DISPLAY REFERENCED ADDRESS IN INSTRUCTION FORMAT
<	BACKTRACK FROM A PREVIOUS > COMMAND, DISPLAYING THE REFERENCING INSTRUCTION IN INSTRUCTION FORMAT
[M]^	DISPLAY PRECEDING ADDRESS IN INSTRUCTION FORMAT
[M][;[N]]Q	DISPLAY MEMORY IN HEXADECIMAL FORMAT WITH ASCII TRANSLATION (QUICK DUMP)
[M][;N]U	CONVERT MEMORY FROM FLOATING POINT TO ASCII

[M][;]°<INSTRUCTION>	ENTER INSTRUCTIONS INTO MEMORY
[M][;]"[XX]...	ENTER HEXADECIMAL DIGITS INTO MEMORY
<AMPERSAND>[<ASCII>]	ENTER ASCII CHARACTERS INTO MEMORY
S;EMD	MOVE BLOCKS OF MEMORY
[[M];N]L	SEARCH MEMORY FOR THE STRING SPECIFIED IN THE O REGISTER, UNDER THE MASK IN THE M REGISTER

REGISTER DISPLAY AND MODIFICATION

RR	DISPLAY SINGLE REGISTER
P	DISPLAY USER REGISTERS
;[N]=R	SET REGISTER

HOST LINK OPERATIONS

K	ENTER KRONOS LINK/DOWNLOADER

MAG TAPE OPERATIONS

NH	FIND FILE N
[XXXX;XXXX]W	WRITE DATA ONTO TAPE
[XXXX;XXXX]T	READ DATA FROM TAPE
-	BACK UP ONE RECORD
N	BACK UP TO BEGINNING OF FILE

This document presents an overview of the I/O System. It also contains a detailed description of the commonly used I/O System variables.

THE 4051 I/O SYSTEM WAS DESIGNED TO BE AS MODULAR AS POSSIBLE TO PROVIDE FOR AS MUCH DEVICE INDEPENDENCE AS RESONABLE.

THE FOLLOWING DISCRIBES THOSE I/O VARIABLES REFERED TO AS THE NORMAL I/O SYSTEM VARIABLES IN THE ROUTINE DOCUMENTATION. THESE VARIABLES MUST BE SET UP BEFORE CALLING MANY OF THE I/O ROUTINES. SOME ROUTINES DO EXIST FOR MANAGEMENT OF MANY OF THESE VARIABLES, THESE ROUTINES INCLUDE ADRDEV, BFRALC, UNADR.

NOTE: ^HNN IS THE NOTATION USED TO REPRESENT NN AS A HEX (BASE 16) NUMBER.

```
************** IOFUNC          : PRESENT I/O OPERATION (BASIC KEYWORD)
************** A.STAT          : CHANNEL A STATUS
;       STATUS BYTE FORMAT
DIRCT  = ^H80                   ; DIRECTION 0-OUTPUT 1-INPUT
BFRSTT = ^H20                   : BUFFER STATUS 0-EMPTY 1-FULL
BUSACT = ^H10                   : ACTIVE BUS 0-NO 1-YES
FMTVLD = ^H08                   : SET IF USING STATEMENT APPLIES
ATVLD  = ^H04                   ; SET IF <ATSIGN> OR <PERCENTSIGN>
                                : INFORMATION IS ON STACK
SECDEF = ^H02                   : SECONDARY DEFINED 0-NO 1-YES
PRIDEF = ^H01                   : PRIMARY DEFINED 0-NO 1-YES
;
************** A.PRIM          : PRESENT PRIMARY ADDRESS
************** A.SEC           : PRESENT SECONDARY ADDRES
************** A.STRT          : POINTER TO FIRST VALID
                                : CHARACTER POSITION IN THE BUFFER.
************** A.MAX           ; POINTER TO LAST VALID CHARACTER
                                : POSTITION IN THE BUFFER.
************** A.PTR           : MOVING POINTER IN I/O BUFFER
                                : IT GENERALLY POINTS TO THE NEXT
                                : USEABLE CHARACTER SLOT.
************** A.END           : MOVING POINTER IN I/O BUFFER
                                : DURING OUTPUT IT POINTS TO THE
                                : MOST RECENTLY OUTPUT CHARACTER.
************** A.MAX           : POINTER TO LAST VALID CHARACTER
                                : POSTITION IN THE BUFFER.
;
;
;
************** PPMODE          : <PERCENTSIGN> MODE FLAG
                                : (SET IF <PERCENTSIGN> ENVOKED)
************** IECRLF          ; CR VS. CR/LF FLAG
************** NOOUT           : OUTPUT BUFFER FLAGS
NOWRIT = ^H01                   : IF SET NEVER OUTPUT BUFFER
LSTFMT = ^H02                   : SET TO OUTPUT IN LIST MODE
SNDIT  = ^H80                   : SET TO PUT EOI WITH LAST BYTE IN BUFFER
```

The following variables are of importance during input operations.

```
********** CRSTAT          : INPUT BUFFER STATUS BYTE
                           : (CRSTAT=0 IF NO DELIMITER IN BUFFER)
      CRNORM  = ^H01       : NORMAL EOLCHR <CR> FOUND
      CRDC3   = ^H02       : NOT USED
      CREOI   = ^H04       : EOI FOUND
      CREOF   = ^H08       : EOF FOUND (FILE ONLY)
      CRETX   = ^H10       : ETX FOUND (LOGICAL END-OF-FILE CHAR.)
      CREOT   = ^H20       : EOT FOUND (EOF ON INTERNAL TAPE)
      EOFTYP  = ^H38       : (EOF+ETX+EOT)
      CRVLD   = ^H80       : CRSTAT IS VALID IF SET
                           : (REACHED END OF BUFFER)
********** EOLCHR          : CHARACTER USED AS EOL DELIMITER
                           : NOTE: THIS CHARACTER IS ALSO
                           : USED DURING OUTPUT FUNCTIONS!!!!
********** ETXCHR          : CHARACTER USED AS EOF DELIMITER
********** NULCHR          : CHARACTER USED AS NULL CHARACTER
                           : NOTE: IF 128. OR GREATER NO CHARACTER
                           : WILL BE USED AS A NULL CHARACTER.
```

4051 SYMBOL DEFINITIONS

CONTAINED WITHIN IS A LIST OF THE 4051 SYSTEM GLOBAL SYMBOLS. THE SYMBOLS ARE LISTED IN ALPHABETICAL ORDER. EACH SYMBOL ENTRY CONTAINS INFORMATION IN THE FOLLOWING FORMAT:

```
SYMBOL(XXXX)T    COMMENT ABOUT THE SYMBOL.
```

SYMBOL SPECIFIES THE GLOBAL SYMBOL NAME.
(XXXX) SPECIFIES THE VALUE ASSOCIATED WITH THE GLOBAL SYMBOL.
T SPECIFIES THE TYPE OF THE SYMBOL ENCODED AS BELOW:

- E -- SYMBOL REFERS TO AN ENTRY POINT FOR AN INDIVIDUAL ROUTINE. THE ADDRESS OF THE ENTRY POINT IS INDICATED BY THE (XXXX) ENTRY.
- C -- SYMBOL IS A CONSTANT WITH THE VALUE XXXX.
- V -- SYMBOL IS A GLOBAL VARIABLE WITH THE ADDRESS XXXX. NOTE: IF XXXX IS LESS THAN 100 (HEX) THEN THE VARIABLE IS ON PAGE ZERO AND MAY BE ADDRESS USING THE DIRECT MODE.
- M -- SYMBOL IS A MAJOR MODULE NAME (THE NAME ASSOCIATED WITH EACH BLOCK OF LISTINGS). THE VALUE XXXX ASSOCIATED WITH THIS SYMBOL IS MEANINGLESS (AT LEAST PSEUDO RANDOM).

A Symbols

```
A0X     (AA43)E    ANX - ROUTINES TO ADD N TO THE X REGISTER.
A5X     (AA3E)E     " "
A6X     (AA3D)E     " "
A7X     (AA3C)E     " "
A8X     (AA3B)E     " "
A9X     (AA3A)E     " "
A10X    (AA39)E     " "
A11X    (AA38)E     " "
A12X    (AA37)E     " "
A13X    (AA36)E     " "
A14X    (AA35)E     " "
A15X    (AA34)E     " "
A16X    (AA33)E     " "
ABS     (EE06)E    ROUTINE TO GET ABSOLUTE FLOATING POINT VALUE.
ABSCOD  (003F)C    CODE FOR ABS TOKEN.
ACCEL   (05A4)V    KEYBOARD DRIVER ACCELERATION RATE.
ACIA    (87C6)C    THE ADDRESS OF THE ACIA IN THE COMM BACK PACK.
ACS     (00A8)C    BANK SWITCH CONTROL CONSTANT.
ACSCOD  (0048)C    TOKEN CODE FOR ARC COS FUNCTION.
ADBANK  (95DE)M    THE PART OF THE MATH PACK THAT IS IN THE OVERFLOW BANK.
ADMAIN  (E87A)M    THE PART OF THE ADVANCED MATH PACK IN THE MAIN BANK.
ADRDEV  (AE22)E    ROUTINE TO SET UP I/O SYSTEM STATUS
                   INCLUDES A.PRIM, A.SE SET UP.
AFPITT  (B05D)E    ROUTINE TO CONVERT ASCII TO FPN, INTERNAL BUFFERS.
ALLCOD  (00AE)C    CODE FOR ALL TOKEN.
ALLTG   (0014)C    ALL TAG - STACK TAG FOR DELETE ALL CONSTRUCT.
ANCRNT  (0056)V    AUTO NUMBER CURRENT - THE CURRENT VALUE FOR AUTO NUMBER
                   TO OUTPUT.
```

AND (EE25)E SUBROUTINE TO DO LOGICAL AND OF OPERANDS ON STACK.

ANDCOD(000A)C CODE FOR AND TOKEN.

ANHOLD(0462)V AUTO NUMBER HOLD - IF NOT ZERO DO NOT AUTO NUMBER ON THE NEXT LINE.

ANINCR(0068)V AUTO NUMBER INCREMENT - THE CURRENT INCREMENT FOR AUTO NUMBER.

ANYINT(050B)V INTERRUPTS PENDING COUNT FOR ROM PACK RECALL WHEN SYSTEM IS SAFE.

APPCOD(0059)C CODE FOR APP TOKEN.

APPEND(9204)E THE APPEND STATEMENT ROUTINE.

APPOLD(D9EC)E ROUTINE TO PERFORM I/O PORTION OF APPEND.

AS (AF82)E ROUTINE TO PROCESS AS. STACKS A TAG FOR USE LATER.

AS2COD(0051)C CODE FOR AS2 TOKEN.

A.END (0095)V MOVING POINTER IN I/O BUFFER. DURING OUTPUT IT POINTS TO THE MOST RECENTLY OUTPUT CHARATERS.

A.MAX (0093)V POINTER TO LAST VALID CHARACTER POSITION IN THE BUFFER.

A.PRIM(0090)V PRESENT PRIMERY ADDRESS.

A.PTR (0098)V MOVING POINTER IN I/O BUFFERR. IT GENERALLY POINTS TO THE NEXT USABLE CHARACTER SLOT.

A.SEC (0097)V PRESENT SECONDARY ADDRESS.

A.STAT(008F)V I/O SYSTEM STATUS FLAG BYTE.

A.STRT(0091)V POINTER TO FIRST VALID CHARACTER POSITION IN THE BUFFER.

ASC (0058)C BANK SWITCH CONTROL CONSTANT.

ASCCOD(C01E)C CODE FOR ASC TOKEN.

ASCFPN(B56F)E CONVERTS STRING OF ASCII CHARACTERS INTO A FLOATING POINT NUMBER.

ASCOD (0085)C CODE FOR AS TOKEN.

ASG (C003)E ASSIGN - THE ROUTINE TO IMPLEMENT NORMAL ASSIGNMENT.

ASGCOD(008C)C CODE FOR ASG TOKEN.

ASN (00A0)C BANK SWITCH CONTROL CONSTANT.

ASNCOD(0047)C CODE FOR ARC SIN TOKEN.

ASSCOD(0074)C CODE FOR ASS TOKEN.

ASSCSC(C030)E ASSIGN A SCALER TO A SCALER - SYSTEM UTILITY.

ASSIGN(1000)C ASSIGN COMMAND I/O SYSTEM CONTROL CONSTANT.

ATLOAD(830D)E AUTO LOAD FUNCTION - CALLED FROM IDLE LOOP IF AUTO LOAD IS REQUIRED.

ATN (00B0)C BANK SWITCH CONTROL CONSTANT.

ATNCOD(0049)C TOKEN CODE FOR ARC TAN.

ATNOFF(F588)E ROUTINE TO TURN OF ATN GPIB LINE.

ATNON (F57F)E ROUTINE TO TURN ON ATN GPIB LINE.

ATPROC(AF94)E ROUTINE TO PROCESS @<ATSIGN> OR <PERCENTSIGN> INFORMATION ON THE STACK.
SETS UP A.PRIM AND A.SEC.

ATSNTG(0012)C STACK TAG TO MARK @<ATSIGN> OR <PERCENTSIGN> INFORMATION.

AUTONO(8279)E AUTO NUMBER - CALLED FROM THE IDLE LOOP TO OUTPUT LINE NUMBERS IF REQUIRED.

AXICOD(006D)C CODE FOR AXI TOKEN.

AXIS (00D0)C BANK SWITCH CONTROL CONSTANT.

B Symbols

BACKUP(011D)E ROUTINE TO BACK UP ONE STACK ENTRY, R0 IS FAKE SP.

BAKARS(FE81)E MAGTAPE DRIVER ROUTINE TO BACKUP ONE PHYSICAL RECORD.

BAKSTG(0013)C STACK TAG FOR I/O SYSTEM BACKWARD POINTERS INTO THE I/O LIST ON THE STACK.

BANK (007C)V THE SAVED VALUE THAT THE BANK SWITCH IS CURRENTLY SET TO.

BANKSW(87C0)C THE ADDRESS OF THE BANK SWITCH REGISTER.

BELCAL(C727)E ROUTINE TO BEEP THE BELL.

BFRALC(AEC7)E ROUTINE TO ALLOCATE I/O BUFFERS BASED UPON A.PRIM.

BINCTL(CC56)M BINARY I/O CONTROL MODULE.

BLINK (00AE)V CURSOR BLINK CONTROL. SET TO DO WRITE-THRU CHARACTERS.

BNKADR(8800)C BANK ADDRESS - THE ADDRESS OF THE BANK SWITCHED AREA.

BNKEND(8816)C BANK END - THE ADDRESS OF THE END OF THE HEADER FIELD IN A BANK.

BNKMAP(880D)M BANK MAP - THE MODULE WITH THE HEADER FOR THE OVERFLOW ROM.

BOF (0009)C MAGTAPE DRIVER BEGGINING OF FILE FLAG.

BPIAMT(C0BA)V BUFFER FOR PIAMTB.

BRKCNT(043D)V BRACKET COUNT FOR DIMENSION/SUBSCRIPT (DIMSUB) ROUTINE.

BRKCOD(003C)C CODE FOR BRK TOKEN.

BSTMT (C0AC)M BASIC STATEMENT PROCESSORS.

BYTCAL(E5C4)V PERFORMS THE FOLLOWING CALCULATION ((AB+(INT1*INT2))*8)+5.

BYTCNT(001E)V BYTE ALLOCATION COUNT.

C Symbols

CALCOD(006A)C CODE FOR CAL TOKEN.

CALL (E286)E ROUTINE TO IMPLEMENT THE CALL STATEMENT.

CALLBS(E296)M MODULE CONTAINING CALL STATEMENT PROCESSOR.

CALLTG(0017)C CALL TAG - STACK TAG FOR CALL ENTRY.

CASCOD(009C)C CODE FOR CAS TOKEN.

CASE (C248)E ROUTINE TO IMPLEMENT THE SET CASE STATEMENT.

CAT (0098)C BANK SWITCH CONTROL CONSTANT.

CATCOD(0017)C CODE FOR CAT TOKEN.

CCODES(0098)V BUFFER FOR CONDITION CODES IN MAGTAPE DRIVER.

CDOPTR(043B)V CURRENT DATA OBJECT POINTER - THE ADDRESS OF THE CURRENT DATA ITEM IN A DATA STATEMENT.

CDSPTR(0439)V CURRENT DATA STATEMENT POINTER - THE ADDRESS OF THE CURRENT DATA STATEMENT. IF ZERO FIND THE FIRST DATA STATEMENT IN THE PROGRAM.

CHAR (009D)V GLOBAL CONTAINING FIRST CHARACTER OF ASCII STRING.

CHR (005C)C BANK SWITCH CONTROL CONSTANT.

CHRCNT(0089)V ACTUAL CHARACTER COUNT DURING STRING INPUT OR READ.

CHRCOD(001C)C CODE FOR CHR TOKEN.

CHRVCT(0473)V VECTOR TO OPTIONAL CHARACTER PAINTER ROUTINE.

CHSCAN(C888)E ROUTINE TO PAINT CHARACTERS ON THE DISPLAY. THIS IS THE ACTUAL PAINTER NOT THE ENTIRE CONTROLLER.

CIDLE (C65C)E ROUTINE THAT DISPLAYS THE CURSOR WHILE WAITING FOR KEYBOARD I/O TO COME TO A LOGICAL END.

CLOCOD(007E)C CODE FOR CLO TOKEN.

CLOSE (F7AA)E ROUTINE TO HANDLE THE CLOSE COMMAND.

CLPTR (004D)V CURRENT LINE POINTER - THE ADDRESS OF THE STATEMENT BEING EXECUTED.

CLRARG(CE74)E CLEAR ARGUMENT - SPECIAL ENTRY POINT INTO TYPARG.

CLRBNK(A9D4)E CLEAR BANK - ROUTINE TO SET THE BANK SWITCH TO ZERO.

```
CMD    (1A00)C    COMMAND COMMAND I/O SYSTEM CONTROL CONSTANT.

CMDCOD(0083)C     CODE FOR CMD TOKEN.

CMDTBL(BB1A)E     COMMAND TABLE - A SECTION OF THE EVALUATOR TABLES.

CMPFPN(B700)E     ROUTINE TO COMPARE TWO FLOATING POINT NUMBERS: ONLY ONE IS
                  ON STACK.

CNFRCS(FC57)E     MAGTAPE DRIVE ROUTINE TO CLEAR NFR COUNTER.

CNT    (0031)V    FLAG WHICH CONTROLS FLOATING POINT INPUT CONVERSION.

CNTL   (0591)V    KEYBOARD DRIVER CONTROL KEY FLAG.

COL    (0596)V    KEYBOARD COLUMN FLAG.

COLCNT(009E)V     I/O SYSTEM MATRIX COLUMN COUNT.

COLCOD(008D)C     CODE FOR COL TOKEN.

COLCT  (00AC)V    TEMP. COUNTER FOR DISPLAY CHARACTER PAINTER (COLUMN).

COMCNT(0619)V     COMMA REMAINDER COUNT.

COMCOD(0090)C     CODE FOR COM TOKEN.

COMFLG(00F5)V     COMMA FORMAT FLAG.

COMP   (003A)V    COMPRESS LINE FLAG FOR UNCOMPILE.

COMPR  (F9AA)E    THE MEMORY COMPRESS (GARBAGE COLLECTION) ROUTINE.

CONCOD(0086)C     CODE FOR CON TOKEN.

CONV1  (05CC)V    POINTER TO CONVERSION FACTOR FOR CURRENT TRIG MODE.

COPCOD(0073)C     CODE FOR COP TOKEN.

COPTYP(0A20)C     COPY COMMAND I/O SYSTEM CONTROL CONSTANT.

COPY   (F7C8)E    ROUTINE TO HANDLE THE COPY COMMAND.

COSCOD(0042)C     CODE FOR COS TOKEN.

COSTHA(04E2)V     COSINE OF ROTATED ANGLE FOR RELATIVE GRAPHICS.
                  IT'S AN FPN.

CPSIEC(C49D)E     ROUTINE TO SET CR VS. CR/LF. PRI@37,26:1 <OR 0>

CPSSET(C479)E     ROUTINE TO SET OPTIONAL DELIMITERS FOR  <PERCENTSIGN>
                  MODE I/O.
                  PRINT @37,0: EOLCHR, ETXCHR, NULCHR
```

CPYFLG(0008)C DISPLAY COPY SUSPENDED FLAG IN CRTSTT.

CRASH (041A)V HOLDING AREA FOR SYSTEM ERROR DATA.

CRCOD (0094)C CODE FOR CR TOKEN.

CREATE(0500)C CREATE COMMAND I/O SYSTEM CONTROL CONSTANT.

CRECOD(009F)C CODE FOR CRE TOKEN.

CRLF (C278)E ROUTINE TO SEND A CR TO THE OUTPUT BUFFER.

CRLFLF(C276)E ROUTINE TO SEND TWO CR'S TO THE OUTPUT BUFFER.

CROCOD(0084)C CODE FOR CRO TOKEN.

CROS (8F3A)E DISPLAY GRAPHIC POINTER ROUTINE.

CROSS (0048)C BANK SWITCH CONTROL CONSTANT.

CRSTAT(006A)V INPUT BUFFER STATUS FLAG BYTE.

CRTDRV(C634)M DISPLAY DRIVER MODULE.

CRTRST(C8BF)E ROUTINE TO RESTORE THE DISPLAY AFTER A MAGTAPE OPERATION.

CTKN (0053)V CURRENT TOKEN - A HOLDING AREA FOR THE TOKEN BEING EXECUTED.

CTLCHR(F22E)E ROUTINE TO SEND AN ASCII CHARACTER TO THE DISPLAY. IT ALSO HANDLES THE PAGE FULL CONDITIONS.

CURSER(00E6)V RELATIVE CURSOR POINTER.

CURSOR(00B2)V ASCII CODE FOR THE CURSOR CHARACTER.

D Symbols

```
D5X     (AA4C)E     DNX - SUBTRACT N FROM X REGISTER.
D6X     (AA4B)E     " "
D7X     (AA4A)E     " "
D8X     (AA49)E     " "
D9X     (AA48)E     " "
D10X    (AA47)E     " "
D11X    (AA46)E     " "
D12X    (AA45)E     " "
D13X    (AA44)E     " "
DATACN  (BE56)E     DATA CONSTANT - ROUTINE TO STACK CONSTANT DURING LINE
                    EXECUTION.
DATCOD  (00A5)C     CODE FOR DAT TOKEN.
DATIN   (CD9E)E     ROUTINE TO READ FROM A DATA STATEMENT.
DB      (05E0)V     DATA BASE POINTER (PRINT USING).
DDP     (05F3)V     DIGITS DECIMAL POINT.
DEF     (9396)E     SUBROUTINE TO IMPLEMENT DEF FNX(A)
DEFCOD  (00A7)C     CODE FOR DEF TOKEN.
DEFPNT  (D183)E     DEFAULT PRINT.
DEG     (C228)E     ROUTINE TO IMPLEMENT SET DEGREE STATEMENT.
DEGCOD  (0096)C     CODE FOR DEG TOKEN.
DEGCON  (E98A)C     DEGREE CONSTANT - CONVERSION FACTOR FOR TRIG RANGE
                    REDUCTION.  =8/360.  DEGCON+8 CONTAINS 360/8.
DELCOD  (0058)C     CODE FOR DEL TOKEN.
DELET   (E6AB)E     ROUTINE THAT DELETES PROGRAM LINES.
DELETE  (E61E)E     ROUTINE THAT PERFORMS THE DELETE STATEMENT.
DELY1S  (FEB0)E     ROUTINE TO WAIT SOME TIME IN 40USEC. INCREMENTS.
DELY3S  (FE32)E     ANOTHER WAIT ROUTINE.
```

DET (0011)C BANK SWITCH CONTROL CONSTANT.

DETADD(05CE)V HOLDING AREA FOR VALUE OF DETERMINATE.

DETCOD(00A2)C CODE FOR DET TOKEN.

DEXP (0004)V TWO BYTE BUFFER FOR EXPONENT DURING FLOATING POINT INPUT CONVERSION.

DIGCNT(05F6)V NUMBER OF OUTPUT DIGITS.

DIGFLG(05A7)V FLAG FOR FP INPUT: "NO DIGITS SEEN YET."

DIM (E34A)E DIMENSION TOKEN HANDLER ROUTINE.

DIMCNT(001A)V DIMENSIONSUBSCRIPT COUNTER.

DIMCOD(00A6)C CODE FOR DIM TOKEN.

DIMLP (002C)V DIMENSION LOOP COUNTCONTROL.

DIMSTR(E4D4)E DIMENSION (ALLOCATE MEMORY) FOR A STRING VARIABLE.

DIMSUB(E34A)M DIMENSIONSUBSCRIPT MODULE.

DIR (0900)C DIRECTORY COMMAND I/O SYSTEM CONTROL CONSTANT.

DIRCOD(007D)C CODE FOR DIR TOKEN.

DISBLE(A9CC)E DISABLE INTERRUPTS - ROUTINE TO SET SYSTEM NOT SAFE STATUS.

DISCNT(00B1)V CURSOR GENERATOR COUNT REGISTER.

DISKCL(AB3B)E DISK CALL - ROUTINE TO CALL DISK INTERFACE ROM PACK.

DIVCOD(000F)C CODE FOR DIV TOKEN.

DL (00EB)V DATA LENGTH.

DLTALL(C1A6)E DELETE ALL - ROUTINE TO DO A DELETE ALL.

DLTXS (049A)V DLTXS := XMAX VIEWPORT - XMIN VIEWPORT

DLTYS (04AA)V DLTYS := YMAX VIEWPORT - YMIN VIEWPORT

DN1 (05F4)V NUMBER OF LEFT DIGITS.

DN2 (05F5)V NUMBER OF RIGHT DIGITS.

DOFP (E8DE)E ROUTINE TO INTERPRET FLOATING POINT CODES.

DOLFLG(00F4)V DOLLAR FLAG.

DOTON (00AF)V DOT CONTROL REGISTER FOR CRTDRV.

DP (0008)V DATA POINTER.

DPCONT(C60A)V DECIMAL POINT CONTROL.

DRACOD(0003)C CODE FOR DRA TOKEN.

DRAW (0028)C BANK SWITCH CONTROL CONSTANT.

DRBUSY(C684)E ROUTINE TO WAIT FOR DISPLAY TO BE READY.

DREXTA(0058)E DIRECT EXIT A - A JMP INSTRUCTION THAT IS IN RAM FOR DIRTY EXITS FROM ROUTINES THAT MUST PUSH OR PULL PARAMETERS ON THE STACK.

DREXTB(005B)E DIRECT EXIT B - SEE DREXTA.

DSDRAW(F3AE)E ROUTINE TO SEND GDU FPN'S TO DISPLAY VECTORS (DRAW).

DSFLG (0022)V DIMENSIONSUBSCRIPT FLAG.

DSFONT(F42B)E ROUTINE TO SELECT DISPLAY CHARACTER FONT.
PRINT @32,18:FONT

DSFULL(F434)E ROUTINE TO SET UP PAGE FULL FUNCTION.
PRINT @32,26: FULL FUNCTION <0,1,2,3>

DSGRAF(F3B1)E ROUTINE TO SEND GDU FPN'S TO DISPLAY VECTORS.

DSHOME(C6C8)E ROUTINE TO HOME THE DISPLAY CURSOR.

DSMOVE(F3A8)E ROUTINE TO SEND GDU FPN'S TO DISPLAY VECTORS (MOVE).

DSPAGE(C6BE)E ROUTINE TO PAGE THE DISPLAY.

DSPCHR(F2A6)E ROUTINE TO SEND ASCII CHARACTERS TO THE DISPLAY.
NOTE: CONTROL CHARACTERS EXCEPT <CR> WILL
BE CONVERTED TO "LIST FORMAT" BEFORE DISPLAYING.

DSPCP2(F354)E ROUTINE TO INITIATE A DISPLAY COPY.

DSPCPY(F34B)E ROUTINE TO INITIATE A DISPLAY COPY, BUT SET COPY SUSPENDED FLAG IN CRTSTT IF DISPLAY DISABLED.

DSPGIN(F11F)E ROUTINE TO PERFORM DISPLAY INPUT FUNCTIONS.
INPUT AND GIN.

DSPLY (F447)E ROUTINE TO HANDLE THE PAGE AND HOME COMMANDS.

DSPOUT(F2EA)E ROUTINE TO SEND THE OUTPUT BUFFER TO THE DISPLAY IN AN APPROPRIATE MANNER AS DICTATED BY A.SEC.

DSPSTT(0071)V DISPLAY STATUS FLAG BYTE.

DSPVCT(046D)V VECTOR TO OPTIONAL DISPLAY CHARACTER HANDLER. USED BY ROM PACKS AND THE OPT. 1 COMMUNICATIONS.

DT (00ED)V DATA TYPE (1=NUMERIC).

E Symbols

E1 (05E3)V EXPONENT LSD.

E2 (05E4)V EXPONENT DIGIT.

E3 (05E5)V EXPONENT MSD.

EARLYW(0087)V MAGTAPE DRIVE STATUS FLAG.

EDTBFR(0221)V EDIT BUFFER - A 74 BYTE BUFFER FOR THE LINE EDITOR WORK AREA.

EDTCLR(81FB)E EDIT CLEAR - ROUTINE TO CLEAR THE LINE BUFFER. SETS BUFFER TO SPACES.

EDTCLS(822F)E EDIT CLOSE - ROUTINE TO CLOSE THE EDIT BUFFER. LINE IS NOT LOST.

EDTEND(0060)V EDIT END - A VARIABLE WITH THE END OF EDIT BUFFER ADDRES (EDTBFR+71).

EDTFLG(0478)V USED BY OPTION 1.

EDTMAX(0064)V EDIT MAX - WORKING VARIABLE FOR LINE EDITOR.

EDTPTR(0062)V EDIT POINTER - THE LOCATION OF THE CURSOR IN THE EDIT BUFFER.

EFLAG (05A9)V FLAG SET OF FLOATING POINT INPUT CONVERSION. "SEEN AN E?"

ENABLE(A994)E ENABLE INTERRUPTS - SET THE SYSTEM INTO THE SAFE STATUS.

END (AAB6)E ROUTINE TO IMPLEMENT THE END STATEMENT.

ENDCOD(0060)C CODE FOR END TOKEN.

ENDKEY(0040)C KBFLAG STATUS BIT -- RETURN KEY STRUCK, EDIT BUFFER READY FOR PROCESSING.

ENDTBL(041A)C END OF TABLE - END OF TABLES TO BE CLEARED BY INIT.

EOF (C0AC)E ROUTINE TO IMPLEMENT ON EOF(UNIT) THEN STMT AND OFF EOF(UNIT) STATEMENTS.

EOFCOD(009E)C CODE FOR EOF TOKEN.

EOFTBL(03C9)V EOF TABLE - TABLE TO STORE ACTIVE ON EOF LINE ADDRESSES.

EOICOD(00AD)C CODE FOR EOI TOKEN.

EOIOFF(F576)E ROUTINE TO TURN OF EOI GPIB BIT.

EOION (F56D)E ROUTINE TO TURN ON EOI GPIB BIT.

EOIRDY(F5AB)E ROUTINE TO CLEAR EOI INTERRUPT.

EOL (C12C)E END OF LINE - ROUTINE TO IMPLEMENT LINE TERMINATION FUNCTION.

EOLCHR(0078)V I/O SYSTEM END OF LINE CHARACTER. NORMALLY <CR>.

EOLTG (0018)C END OF LINE TAG - STACK TAG FOR NEW LINE MARK.

EOSTG (0019)C END OF STACK - STACK TAG FOR MARKING THE ABSOLUTE END OF STACK.

EQ (E9EE)E ENTRY FOR FUZZY EQ COMPARE.

EQU (0030)V FLAG SET DURING FP INPUT CONVERSION.

EQUCOD(0011)C CODE FOP EQU TOKEN.

ERAPP1(000E)C APPEND ERROR 1 - BAD ARGUMENTS ON STACK.

ERAPP2(000F)C APPEND ERROR 2 - ATTEMPTING TO APPEND INTO A NONEXISTANT LINE NUMBER.

ERARC (0018)C ERROR CODE FOR INVERSE TRIG FUNCTIONS.

ERASGN(0015)C ASSIGNMENT ARGUMENT INVALID.

ERATSN(0042)C INVALID @<ATSIGN> OR <PERCENTSIGN> SPECIFICATIONS.

ERBCMU(004F)C BAD COMMA USAGE.

ERBDPU(0050)C BAD DECIMAL POINT USAGE.

ERBMDU(004D)C BAD MODIFIER USAGE.

ERBPNU(004C)C BAD PARENTHESIS USAGE.

ERBRK (0028)C ABORT ERROR CODE.

ERCARC(001A)C NOT USED (WE HOPE).

ERCHAR(005E)C BAD CHARACTER ERROR, FROM LEX.

ERCLAI(000C)C ROM-PACK CALL ARGUMENT ERROR.

ERCNSF(0020)C CALL NAME NOT FOUND.

ERCTRA(0435)V CONTAINS TOKEN NUMBER AT WHICH ERROR OCCURED ON ENTRY TO UNLEX.

ERCTRB(0436)V CONTAINS CHARACTER NUMBER AT WHICH ERROR OCCURED ON EXIT FROM UNLEX.

ERDAGN(0056)C USING ERROR -- DATA GONE AFTER PAGE FULL.

ERDATM(0051)C DATA TYPE MISMATCH.

ERDEOR(0053)C EXPONENT OUT OF RANGE FOR 'D' OPERATOR.

ERDET (001F)C MATRIX INVERSION HAD A 0 DETERMINATE.

ERDIM (0009)C DIMENSION ERROR.

ERDOMN(001B)C DOMAIN ERROR.

EREOFN(0014)C EOF OF UNIT N

EREOM (0036)C ERROR -- END OF MEDIUM ON MAGTAPE.

EREXEC(0044)C AN ERROR WAS SEEN BY THE EXECUTE ROUTINE, EXEC.

EREXP (0004)C ERROR CODE FOR EXPONENTIAL - ARGUMENT TOO BIG, RESULT OVERFLOWED.

ERFBFR(0031)C INTERNAL ERROR -- FULL OUTPUT BUFFER WITH NOOUT SET.

ERFILE(003E)C FILE SYSTEM ERROR MESSAGE, INVALID I/O OP.

ERFIXN(005F)C FIX1A ATTEMPTED OF NEGATIVE NUMBER.

ERFLDO(0057)C FIELD OVERFLOW.

ERFNFD(0034)C MAGTAPE ERROR -- FILE NOT FOUND.

ERFORA(0013)C FOR ARGUMENT ERROR.

ERFPDV(0002)C ATTEMPT TO DIVIDE A FLOATING POINT NUMBER BY ZERO.

ERFPOV(0001)C FLOATING POINT RESULT OVERFLOWED.

ERFUZZ(0010)C ERROR IN SPECIFYING FUZZ PARAMETERS.

ERFXOV(0060)C FIX1A ATTEMPTED; RESULT OVERFLOWED.

ERIFC (0049)C ILLEGAL FORMAT CHARACTER.

ERINFS(0048)C OUT OF FORMAT STRING.

ERINTR(0058)C INVALID DATA TYPE TRANSFERED TO AN INTERNAL DEVICE.
PRINT @32,20: "50,65"

```
ERINU (004A)C     ILLEGAL NUMBER USAGE.
ERIOE (0045)C     GPIB BUS ERROR. NOBODY LISTENING (NRFD AND NDAC HIGH).
ERISGN(0054)C     IMAGE STRING GONE AFTER PAGE FULL
ERISTS(0055)C     IMAGE STRING TOO SHORT AFTER PAGE FULL.
ERLNNF(0033)C     REFERENCED LINE NOT FOUND.
ERLNNO(0007)C     LINE NUMBER OUT OF RANGE.
ERLOG (0017)C     ERROR CODE FOR LOG - ARGUMENT ZERO OR NEGATIVE.
ERLOLD(0032)C     LINE TO LONG TO INPUT UNDER OLD OR APP.
ERLRSU(004E)C     BAD LRS USAGE.
ERMARK(0012)C     NOT USED (WE HOPE).
ERMFMT(003A)C     MAGTAPE FORMAT ERROR DETECTED DURING FIND.
ERMILA(0037)C     ILLEGAL MAGTAPE ACCESS.
ERMOPN(003D)C     MAGTAPE FILE OPEN WHEN MARK CALLED.
ERMTRD(0035)C     MAGTAPE READ ERROR.
ERMUL (001E)C     MATRIX MULT. ARGUMENTS INVALID.
ERNCN (0021)C     NOT USED (WE HOPE).
ERNCRT(0039)C     NO MAGTAPE CARTRIDGE.
ERNDT (0022)C     DATA STATEMENT REFERENCE ERROR.
ERNFST(0047)C     NO FORMAT STRING.
ERNIMX(0023)C     INVALID STATEMENT IN IMMEDIATE MODE.
ERNIOD(0043)C     NO I/O DEVICE HANDLER FOR SPECIFIED A.PRIM.
ERNOFN(000B)C     FNN NOT DEFINED.
ERNOLD(003B)C     NOTHING FOUND TO OLD OR APPEND.
ERNON1(0029)C     SIZE ON UNIT.
ERNON2(002A)C     FULL ON UNIT.
ERNON3(002B)C     SRQ ON UNIT.
```

```
ERNON4(002C)C    EOI ON UNIT.

ERNON5(002D)C    EXTERNAL INTERRUPT  1.

ERNON6(002E)C    EXTERNAL INTERRUPT  2.

ERNON7(002F)C    EXTERNAL INTERRUPT  3.

ERNON8(0030)C    EOF N ON UNIT.

ERNSEP(0041)C    NOT USED.

ERNXFN(0025)C    EXTERNAL FUNCTION ROM REQUIRED.

ERNXTA(0013)C    NEXT STATEMENT INVALID.

EROFA (0013)C    GOTO N OF ... OR GOSUB N OF ... WHERE N IS INVALID.

EROVRD(003C)C    ILLEGAL MAGTAPE RECORD LENGTH ENCOUNTERED.

EROUOT(0046)C    INCOMPLETE LITERAL.

ERRCD (0045)V    ERROR CODE - HOLDING AREA FOR ERROR CODE.  IF ZERO NO
                 ERRORS HAVE OCCURED IN THIS LINE.

ERRCDB(004C)V    ERROR CODE B - BACK UP HOLDING AREA FOR ERRCD WHILE
                 PROCESSING SIZE ERRORS.

ERREAD(003F)C    BINARY HEADER ERROR.

ERREN (0011)C    RENUMBER ERROR.

ERREP (001C)C    REP FUNCTION ARGUMENT IS BAD.

ERROM1(0059)C    ANY ERROR IN ROM PACK.

ERROM2(005A)C    STOP EVAL ONLY -- ROM PACK USAGE.

ERSEC (0026)C    PROGRAM SECRET.

ERSFE (0019)C    CHR N --- N OUT OF RANGE.

ERSHAP(0008)C    MATRICIES ARE NOT CONFORMABLE.

ERSQR (0006)C    ERROR CODE FOR SQUARE ROOT.

ERSTGL(004G)C    NOT USED.

ERSTOP(005B)C    STOP ERROR CODE.

ERSUBC(000A)C    SUBSCRIPT ERROR.
```

```
ERSYNX(005D)C   PARSER DOWN ARROW.
ERTABO(0052)C   TABBING ERROR.
ERTERM(005C)C   PROGRAM ABORTED WITH BREAK KEY.
ERTMDS(004B)C   TOO MANY DATA SPECIFIERS.
ERTRNG(0005)C   ERROR CODE: TRIGONOMETRIC ARGUMENT TOO LARGE.
ERUNDF(0024)C   ARGUMENT UNDEFINED.
ERUP  (0003)C   ERROR CODE FOR 0**0.
ERUPN (0016)C   ERROR CODE FOR (NEG)**B.
ERVAL (001D)C   VAL A$ -- NO VALUE FOUND IN A$.
ERWBYT(000D)C   ILLEGAL VALUE SPECIFIED FOR WBYTE.
ERWPT (0038)C   MAGTAPE WRITE LOCKED.

ERWSFL(0027)C   WORKSPACE FULL ERROR (MEMORY FULL).
ES    (05E2)V   ASCII EXPONENT SIGN.
ESTG  (0002)C   EVALUATOR STATUS TAG - STACK TAG FOR EVAL STATUS.
ETOX  (EC4C)E   COMPUTES FLOATING POINT EXPONENTIAL FUNCTION E**X.
ETXCHR(0079)V   I/O SYSTEM END OF TEXT CHARACTER (EOF)
EVLEN (9368)M   EVAL ENTRY POINTS - MAIN ENTRY POINTS AND RECURSION
                POINTS FOR EVAL.
EVLSUB(CE70)M   EVAL SUBROUTINES - EVAL SERVICE SUBROUTINES.
EXEC  (F45C)E   ASCII TO HEX LOAD ROUTINE: LOADS STARTING AT SCRTCH.
EXH   (00D4)V   DECIMAL EQUIVALENT EXPONENT, HIGH ORDER.
EXL   (00D5)V   DECIMAL EQUIVALENT EXPONENT, LOW ORDER.
EXP   (05F9)V   CONDENSED EXPONENT.
EXPCOD(0046)C   CODE FOR EXP TOKEN.
EXS   (05FA)V   EXPONENT BINARY SIGN.
```

```
F1      (05F7)V    "AS REQUIRED" LEFT OF DECIMAL POINT.
F2      (05F8)V    "AS REQUIRED" RIGHT OF DECIMAL POINT.
FA      (0090)C    TAG FOR DOFP: ADD TWO WORDS ON STACK.
FART    (001C)C    TAG FOR DOFP: NO-OP USED WHEN ONLY FA,FD,FM, OR FS IS TO
                   BE PERFORMED.
FD      (00EC)C    TAG FOR DOFP: CIVIDE TWO WORDS ON STACK.
FDUP    (0016)C    TAG FOR DOFP: DUPLICATE TOP FLOATING POINT STACK ENTRY.
FFLG    (00F1)V    "AS REQUIRED" FLAG.
FFOT    (0004)C    MAGTAPE DRIVER FLAG --- NOT USED.
FHEX    (0000)C    TAG FOR DOFP: FLOATING POINT PUSH EXTENDED.
FHIM    (001D)C    TAG FOR DOFP: FLOATING POINT PUSH IMMEDIATE.
FHIN    (0008)C    TAG FOR DOFP: FLOATING POINT PUSH INDIRECT.
FIL     (E870)E    ROUTINE TO HANDLE  .
FILCOD  (004F)C    CODE FOR FIL TOKEN.
FILEIN  (E858)E    ROUTINE TO ESTABLISH LINKAGE TO FILE SYSTEM FOR INPUT.
FILEOT  (E868)E    ROUTINE TO ESTABLISH LINKAGE TO FILE SYSTEM FOR OUTPUT.
FILERQ  (E876)E    ROUTINE TO ESTABLISH LINKAGE WITH FILE SYSTEM DIRECTLY.
FILES   (E86C)E    ROUTINE TO ESTABLISH LINKAGE TO FILE SYSTEM
                   FOR THE COMMAND COMMAND.
FILFND  (00BD)V    MAGTAPE FILE TO BE FOUND REGISTER.
FILLOC  (00BB)V    PRESENT MAGTAPE FILE LOCATION REGISTER.
FINCOD  (0076)C    CODE FOR FIN TOKEN.
FIND    (1B21)C    FIND COMMAND I/O SYSTEM CONTROL CONSTANT.
FIX1    (B1A2)E    ROUTINE TO GET INTEGER PART OF FLOATING POINT NUMBER.
FIXNUM  (B08F)E    I/O ROUTINE TO FIX I/O LIST ENTRIES FOR INTERNAL USE.
FLAG    (0032)V    USED AS A COUNTER FOR LITERAL LENGTH IN ROUTINE UNCOMP.
FLAGS   (00F7)V    STATUS FLAG BYTE FOR PRINTF.
```

FLCTRL(9F16)M FLOW OF CONTROL - EVAL ROUTINES FOR FLOW OF CONTROL STATEMENTS. GOTO, GOSUB, OF, RUN, STOP, RETURN, IF, FOR AND NEXT.

FLDATA(BE56)M FLOW OF DATA - EVAL ROUTINES FOR STACKING CONSTANTS AND VARIABLES AND FOR MOVING DATA FOR USER DEFINED FUNCTIONS AND ASSIGN.

FLEX (000D)C TAG FOR DOFP: FLOATING POINT POP EXTENDED.

FLIN (0013)C TAG FOR DOFP: FLOATING POINT POP INDIRECT.

FLOAT1(B1FB)E SAME AS FLOAT2.

FLOAT2(B1FB)E ROUTINE TO CONVERT INTEGER TO FLOATING POINT.

FLUNIT(000B)C BANK CONTROL CONSTANTS FOR ON PAGE AND OFF PAGE STATEMENTS.

FM (0000)C TAG FOR DOFP: MULTIPLY TWO WORDS ON STACK.

FMTCLN(0208)E FORMAT STRING CLEAN-UP ROUTINE.

FMTFLG(00F6)V FORMAT TYPE IN PROCESS (0=FMTOUT).

FMTINI(0107)E FORMAT INITIALIZE ROUTINE.

FMTPNT(0178)E FORMAT DATA ITEM OUTPUT ROUTINE.

FNACOD(0024)C CODE FOR FNA TOKEN.

FNASGN(BEE1)E FUNCTION ASSIGN - ROUTINE TO DO ASSIGNMENT IN FNX LINES.

FNBCOD(0025)C CODE FOR FNB TOKEN.

FNCCOD(0026)C CODE FOR FNC TOKEN.

FNDCOD(0027)C CODE FOR FND TOKEN.

FNECOD(0028)C CODE FOR FNE TOKEN.

FNEVL (9368)E FUNCTION EVALUATION - ROUTINE TO IMPLEMENT FNX(PARM) CALLS.

FNFCOD(0029)C CODE FOR FNF TOKEN.

FNGCOD(002A)C CODE FOR FNG TOKEN.

FNHCOD(0026)C CODE FOR FNH TOKEN.

FNICOD(002C)C CODE FOR FNI TOKEN.

```
FNJCOD(002D)C    CODE FOF FNJ TOKEN.
FNKCOD(002E)C    CODE FOR FNK TOKEN.
FNLCOD(002F)C    CODE FOR FNL TOKEN.
FNMCOD(0030)C    CODE FOR FNM TOKEN.
FNNCOD(0031)C    CODE FOR FNN TOKEN.
FNOCOD(0032)C    CODE FOR FNO TOKEN.
FNPARM(BEC1)E    FUNCTION PARAMETER - ROUTINE TO FETCH PARAMETER FOR
                 FNX USAGE.
FNPCOD(0033)C    CODE FOR FNP TOKEN.
FNQCOD(0034)C    CODE FOR FNQ TOKEN.
FNRCOD(0035)C    CODE FOR FNR TOKEN.
FNSCOD(0036)C    CODE FCR FNS TOKEN.
FNTBL (03DD)V    FUNCTION TABLE - TABLE TO STORE ADDRESSES OF FNX LINES.
FNTCOD(0037)C    CODE FOR FNT TOKEN.
FNUCOD(0038)C    CODE FOR FNU TOKEN.
FNVCOD(0039)C    CODE FOR FNV TOKEN.
FNWCOD(003A)C    CODE FOR FNW TOKEN.
FNXCOD(003B)C    CODE FOR FNX TOKEN.
FNYCOD(003C)C    CODE FOR FNY TOKEN.
FNZCOD(003D)C    CODE FOR FNZ TOKEN.
FONT  (0083)V    PRESENT DISPLAY FONT.
FOR   (A026)E    ROUTINE TO IMPLEMENT THE FOR STATEMENT.
FORCOD(0050)C    CODE FOR FOR TOKEN.
FORTG (0004)C    FOR TAG - STACK TAG FOR FOR/NEXT STATMENT WORK AREA.
FPA   (05AA)V    8-BYTE BUFFER TO HOLD FP NUMBERS.
FPADD (B245)E    ROUTINE TO DO FP ADD.
FPAITT(B014)E    ROUTINE TO CONVERT FPN TO ASCII. INTERNAL BUFFERS.
```

FPC (053A)V 8-BYTE BUFFER FOR FP NUMBERS.

FPCMP (EE3E)E ROUTINE TO COMPARE TWO FP NUMBERS ON STACK.

FPDIV (B303)E ROUTINE TO DO FLOATING POINT DIVIDE.

FPMUL (B4AE)E ROUTINE TO DO FLOATING POINT MULTIPLY.

FPNASC(B853)E ROUTINE TO REDUCE A FP NUMBER TO A FRACTION FROM 0.1 TO 1.0 FOR CONVERSION TO ASCII LATER.

FPNEG (EBAA)E NEGATES FP NUMBER ON TOP OF STACK.

FPONE (E950)C FLOATING POINT 1.0

FPSUB (B23E)C ROUTINE TO DO FP SUBTRACT.

FPTWO (E92E)C FLOATING POINT 2.0

FPZERO(E926)C FLOATING POINT 0.0

FRES (0019)C TAG FOR DOFP: COPY FP NUMBER FROM FPA TO TOP OF STACK.

FRET (0000)C TAG FOR DOFP: END INTERPRETATION AND RETURN TO CALLER.

FS (00A0)C TAG FOR DOFP: SUBTRACT TWO WORDS ON TOP OF STACK.

FSAV (0018)C TAG FOR DOFP: POP FP NUMBER FROM TOP OF STACK TO FPA.

FSWP (0012)C TAG FOR DOFP: SWAP TOP TWO FP ENTRIES ON STACK.

FUDGH (0001)C FUDGE - HIGH BYTE OF STACK SPACE REQUIRED TO EXECUTE A LINE CONSTANT.

FUDGL (00F4)C LOW BYTE OF FUDGE CONSTANT.

FULCOD(00AB)C CODE FOR FUL TOKEN.

FULL (0030)V INDICATES WHEN DISPLAY BECOMES FULL.

FULSCN(F2C2)E ROUTINE TO SCAN "BLINKING F" WHILE WAITING FOR PAGE KEY.

FULSTT(058E)V INDICATES WHAT ACTION IS TO BE TAKEN WHEN DISPLAY BECOMES FULL.

FUZA (05C2)V CONTAINS FUZZ FACTORE FOR NON-ZERO COMPARES.

FUZB (05C3)V CONTAINS EXPONENT FOR FUZZY COMPARE TO ZERO.

FUZCOD(0080)C CODE FOR FUZ TOKEN.

FUZR (05CB)V VARIABLE OFR FUZZ: SET IN NORM: EQUALS NUMBER FO SHIFTS REQUIRED TO NORMALIZE FRACTION.

FUZZIE(B80D)E ROUTINE TO DO FUZZY COMPARE.

G Symbols

GENCUR(F36F)E ROUTINE TO GENERATE A BLINKING CURSOR.

GETCHR(F085)E ROUTINE TO GET NEXT CHARACTER FOR FP INPUT CONVERSION.

GETKEY(F7DC)E ROUTINE TO HANDLE THE KEYBOARD HARDWARE.

GETLAR(9106)E ROUTINE TO FIND THE ADDRESS AND LINE NUMBER OF THE LINE WITH THE SMALLEST LINE NUMBER LARGER THAN OR EQUAL TO THE ARGUMENT PASSED ON THE STACK.

GETLIN(9088)M MODULE CONTAINING GETSMA AND GETLAR.

GETLN (0087)E GET LINE - ROUTINE TO CONVERT FLOATING POINT NUMBER ON THE STACK TO THE ADDRESS OF THE GIVEN LINE.

GETLNA(0082)E GET LINE A - ROUTINE TO CONVERT INTEGER IN R0 TO THE ADDRESS OF THE GIVEN LINE.

GETSMA(90F8)E ROUTINE TO FIND THE ADDRESS AND LINE NUMBER OF THE LINE WITH THE LARGEST LINE NUMBER SMALLER THAN OR EQUAL TO THE ARGUMENT PASSED ON THE STACK.

GIN (0050)C BANK SWITCH CONTROL CONSTANT.

GINCOD(0063)C CODE FOR GIN TOKEN.

GINSET(92ED)E ROUTINE TO SET UP POINTERS FOR GIN INPUT.

GLBFLG(005E)V GLOBAL FLAGS - EVAL GLOBAL CONTROL FLAGS.

GOCOD (0052)C CODE FOR GO TOKEN.

GOSCOD(0053)C CODE FOR GOS TOKEN.

GOSTG (0003)C GOSUB TAG - STACK TAG FOR GOSUB/RETURN ENTRY.

GOSUB (9F60)E ROUTINE TO IMPLEMENT THE GOSUB STATEMENT.

GOTO (9F52)E ROUTINE TO IMPLEMENT TO GOTO STATEMENT.

GRACOD(0095)C CODE FOR GRA TOKEN.

GRAD (C22D)E ROUTINE TO IMPLEMENT SET GRAD COMMAND.

GRAF (8FFE)M GRAPHICS CONTROL MODULE.

GRAPH (9238)E ROUTINE TO HANDLE DRAW, MOVE, RDRAW, RMOVE, GIN.

GRDCON(E9CA)C GRAD CONSTANT - CONSTANT FOR TRIG RANGE REDUCTION. =9/400
GRDCON+8 = FP CONSTANT 400/8

GRFDR.(9469)E ROUTINE TO PERFORM CLIPPING ON VECTORS.

GRFINI(8FFE)E ROUTINE TO RESET GRAPHIC SYSTEM DEFAULT VALUES.

GSCNUM(F423)C GRAPHIC CONVERSION CONSTANT TO GDU'S TO TEKPOINTS. 1024*10C/780

GT (E9FE)E ENTRY FOR FUZZY GT COMPARE.

GTCOD (0016)C CODE FOR GT TOKEN.

GTDATA(8F02)E GET DATA - ROUTINE TO EXTRACT ONE OBJECT FROM THE DATA STATEMENTS IN THE PROGRAM.

GTE (E9F2)E ENTRY FOR FUZZY GT.EQ COMPARE.

GTECOD(0015)C CODE FOR GTE TOKEN.

H Symbols

HALTA (D150)E	HALT AND TAKE DREXTA - ROUTINE TO SET LINE EXECUTION ABORT FLAG.
HALTR (D159)E	HALT AND RETURN - ROUTINE TO SET ABORT FLAG.
HCTRCS(0388)C	POINTER TO THE END OF THE SYSTEM WORK AREA.
HEP (003F)V	HIGH END POINTER - THE ADDRESS OF THE LAST BYTE OF INSTALLED RAM.
HIRAMZ(BC66)C	HI RAM - LOAD X INSTRUCTION IN POWER UP ROUTINE WITH THE HIGHEST POSSIBLE RAM ADDRESS-10.
HLDEOF(0434)V	WORK AREA FOR ERROR MESSAGE WRITER.
HOMCOD(007C)C	CODE FOR HOM TOKEN.
HOME (1720)C	HOME COMMAND I/O SYSTEM CONTROL CONSTANT.

I Symbols

IDLE (C5B2)E ROUTINE THAT IS MASTER IDLE LOOP. ALL CONTROL IS PASSED FROM HERE ONCE THE SYSTEM HAS BEEN POWERED UP.

IDLVCT(047C)V VECTOR TO OPTIONAL ROUTINE TO GAIN CONTROL FROM MASTER IDLE LOOP.

IEC (AF82)E ROUTINE TO HANDLE @<ATSIGN>.

IECCOD(004E)C CODE FOR IEC TOKEN.

IECDRV(0001)M GPIB (IEC) CONTROL MODULE.

IECEOI(F591)E ROUTINE TO SERVICE EOI HARDWARE INTERRUPTS.

IECIFC(F55A)E ROUTINE TO GENERATE IFC FUNCTION ON GPIB.

IECIN (F60F)E ROUTINE TO RECEIVE A BUFFER FROM THE GPIB.

IECOFF(F408)E ROUTINE TO INITIALIZE GPIB HARDWARE TO IDLE STATE.

IECOUT(F5C0)E ROUTINE TO SEND A BUFFER ALONG THE GPIB.

IECRD (F4E3)E ROUTINE TO READ ONE CHARACTER FROM THE GPIB.

IECRLF(0463)V INDICATES CR VS. CR/LF FLAG STATUS.

IECSND(F50E)E ROUTINE TO SEND ONE CHARACTER ALONG THE GPIB.

IF (A013)E ROUTINE TO IMPLEMENT THE IF STATEMENT.

IFCOD (0054)C CODE FOR IF TOKEN.

IMACOD(00A3)C CODE FOR IMA TOKEN.

IMMTBL(A6A9)C IMMEDIATE MODE TABLE IN PARSE.

IMXTG (0007)C IMMEDIATE EXECUTE TAG - STACK TAG FOR ADDRESS OF A LINE.

INAGTE(B0C2)E SPECIALIZED ROUTINE TO CONVERT INTEGER TO ASCII.

INAITT(B004)E GENERAL ROUTINE TO CONVERT INTEGER TO ASCII. INTERNAL BUFFER.

INCXN (A2CC)E INCREMENTS INDEX REGISTER TO THE NEXT LEX TOKEN.

INFO (90A6)E RETURNS THE INFORMATION IN OPTABLE FOR THE SPECIFIED 4051 BASIC TOKEN.

INICOD(0062)C CODE FOR INI TOKEN.

INIT (C143)E ROUTINE TO IMPLEMENT INIT COMMAND.

INITMT(EC0C)E ROUTINE TO INITIALIZE MAGTAPE HARDWARE AND DISABLE DISPLA

INITX (C146)E SPECIAL ENTRY POINT TO INIT FOR POWER UP AND DELETE ALL ROUTINES TO USE.

INPCOD(0069)C CODE FOR INP TOKEN.

INPCTL(EFF8)M INPUT CONTROL MODULE.

INPNT (0098)V INPNT := A.PTR

INPSTG(EFF8)E ROUTINE TO INPUT A STRING.

INPUT (0D9F)C INPUT COMMAND I/O SYSTEM CONTROL CONSTANT.

INPVAL(F033)E ROUTINE TO INPUT A VALUE.

INT (EF3C)E ROUTINE TO GET INTEGER PART OF FPN AND RETURN AS FPN.

INT1 (0016)V INTEGER WORKING STORAGE.

INT2 (0018)V INTEGER WORKING STORAGE.

INTACP(F4B0)E ROUTINE TO SET GPIB HARDWARE TO LISTEN STATE.

INTCN (9E84)E INTEGER CONSTANT - ROUTINE TO FLOAT AND STACK INTEGER.

INTCOD(0043)C CODE FOR INT TOKEN.

INTMLA(E5FB)E INTEGER MULTIPLY ROUTINE WITH ACCUMULATE.

INTMLC(E5F9)E INTEGER MULTIPLY ROUTINE, NO ACCUMULATE.

INTQQQ(005E)V INTERRUPT ROUTINE TEMPORARY STORAGE.

INTSRC(F539)E ROUTINE TO INITIALIZE GPIB HARDWARE TO SOURCE STATE.

INTSRV(EFB8)E HARDWARE INTERRUPT SERVICE ROUTINE. DISPATCHES TO INDIVIDUAL HANDLERS USING THE PIATBL.

INV (0019)C BANK SWITCH CONTROL CONSTANT.

INVCOD(001B)C CODE FOR INV TOKEN.

IOBFR1(026B)C BEGINNING OF GENERAL I/O BUFFER.

IOCLNR(AB67)E ROUTINE TO CLEAN UP STACK AFTER I/O PROCESSING.

IOFLGS(008A)V I/O PROCESSOR STATUS FLAG BYTE.

IOFUNC(008E)V PRESENT I/O STATEMENT CODE FOR I/O PROCESSOR.

IOKONS(E61E)M MODULE CONTAINING THE DEFINITION OF ALL I/O SYSTEM CONTROL CONSTANTS.

IOPROC(AB5C)E ROUTINE TO PROCESS AN I/O LIST. PRINT, INPUT, READ, WRITE, RBYTE, WBYTE, FIND, MARK, KILL.

IOSCAN(AB86)E ACTUAL I/O LIST SCANNER.

IOSYSF(0467)V I/O SYSTEM ADDRESSED FLAG. USED BY ROM PACKS.

ISB (05DE)V IMAGE STRING BASE (PRINT USING)

ISL (00E9)V IMAGE STRING LENGTH.

ISP (00D6)V IMAGE STRING POINTER.

ITGCOD(00B2)C CODE FOR ITG TOKEN.

ITM1TG(000E)C ITEM ONE TAG - STACK TAG FOR AN INTEGER.

ITM2TG(000F)C ITEM TWO TAG - STACK TAG FOR AN INTEGER.

ITSINT(05A8)V FLAG FOR FP INPUT: "IT'S AN INTEGER."

J Symbols

```
JMPAX  (0450)E    SELF MODIFYING ROUTINE TO JMP TO (ACC-A)+(X).
                  JMPAX: STA A JMPAX+4  :MODIFY DISP FIELD
                   JMP 0,X  :JUMP TO ACC-A+X

JMPX   (043E)E    SELF MODIFYING ROUTINE TO ((ACC-C)+(X)).
                  JMPX: STA A JMPX+4  :MODIFY DISP FIELD
                   LDX 0,X  :FETCH JUMP ADDRESS
                   JMP 0,X  :EXIT

JUNK   (8F1B)C    LABEL ON THE BEGINNING OF OPTABLE.
```

K Symbols

```
K0      (0000)V    R0.
K1      (0001)V    R0+1.
K2      (0002)V    R1.
K3      (0003)V    R1+1.
K4      (0004)V    R2.
K5      (0005)V    R2+1.
K6      (0006)V    R3.
KBFLAG  (006B)V    KEYBOARD PROCESSOR STATUS FLAG BYTE.
KBIN    (0076)V    VALUE OF LAST KEY TRANSFERED BY THE KEYBOARD.
KBINT   (C4DE)E    ROUTINE TO PROCESS KEYBOARD HARDWARE INTERUPT.
KBMRK   (059A)V    POINTER INTO KEYBOARD PSEUDO STACK.
KBPROC  (C590)E    ROUTINE TO PROCESS KEYS AT THE PROGRAM LEVEL.
KBQIN   (C69A)E    ROUTINE TO STUFF A KEY INTO THE TYPE-AHEAD QUE.
KBQOUT  (C64A)E    ROUTINE TO FETCH A KEY FROM THE TYPE-AHEAD QUE.
KBSTK   (059C)V    KEYBOARD DRIVER PSEUDO STACK.
KERNEL  (05D6)V    FF NUMBER WHICH CONTAINS KERNEL OF RANDOM NUMBER GENERATOR.
KEY     (C240)E    ROUTINE TO IMPLEMENT SET KEY.
KEYCNT  (0599)V    DELAY COUNT FOR "OLDEST" KEY.
KEYCOD  (009A)C    CODE FOR KEY TOKEN.
KEYDRV  (F7DC)E    ROUTINE TO DECODE KEYBOARD HARDWARE INTO ASCII KEYS.
KEYIN   (F163)E    ROUTINE TO DO INPUT FROM KEYBOARD.
KEYQUE  (0100)C    BEGINNING OF KEYBOARD TYPE-AHEAD QUE.
KEYSTK  (0411)V    KEY STACK - USED TO STORE UNPROCESSED USER DEFINABLE
                   KEYS.  INPUTS FROM KEYBOARD DRIVER,  OUTPUTS IN EVAL.
KEYTIM  (0598)V    PRESENT KEY DELAY TIME REGISTER.
KILCOD  (0071)C    CODE FOR KIL TOKEN.
KILL    (D9DA)E    ROUTINE TO HANDLE KILL COMMAND.
KILTYP  (0721)C    KILL COMMAND I/O SYSTEM CONTROL CONSTANT.
KYRATE  (05A5)V    MAXIMUM KEYBOARD RATE.
```

L Symbols

```
L1      (00DA)V    REPEAT LOOP 1.

L2      (00DD)V    REPEAT LOOP 2.

L3      (00E0)V    REPEAT LOOP 3.

L4      (00E3)V    REPEAT LOOP 4.

LBKCOD  (0093)C    CODE FOR LBK TOKEN.

LBRKTG  (0010)C    LEFT BRACKET TAG - STACK TAG FOR LEFT BRACKET IN
                   OBJECT STRING.

LCLFLG  (0054)V    LOCAL FLAGS - EVAL FLAGS FOR ONE LINE.

LCOM    (05FF)V    COMMA COUNT.

LDAX    (0448)E    SELF MODIFYING ROUTINE TO LOAD ACC-A FROM (ACC-A)+(X).
                   LDAX: STA A LDAX+4  ;MODIFY DISP FIELD
                    LDA A 0,X  ;GET DATA BYTE
                    RTS   ;EXIT

LDBX    (0451)E    SELF MODIFYING ROUTINE TO LOAD ACC-B FROM (ACC-A)+(X).
                   LDBX: STA A LDBX+4  ;MODIFY DISP FIELD
                    LDA B 0,X  ;GET DATA
                    RTS   ;EXIT

LDIG    (05FC)V    LEFT DIGIT COUNT.

LDIGA   (05FD)V    LEFT DIGIT BEGINNING ADDRESS.

LDXX    (0445)E    SELF MODIFYING ROUTINE TO LOAD X FROM (ACC-A)+(X).
                   LDXX: STA A LDXX+4  ;MODIFY DISP FIELD
                    LDX 0,X  ;GET DATA
                    RTS

LEN     (0068)C    BANK SWITCH CONTROL CONSTANT.

LENCOD  (0020)C    CODE FOR LEN TOKEN.

LENGTH  (009A)V    DIMENSIONED STRING LENGTH FOR I/O PROCESSING OF STRINGS.

LETCOD  (0065)C    CODE FOR LET TOKEN.

LEX     (8826)E    LEXICAL ANALYZER. CONVERTS INPUT LINES FROM ASCII
                   TO INTERNAL FORMAT.

LEXCNT  (0036)V    VARIABLE RETURNED TO TRANSL BY LEX EQUAL TO THE
                   NUMBER OF BYTES OUTPUT INTO SCRTCH BY LEX.

LGN     (ED27)E    ROUTINE TO DO NATURAL LOGARITHM.
```

LGNCOD(0044)C TOKEN CODE FOR LGN.

LINCN (BE84)E LINE CONSTANT - ROUTINE TO FLOAT A LINE NUMBER AND STACK IT.

LINCOD(00B3)C CODE FOR LIN TOKEN.

LINELN(058F)V DISPLAY LINE LENGTH. MODIFIED BY OPT. 1 COMMUNICATIONS.

LISCOD(005A)C CODE FOR LIS TOKEN.

LIST (1320)C LIST COMMAND I/O SYSTEM CONTROL CONSTANT.

LISTTG(0005)C LIST TAG - STACK TAG FOR LIST COMMAND ENTRY.

LITCN (BE69)E LITERAL CONSTANT - ROUTINE TO PUSH POINTER TO LITERAL.

LITCOD(00B5)C CODE FOR LIT TOKEN.

LITFLG(00EE)V LITERAL FLAG.

LITREL(9598)E LITERAL RELATION - ROUTINE TO IMPLEMENT STRING COMPARE.

LNEVL (96BC)E LINE EVALUATOR - ROUTINE TO EVALUATE ONE LINE.

LNNOTG(000D)C LINE NUMBER TAG - STACK TAG FOR SAVED LINE NUMBER.

LOCTG (D0E9)E LOCATE TAG - ROUTINE TO SEARCH FOR A GIVEN TAG.

LOCTGR(D104)E LOCATE TAG IN A RANGE - ROUTINE TO SEARCH THE STACK FOR A TAG IN A RANGE OF VALUES.

LOG (ED27)E ROUTINE TO COMPUTE LOG TO BASE TEN.

LOG2 (E93E)C FP NATURAL LOG OF 2.0

LOGCOD(0045)C CODE FOR LOG TOKEN.

LPNCOD(0092)C CODE FOR LPN TOKEN.

LRSFLG(00EF)V <CR> SUPRESS FLAG FOR PRINTF.

LSP (0045)V LOW STACK POINTER - TOP OF STACK POINTER FOR LOW RAM STACK. THIS IS WORKING SPACE ALLOCATION AREA.

LSPS (05FB)V LEADING SPACES COUNT FOR PRINTF.

LSTCOD(00B6)C CODE FOR LST TOKEN.

LSTKEY(006C)V LAST VALID KEYCODE --- I DON'T THINK IT'S USED.

LT (E9FA)E ENTRY FOR FUZZY LT COMPARE.

LTCOD (0013)C CODE FOR LT TOKEN.

LTE (E9F6)E ENTRY FOR FUZZY LT.EQ COMPARE.

LTECOD(0014)C CODE FOR LTE TOKEN.

LUNNO (007D)V FOR USE BY FILE SYSTEM. ACTIVE LUN.

M Symbols

MAGEND(006F)V MAGTAPE DRIVER FLAG TO INDICATE MAGTAPE BUSY.

MARCOD(0079)C CODE FOR MAR TOKEN.

MARK (1C21)C MARK COMMAND I/O SYSTEM CONTROL CONSTANT.

MARKS (FC73)E ROUTINE TO MARK NEW MAGTAPE FILES.

MASK (0039)V UNLEX FLAG, IF NONZERO INDICATES THAT THE LAST CHARACTER OUTPUT WAS A SPACE.

MATCOD(0066)C CODE FOR MAT TOKEN.

MATMAT(94DB)E MATRIX-MATRIX - ROUTINE TO APPLY A SCALER OPERATOR OVER TWO ARRAYS.

MATMOV(BF8B)E MATRIX MOVE - ROUTINE TO ASSIGN ONE MATRIX TO ANOTHER.

MATOPR(94CA)M MATRIX OPERATORS - MODULE WITH MATRIX FUNCTIONS IN IT.

MATSCL(94E6)E MATRIX-SCALER - ROUTINE TO APPLY A SCALER TO AN ARRAY AND SCALER.

MATSIZ(CFFA)E ROUTINE TO CALCULATE CONTROL CONSTANTS FOR A MATRIX.

MAX (EF2C)E FINDS MAX OF TWO FP NUMBERS ON STACK.

MAXANS(B3A1)E ROUTINE TO PUSH LARGEST FP NUMBER ON STACK.

MAXCOD(0009)C CODE FOR MAX TOKEN.

MEMCOD(00A1)C CODE FOR MEM TOKEN.

MEMORY(C1FB)E ROUTINE TO IMPLEMENT THE MEMORY OPERATOR.

MFLG (00F3)V MINUS FLAG.

MIN (EF1B)E FINDS MINIMUM OF TWO FP NUMBERS ON STACK.

MINCOD(0008)C CODE FOR MIN TOKEN.

MKFILE(DB0A)E ROM PACK ENTRY FOR MARKING MAGTAPE FILES.

MMICOD(0022)C CODE FOR MMI TOKEN.

MNSCOD(000D)C CODE FOR MNS TOKEN.

MODMTH(DA02)E ROUTINE TO MODIFY MAGTAPE HEADERS.

MODT88(D9FF)E SPECIAL ROUTINE TO MODIFY MAGTAPE HEADERS.

MOVCOD(0001)C CODE FOR MOV TOKEN.

MOVE (0038)C BANK SWITCH CONTROL CONSTANT.

MPLCOD(0021)C CODE FOR MPL TOKEN.

MPY (0029)C BANK SWITCH CONTROL CONSTANT.

MPYCOD(0018)C CODE FOR MPY TOKEN.

MSKCTR(050A)V SYSTEM MASK COUNTER (ENABLE/DISABLE)

MTAFIN(DBDD)E ROUTINE TO LOACATE AND OPEN A NEW MAGTAPE FILE.

MTBFR (011F)C BEGINNING OF MAGTAPE RECORD BUFFER.

MTBINT(DDF5)E ROUTINE TO INITIALIZE MAGTAPE BUFFER POINTERS.

MTBSWP(DDFE)E ROUTINE TO SWAP MAGTAPE BUFFER POINTERS.

MTCHKS(DE21)E ROUTINE TO COMPUTE CHECKSUM ON A MAGTAPE RECORD.

MTCLOS(DD18)E ROUTINE TO CLOSE MAGTAPE FILES.

MTDRV (FB90)M MAGTAPE HARDWARE CONTROL MODULE.

MTEOF (0001)C MAGTAPE DRIVE END-FILE FLAG BIT.

MTERR (0590)V MAGTAPE RE-READ COUNTER.

MTFFST(0080)C MTSTT2 FLAG BIT INDICATES HEAD OVER A GAP.

MTFIND(DBD1)E ROUTINE TO LOACATE AND OPEN A NEW MAGTAPE FILE GIVEN AN FPN.

MTFLGS(0089)V MAGTAPE DRIVER HARDWARE FLAGS.

MTFPOS(0040)C MTSTAT FLAG BIT -- SET IF OPEN AND ACCESSED (READ/WRITE).

MTFSIZ(0080)C MTSTAT FLAG BIT -- SET IF OPEN FILE IS 128. BYTE RECORD FORMAT, ELSE 256. BYTE ASSUMED.

MTKILL(DA64)E ROUTINE TO PERFORM KILL N ON THE MAGTAPE.

MTMARK(DAA0)E ROUTINE TO MARK A NEW FILE.

MTMASK(0468)V NOT USED.

MTMAX (0083)V POINTER TO LAST CHARACTER IN MAGTAPE BUFFER.

MTNULL(DD0C)E ROUTINE TO NULL A MAGTAPE BUFFER.

MTOPEN(0020)C MTSTAT FLAG BIT -- SET IF MAGTAPE FILE IS OPEN.

MTPADR(DFE8)E ROUTINE TO ADDRESS, SET UP, MAGTAPE SYSTEM.

MTPIN (DF35)E ROUTINE TO GET A LOGICAL BUFFER FROM THE MAGTAPE BUFFER.

MTPINR(DE53)E MAGTAPE INPUT REQUEST VALIDITY INTERPRETER.

MTPINW(DE31)E MAGTAPE OUTPUT FUNCTION INTERPRETER.

MTPOUT(DD98)E ROUTINE TO SEND A BUFFER TO THE MAGTAPE BUFFER.

MTPRWD(C506)E ROUTINE TO DO A CONTROLLED REWIND OF THE MAGTAPE.

MTPTR (0085)V MOVING MAGTAPE BUFFER POINTER.

MTRBFR(DF43)E ROUTINE TO DO A CONTROLLED READ OF A MAGTAPE BUFFER.

MTREAD(DCA8)E ROUTINE TO READ A PHYSICAL MAGTAPE RECORD.

MTSASC(0008)C MTSTAT FLAG BIT -- PRESENT FILE IS ASCII

MTSBIN(0004)C MTSTAT FLAG BIT -- PRESENT FILE IS BINARY

MTSCRT(DFE1)E ROUTINE TO SET MAGTAPE SECRET FILE BIT.

MTSNEW(0010)C MTSTAT FLAG BIT -- PRESENT FILE IS NEW.

MTSPGM(0002)C MTSTAT FLAG BIT -- PRESENT FILE IS PROGRAM.

MTSRD (0010)C MTSTT2 FLAG BIT -- PRESENT MAGTAPE BUFFER VALID FOR READ.

MTSREG(0081)V MAGTAPE FORMAT STATUS REGISTER.

MTSSEC(0001)C MTSTAT FLAG BIT -- PRESENT MAGTAPE FILE IS SECRET.

MTSSET(DFBB)E ROUTINE TO SET THE MAGTAPE FORMAT STATUS REGISTER (MTSREG).
PRINT @33,0: LENGTH, CHECKSUM, HEADER

MTSTAT(0082)V MAGTAPE CONTROLLER STATUS FLAG BYTE.

MTSTT2(0070)V MAGTAPE CONTROLLER STATUS FLAG BYTE.

MTSWRT(0002)C MTSTT2 FLAG BIT -- PRESENT MAGTAPE BUFFER CONTAINS WRITTEN INFORMATION.

MTWRIT(DD3F)E ROUTINE TO WRITE A MAGTAPE BUFFER.

MTXBYT(00B4)V EXTRA "BYTE" COUNT FOR MARKING FILES.

MULCOD(000E)C CODE FOR MUL TOKEN.

MULPNT(00C6)V POINTER USED IN FP OUTPUT CONVERSION TO ACCESS PROPER POWER OF 10.

N Symbols

N1 (05E6)V ASCII # LSD.

N11 (05F0)V ASCII # DIGIT.

N12 (05F1)V ASCII # MSD.

NE (E9EA)E ENTRY FOR FUZZY NE COMPARE.

NECOD (0012)C CODE FOR NE TOKEN.

NEWBFR(F031)E ROUTINE TO GET A NEW INPUT BUFFER.

NEXCOD(0050)C CODE FOR NEX TOKEN.

NEXT (A066)E ROUTINE TO IMPLEMENT NEXT COMMAND.

NGTAB (BA3A)C FP TABLE OF NEGATIVE POWERS OF 10.

NIODEV(C42C)E ROUTINE TO SET NO I/O DEVICE ERROR MESSAGE.

NLPTR (004F)V NEXT LINE POINTER - THE ADDRESS OF THE NEXT LINE TO BE EXECUTED.

NMISRV(FDBC)E NON MASKABLE INTERRUPTE (NMI) SERVICE ROUTINE FOR THE MAGTAPE.

NOCASE(C240)E NO CASE - ROUTINE TO IMPLEMENT SET NOCASE COMMAND.

NOCCOD(009D)C CODE FOR NOC TOKEN.

NOKCOD(009B)C CODE FOR NOK TOKEN.

NOKEY (C244)E NO KEY - ROUTINE TO IMPLEMENT SET NOKEY COMMAND.

NOOUT (0087)V OUTPUT BUFFER STATUS FLAG BYTE.

NOP (0158)E NO OPERATION - ROUTINE TO IMPLEMENT NO OPERATION TOKENS.

NORCOD(0099)C CODE FOR NOR TOKEN.

NORM (B343)E ROUTINE TO NORMALIZE FP NUMBER AND CHECK FOR OVER/UNDERFLOW OR ZERO RESULT.

NORMAL(C239)E NORMAL - ROUTINE TO IMPLEMENT SET NORMAL COMMAND.

NORMR (B359)E ALTERNATE ENTRY TO NORM VIA PSHRET.

NOT (EE0E)E LOGICAL NOT SUBROUTINE.

NOTCOD(0023)C CODE FOR NOT TOKEN.

NS (05F2)V ASCII # SIGN.

NTPTR (0051)V NEXT TOKEN POINTER - THE ADDRESS OF THE NEXT TOKEN TO BE EXECUTED.

NULCHR(007A)V I/O SYSTEM NULL CHARACTER. THIS CHARACTER WILL BE DELETED FROM THE INPUT STEAM BEFORE BEING PASSED TO BASIC.

NULLTG(0000)C NULL TAG - STACK TAG FOR ONE NULL BYTE.

NUMCOD(0080)C CODE FOR NUM TOKEN.

NUMFLG(00F0)V NUMBER FLAG.

NUMOUT(9D43)E UNLEX ROUTINE TO CONVERT FPN TO ASCII AND OUTPUT IT.

0 Symbols

OF (9F87)E ROUTINE TO IMPLEMENT GOTO EXP OF LIST AND GOSUB EXP OF LIST COMMANDS.

OFCOD (0088)C CODE FOR OF TOKEN.

OFF (C107)E ROUTINE TO IMPLEMENT OFF COMMAND.

OFFCOD (00A8)C CODE FOR OFF TOKEN.

OLD (04A1)C OLD COMMAND I/O SYSTEM CONTROL CONSTANT.

OLDCOD (006E)C CODE FOR OLD TOKEN.

OLDONE (E08C)E SPECIAL ENTRYPOINT FOR OLD CONTROL.

ON (C11B)E ROUTINE TO IMPLEMENT ON COND THEN LINE COMMAND.

ONCOD (00A9)C CODE FOR ON TOKEN.

ONTBL (03B9)V ON TABLE - THE ADDRESSES OF THE ON UNIT PROCESSING COMMANDS.

ONUNIT (C100)E ROUTINE TO IMPLEMENT ON COND AND OFF COND COMMANDS.

OPECOD (0070)C CODE FOR OPE TOKEN.

OPEN (0300)C OPEN COMMAND I/O SYSTEM CONTROL CONSTANT.

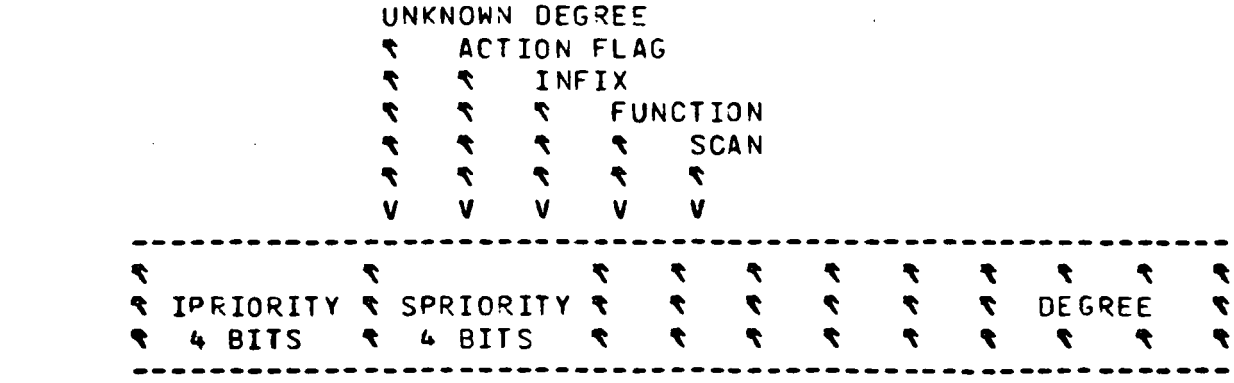
OPRADR (0056)V OPERATOR ADDRESS - HOLDING AREA FOR PRIMARY EVALUATOR DISPATCH DATA.

OPRCL (AAE4)E OPERATOR CALL - ROUTINE TO DO BANK SWITCHED LINKAGES FOR THE EVALUATOR.

OPRRTN (C437)V OPERATOR RETURN - SAVE AREA FOR RETURN POINT FROM OPRCL

OPRTBL (BAA2)C OPERATOR TABLE - SECTION OF EVALUATOR DISPATCH TABLES.

OPTABL (8F48)C OPTABLE, TRANSLATOR OPERATOR TABLE. THIS TABLE HAS ONE ENTRY FOR EACH TOKEN. EACH ENTY HAS THE FOLLOWING FORMAT.



OR (EE25)E LOGICAL OR SUBROUTINE.

ORCOD (000B)C TOKEN CODE FOR LOGICAL OR.

OUTBFR (C374)E ROUTINE TO SEND A FULL OUTPUT BUFFER TO THE CURRENT I/O DEVICE.

OVLN2 (E936)C FLOATING POINT CONSTANT = LOG2(E).

OVLTN (F946)C FP CONSTANT = LOG10(E).

P Symbols

PAETG (0008)C POINTER TO ARRAY ELEMENT TAG - STACK TAG.

PAGCOD(007B)C CODE FOR PAG TOKEN.

PAGE (1620)C PAGE COMMAND I/O SYSTEM CONTROL CONSTANT.

PARCNT(00E8)V PARENTHSIS COUNT.

PARSE (A206)E PARSER ENTRY POINT.

PCHAR (CBEE)E ROUTINE TO SEND A SINGLE CHARACTER TO THE DISPLAY. NOTE: DOES NOT LOOK AT PAGE FULL STATUS.

PFLG (00F2)V PLUS FLAG.

PGCTR (059C)V AUTO-ERASE DELAY COUNTER.

PGFVCT(0003)C BANK CONTROL CONSTANT FOR PAGE FULL.

PGMEVL(9683)E PROGRAM EVALUATOR - ROUTINE TO START AND RUN A PROGRAM.

PGMLIS(913E)E PROGRAM LIST BUILDER, INSERTS A NEW LINE INTO THE APPROPRIATE PLACE IN THE PROGRAM.

PGMPTR(0030)V PROGRAM POINTER - THE ADDRESS OF THE FIRST PROGRAM LINE.

PGMTG (0006)C PROGRAM TAG - STACK TAG FOR A POINTER TO A PPOGRAM LINE.

PGNEND(0600)C PAGE N END - THE ADDRESS OF THE FIRST FREE BYTE IN USER RAM. THE END OF SYSTEM RAM+1.

PGZEND(00F8)C PAGE ZERO END - THE FIRST FREE BYTE IN PAGE ZERO.

PI (E918)E ROUTINE TO STACK PI.

PIAEND(059A)C PIA END - THE ADDRESS OF THE END OF THE PIA TABLE.

PIAEOI(8790)C THE ADDRESS OF THE FIRST IEC BUS PIA.

PIAHX (8794)C THE ADDRESS OF THE CRT HI X DAC.

PIAHY (878C)C THE ADDRESS OF THE CRT HI Y DAC.

PIAKB (87A8)C THE ADDRESS OF THE KEYBOARD INTERFACE PIA.

PIALT (87AA)C THE ADDRESS OF THE PIA FOR THE LIGHTS AND THE REST OF THE KEYBOARD.

PIALX (8796)C THE ADDRESS OF THE CRT LOW X DAC.

PIALY (878E)C THE ADDRESS OF THE CRT LOW Y DAC.

PIAMTA(8798)C THE ADDRESS OF THE FIRST MAG TAPE PIA.

PIAMTB(879A)C THE ADDRESS OF THE SECOND MAG TAPE PIA.

PIASRQ(87B2)C THE ADDRESS OF THE SECOND IEC BUS PIA.

PIATBL(050C)C PIA TABLE - THE ADDRESS OF THE ORIGIN OF THE PIA TABLE.

PICOD (009F)C CODE FOR PI TOKEN.

PICON (E91E)C FP CONSTANT PI=3.1415926535...

PLOSTG(0001)C POINTER TO A LITERAL IN AN OBJECT STRING TAG - STACK TAG.

PLOTVL(93AA)E ROUTINE TO CONVERT USER SPACE TO SCREEN SPACE AND DUMP THE RESULTANT VECTOR INTO THE OUTPUT BUFFER.

PLSCOD(000C)C CODE FOR PLS TOKEN.

PLTAB (B9D2)C FP TABLE OF POSITIVE POWERS OF 10.

PNDEOF(006E)V PENDING EOF - IF NONE ZERO. THE NUMBER OF AN I/O UNIT WITH THE EOF CONDITION RAISED. NOTE PNDEOF=128 IF THE UNIT IS 0, THE INTERNAL TAPE.

PNDFLG(006D)V PENDING FLAGS - BIT FLAGS FOR THE ON CONDITIONS THE SYSTEM WILL ACCEPT. THE HIGH ORDER BIT IS SET BY BREAK.

PNTNTG(000A)C POINTER TO NAME TABLE NUMERIC TAG - STACK TAG.

PNTSTG(0008)C POINTER TO NAME TABLE STRING TAG - STACK TAG.

POINT (009C)V I/O SYSTEM POINTER TO PRESENT VALUE OR STRING.

POINTB(0476)V I/O PROCESSOR POINTER FOR RECOVERING AFTER PAGE FULL.

POLCOD(0064)C CODE FOR POL TOKEN.

POLL (F6A2)E ROUTINE TO HANDLE THE POLL STATEMENT.

POPES (007C)E POP EVALUATOR STATUS - ROUTINE TO PULL EVALUATOR STATUS AND RESET IT IN PAGE ZERO.

POS (0070)C BANK SWITCH CONTROL CONSTANT.

POSCOD(0007)C CODE FOR POS TOKEN.

PPEOF (0465)V <PERCENTSIGN> MODE EOF CHARACTER.

PPEOR (0464)V <PERCENTSIGN> MODE EOL CHARACTER.

PPMODE(0077)V <PERCENTSIGN> MODE ENVOKED FLAG.

PPNUL (0466)V <PERCENTSIGN> MODE NULL CHARACTER.

PRGBLN(DA3E)C POINTER TO BLANK PROGRAM MAGTAPE HEADER.

PRGDAT(DA52)C POINTER TO "DATA" PROGRAM MAGTAPE HEADER.

PRGPGM(DA48)C POINTER TO "PROGRAM" PROGRAM MAGTAPE HEADER.

PRICOD(005C)C CODE FOR PRI TOKEN.

PRIDFT(C288)E ASCII OUTPUT CONTROL ROUTINE.

PRINT (0C20)C PRINT COMMAND I/O SYSTEM CONTROL FUNCTION.

PRINTF(D178)M ASCII FORMATTING MODULE.

PRISTG(C264)E ROUTINE TO PRINT A STRING.

PRIVAL(C27E)E ROUTINE TO PRINT A VALUE.

PRMCOD(00B4)C CODE FOR PRM TOKEN.

PRTERR(83B8)E PRINT ERROR - THE ERROR MESSAGE WRITER.

PRTTG (0016)C PRINT TAG - STACK TAG FOR INFORMATION SAVED DURING PAGE FULL ON UNIT PROCESSING.

PSCTG (0009)C POINTER TO STRING COUNT STACK TAG.

PSHFPN(B6EB)E ROUTINE TO PUSH FP NUMBER ON STACK.

PSHRET(A9E7)E PUSH RETURN - SAVES RETURN ADDRESS ON A SECOND STACK.

PT   (AF7F)E ROUTINE TO HANDLE <PERCENTSIGN> CALL.

PTCOD (004D)C CODE FOR PT TOKEN.

PTRNTN(BE99)E ROUTINE TO STACK NUMERIC VARIABLE.

PTRNTS(BEB1)E ROUTINE TO STACK STRING VARIABLE.

PULFPN(B70F)E ROUTINE TO POP FP NUMBER FROM STACK.

PULLRN(AA19)E PULL REGISTER N - ROUTINE TO PULL ANY REGISTER FROM THE STACK.

PUPTAP(FB90)E ROUTINE TO INITIALIZE MAGTAPE HARDWARE.

PUSHES(D062)E PUSH EVALUATOR STATUS - ROUTINE TO SAVE EVALUATOR STATUS ON THE STACK AND TO CLEAR THE COPY IN PAGE ZERO.

PUSHX (AA08)E PUSH X - ROUTINE TO PUSH X AND GIVE IT ITM1TG.

PUSHXT(AA0A)E PUSH X - ROUTINE TO PUSH X AND GIVE IT A SPECIAL TAG.

PUTBYT(C33D)E ROUTINE TO PUT A CHARACTER IN THE OUTPUT BUFFER.

PUTCHR(9C6E)E ROUTINE TO PUT CHARACTERS INTO OUTPUT BUFFER.

PWRUP (BC4B)E POWER UP - THE SYSTEM POWER UP ROUTINE.

Q Symbols

Q0	(8B7C)C	POINTER TO THE BEGINNING OF THE KEYWORD TABLE.
Q1	(8BE7)C	POINTER TO THE BEGINNING OF THE 2ND QUARTER OF THE KEYWORD TABLE.
Q2	(8CA2)C	POINTER TO THE BEGINNING OF THE 2ND HALF OF THE KEYWORD TABLE.
Q3	(8D8E)C	POINTER TO THE BEGINNING OF THE 4RD QUARTER OF THE KEYWORD TABLE.
QACS	(95DE)E	ROUTINE TO IMPLEMENT THE ACS FUNCTION.
QASC	(8ED4)E	ROUTINE TO IMPLEMENT THE ASC STRING FUNCTION.
QASN	(95DE)E	ROUTINE TO IMPLEMENT THE ASN FUNCTION.
QATN	(95FE)E	ROUTINE TO IMPLEMENT THE ATN FUNCTION.
QAXIS	(884C)E	ROUTINE TO IMPLEMENT THE AXIS COMMAND.
QCAT	(8D4E)E	ROUTINE TO IMPLEMENT THE CAT () STRING FUNCTION.
QCHR	(8EEC)E	ROUTINE TO IMPLEMENT THE CHR STRING FUNCTION.
QCROSS	(8F1A)E	ROUTINE TO IMPLEMENT THE POINTER COMMAND.
QDRAW	(9215)E	ROUTINE TO IMPLEMENT THE DRAW COMMAND.
QEND	(C11E)C	LAST POSITION IN THE TYPE-AHEAD QUE.
QGIN	(9231)E	ROUTINE TO IMPLEMENT THE GIN COMMAND.
QIN	(0072)V	TYPE-AHEAD QUE INPUT POINTER.
QLEN	(8E8C)E	ROUTINE TO IMPLEMENT THE LEN STRING FUNCTION.
QMOVE	(9223)E	ROUTINE TO IMPLEMENT THE MOVE COMMAND.
QOUT	(0074)V	TYPE-AHEAD QUE OUTPUT POINTER.
QPOS	(8BE2)E	ROUTINE TO IMPLEMENT THE POS STRING FUNCTION.
QRDRAW	(921C)E	ROUTINE TO IMPLEMENT THE RDRAW COMMAND.
QREP	(8DC1)E	ROUTINE TO IMPLEMENT THE REP STRING FUNCTION.
QRMOVE	(922A)E	ROUTINE TO IMPLEMENT THE RMOVE COMMAND.
QRND	(96F9)E	ROUTINE TO IMPLEMENT THE RND FUNCTION.
QROTAT	(9055)E	ROUTINE TO IMPLEMENT THE ROTATE COMMAND.

QSCALE(9176)E ROUTINE TO IMPLEMENT THE SCALE COMMAND.

QSEG (8C8A)E ROUTINE TO IMPLEMENT THE SEG STRING FUNCTION.

QSETFU(97A2)E ROUTINE TO IMPLEMENT THE FUZZ COMMAND.

QSTR (8BA6)E ROUTINE TO IMPLEMENT THE STR STRING FUNCTION.

QTRNSL(C62A)C THE ADDRESS IN TRANSL THAT IS STACKED FOR EVAL.

QUSRFN(938D)C THE ADDRESS IN EVLEN STACKED FOR A FNX CALL.

QVAL (8B77)E ROUTINE TO IMPLEMENT THE VAL STRING FUNCTION.

QVIEW (9130)E ROUTINE TO IMPLEMENT THE VEIWPORT COMMAND.

QWINDO(90CC)E ROUTINE TO IMPLEMENT THE WINDOW COMMAND.

R Symbols

R0	(0000)V	THE FIRST OF 24 16 BIT SYSTEM REGISTERS.
R1	(0002)V	SYSTEM REGISTER N.
R2	(0004)V	" "
R3	(0006)V	" "
R4	(0008)V	" "
R5	(000A)V	" "
R6	(000C)V	" "
R7	(000E)V	" "
R8	(0010)V	" "
R9	(0012)V	" "
R10	(0014)V	" "
R11	(0016)V	" "
R12	(0018)V	" "
R13	(001A)V	" "
R14	(001C)V	" "
R15	(001E)V	" "
R16	(0020)V	" "
R17	(0022)V	" "
R18	(0024)V	" "
R19	(0026)V	" "
R20	(0028)V	" "
R21	(002A)V	" "
R22	(002C)V	" "
R23	(002E)V	" "
RAD	(C221)E	ROUTINE TO IMPLEMENT SET RAD COMMAND.
RADCOD	(0097)C	CODE FOR RAD TOKEN.

RADCON(E90A)C CONSTANT FOR RANGE REDUCTION IN TRIG ROUTINES. 4/PI
RADCON+8 = FP CONSTANT PI/4

RBYCOD(0077)C CODE FOR RBY TOKEN.

RBYTE (F1BF)E ROUTINE TO IMPLEMENT RBYTE.

RBYTVL(F20F)E ROUTINE TO GET A SINGLE BYTE FROM THE GPIB IN
RBYTE FORMAT.

RDIG (0602)V RIGHT DIGITS PRINTING COUNT.

RDIGA (0603)V RIGHT PRINT POINTER (ADDRESS).

RDRAW (0030)C BANK SWITCH CONTROL CONSTANT.

RDRCOD(0004)C CODE FOR RDR TOKEN.

REACOD(0067)C CODE FOR REA TOKEN.

READ (0EA2)C READ COMMAND I/O SYSTEM CONTROL CONSTANT.

READRS(FD54)E MAGTAPE DRIVER TO READ A PHYSICAL RECORD.

REASTG(CCCF)E ROUTINE TO READ A STRING.

REAVAL(CCAA)E ROUTINE TO READ A VALUE.

RECONT(0085)V PRESENT AVAILABLE RECORD COUNT FOR OPEN
MAGTAPE FILE.

RECNO (007E)V FILE SYSTEM VARIABLE, PRESENT FILE REQUESTED.

RELBL (0B00)C RELABEL COMMAND I/O SYSTEM CONTROL CONSTANT.

RELCOD(009A)C CODE FOR REL TOKEN.

RELTBL(A73C)C RELABEL TABLE IN PARSE.

REMCOD(00A4)C CODE FOR REM TOKEN.

RENCOD(0057)C CODE FOR REN TOKEN.

RENUM (9DC8)E RENUMBER ACCORDING TO REGISTER DIRECTIVES.

RENUMB(9D5A)E RENUMBER ACCORDING TO STACK DIRECTIVES.

REP (0078)C BANK SWITCH CONTROL CONSTANT.

REPCOD(0006)C CODE FOR REP TOKEN.

RESCOD(0056)C CODE FOR RES TOKEN.

```
RESTS (BF65)E    ROUTINE TO IMPLEMENT RESTORE STATEMENT NUMBER.
RESTZ (BF5C)E    INTERNAL ROUTINE TO RESET DATA STATEMENT POINTERS.
RETCOD(005E)C    CODE FOR RET TOKEN.
RETH  (0024)V    TEMPORARY FOR RETURN ADDRESS
RETL  (0025)V    " "
RETURN(9FE4)E    ROUTINE TO IMPLEMENT RETURN STATEMENT.
REWIND(FB07)E    ROUTINE TO REWIND THE MAGTAPE.
RMOCOD(0002)C    CODE FOR RMO TOKEN.
RMOVE (0040)C    BANK SWITCH CONTROL CONSTANT.
RND   (0088)C    BANK SWITCH CONTROL CONSTANT.
RNDATA(B949)E    ROUTINE TO ROUND FP NUMBER AND CONVERT TO ASCII.
RNDCOD(C042)C    CODE FOR RND TOKEN.
ROTATE(C020)C    BANK SWITCH CONTROL CONSTANT.
ROTCOD(0082)C    CODE FOR ROT TOKEN.
ROW   (0595)V    KEYBOARD DRIVER ROW OF PRESENT KEY.
ROWCT (00AD)V    DISPLAY CHARACTER ROW COUNTER.
RPN   (E354)E    RIGHT PARENTHESIS TOKEN HANDLER.
RPNCOD(008F)C    CODE FOR RPN TOKEN.
RTRN  (A9FD)E    ROUTINE TO PULL USE ALTERNATE RETURN POINT.  SEE
                 PSHRET.
RTRNTG(0015)C    RETURN TAG - STACK TAG FOR A INTERNAL RETURN ADDRESS.
RUN   (9F16)E    ROUTINE TO IMPLEMENT RUN COMMAND.
RUNBNK(AA7F)E    ROUTINE TO RUN ALL BANKS FOR POWER UP ECT..
RUNCOD(0055)C    CODE FOR RUN TOKEN.
RWNDFG(0010)C    DSPSTT FLAG BIT -- REWIND PENDING.
```

S Symbols

SAFE (A99E)E ROUTINE TO CALL IF SYSTEM IS SAFE FOR ROM INTERRUPTS BUT IT IS DISABLED.

SAVCOD(0058)C CODE FOR SAV TOKEN.

SAVE (0121)C SAVE COMMAND I/O SYSTEM CONTROL CONSTANT.

SBP (0043)V STACK BASE POINTER - BOTTOM OF EXECUTION STACK.

SCACOD(0081)C CODE FOR SCA TOKEN.

SCALE (0008)C BANK SWITCH CONTROL CONSTANT.

SCLMAT(94F4)E SCALER-MATRIX - ROUTINE TO APPLY A SCALER OPERATOR OVER A SCALER AND AN ARRAY.

SCMSED(0479)V PRESENT SCRAMBLE CHARACTER.

SCORE (0038)V USED IN ROUTINE UNCOMP TO DECIDE WHEN TO UNSTACK AND OPERATOR INTO THE RESULT.

SCRTCH(0235)V SCRATCH - SYSTEM WORK AREA IN PAGE N.

SECCOD(0072)C CODE FOR SEC TOKEN.

SECENC(1025)C SECRET COMMAND I/O SYSTEM CONTROL CONSTANT.

SEG (0080)C BANK SWITCH CONTROL CONSTANT.

SEGCOD(0095)C CODE FOR SEG TOKEN.

SEMCOD(008E)C CODE FOR SEM TOKEN.

SEMITG(0011)C SEMI COLON TAG - STACK TAG.

SERCHS(FECC)E MAGTAPE HARDWARE DRIVER TO DO FAST SEARCH.

SETARG(CE70)E SET UP FOR TYPARG - ROUTINE TO PRIME TYPARG CALLS.

SETBNK(A905)E SET BANK - ROUTINE TO SWITCH BANK AND DO HOUSEKEEPING.

SETCOD(0051)C CODE FOR SET TOKEN.

SETERA(D16A)E SET ERROR AND TAKE EXIT A - ROUTINE TO SET THE ERROR CODE AND RETURN USING DREXTA.

SETERR(D165)E SET ERROR CODE - ROUTINE TO SET THE ERROR CODE AND RETURN TO THE ORIGINAL CALLER.

SETFUZ(00C0)C BANK SWITCH CONTROL CONSTANT.

SETRND(EFA3)E SET DEFAULT VALUE FOR KERNAL OF RND.

IFT1(0593)V KEYBOARD DRIVER SHIFT KEY FLAG.

SHIFT2(0594)V ANOTHER KEYBOARD DRIVER SHIFT KEY FLAG.

SHMR (B21D)E ROUTINE TO RIGHT SHIFT FRACTION PRIOR TO ADD/SUBTRACT.

SHNTCT(0033)V VARIABLE USED TO RETURN THE NUMBER OF BYTES NEEDED TO STORE THE POSTFIX LINE. GENERATED BY SHUNT, RETURNED TO TRANSL.

SHUNT (A0DE)E CONVERTS THE PREFIX-INFIX INTERNAL NOTATION LINES PRODUCED BY LEX INTO POSTFIX, EXECUTABLE LINES.

SIG (EDE7)E CODE TO COMPUTE SIGN OF A FP NUMBER.

SIGCOD(0041)C CODE FOR SIG TOKEN.

SIGNS (00C3)V FLAG SET IN FP INPUT CONVERSION:"SIGN OF NUMBER."

SINCOD(004C)C CODE FOR SIN TOKEN.

SINTHA(04DA)V SINE OF ROTATED ANGLE FOR RELATIVE GRAPHICS. AN FPN.

SIZCOD(00AA)C CODE FOR SIZ TOKEN.

IZERR(0006)C SIZE ERROR - BREAK POINT FROM SIZE ERRORS TO NORMAL ONES.

SKIPS (FE84)E MAGTAPE DRIVER ROUTINE TO SKIP A RECORD.

SLN (EBB6)E ROUTINE TO LEFT SHIFT FRACTION OF FP NUMBER.

SLOPH (0009)C SLOP HIGH BYTE - SPACE DIFFERENCE BETWEEN STACK REQUIPMENT AND FUDGE.

SLOPL (0064)C SLOPE LOW BYTE.

SM2COD(00AF)C CODE FOR SM2 TOKEN.

SNDBFR(C404)E ROUTINE TO SEND AN OUTPUT BUFFER AND SET EOI BIT.

SNGMAT(94CA)E SINGLE MATRIX - ROUTINE TO APPLY A MONADIC SCALER OPERATOR OVER ONE ARRAY.

SP1 (00DB)V RELATIVE STRING POINTER 1.

SP2 (00DE)V RELATIVE STRING POINTER 2.

SP3 (00E1)V RELATIVE STRING POINTER 3.

SP4 (00E4)V RELATIVE STRING POINTER 4.

ACE (C1D5)E ROUTINE TO IMPLEMENT SPACE COMMAND.

SPACOD(00A6)C CODE FOR SPA TOKEN.

SPCIOB(AB83)E ENTRY POINT INTO IOSCAN TO SCAN A RBYTE/WBYTE LIST.

SPCIOF(1020)C SPECIAL FUNCTION COMMAND I/O SYSTEM CONTROL CONSTANT. THIS IS USED BY SOME INTERNAL BUFFER REQUSETS.

SPOLON(F798)E ROUTINE TO SEND SERIAL POLL ENABLE.

SQR (EB0F)E ROUTINE TO COMPUTE SQUARE ROOT OF FP NUMBER.

SQRCOD(0040)C CODE FOR SQR TOKEN.

SRQCOD(00AC)C CODE FOR SRQ TOKEN.

SRQOFF(C19F)E RESETS SRQ PENDING BIT.

SRQRDY(F5AC)E ROUTINE TO PROCESS SRQ LEVEL REQUESTS.

STAT37(007B)V 4051 PROCESSOR IN SECRET OUTPUT STATUS.

STAX (0457)E SELF MODIFYING CODE TO STORE ACC=B AT (ACC-A)+(X).
```
STAX: STA A STAX+4  :MODIFY DISP FIELD
      STA B 0,X     :STORE DATA
      RTS
```

STECOD(C088)C CODE FOR STE TOKEN.

STGCOD(C0B1)C CODE FOR STG TOKEN.

STKBLD(E82E)E ROUTINE TO BUILD A BACKWARDS STACK FOR I/O LISTS.

STKUSE(0597)V KEYBOARD DRIVER STACK USAGE COUNTER.

STOCOD(005F)C CODE FOR STO TOKEN.

STOP (9F4E)E ROUTINE TO IMPLEMENT STOP COMMAND.

STPTR (003B)V SYMBOL TABLE POINTER.

STR (0098)C BANK SWITCH CONTROL CONSTANT.

STRCOD(C01D)C CODE FOR STR TOKEN.

STRING(0010)M MODULE WITH THE STRING FUNCTIONS IN IT.

SUM (9620)E ROUTINE TO IMPLEMENT THE SUM FUNCTION.

SUMCOD(0019)C CODE FOR SUM TOKEN.

SWIERR(BC2E)E SWI ERROR - ROUTINE TO HANDLE SWI INTERRUPTS.

SYMTAB(B0D6)E SYMBOL TABLE ROUTINE, BUILDS AND SEARCHES THE SYMBOL TABLE.

SYSERR(BC3C)E SYSTEM ERROR - ROUTINE TO HANDLE THINGS THAT CAN NOT OCCUR.

```
T1      (003F)V     T1 - T4 ARE FP INTERNAL SCRATCH BYTES.
T2      (00C0)V     "
T3      (00C1)V     "
T4      (00C2)V     "

TABAD  (E94E)C      POINTER TO TABLE OF POWERS OF (2**(1/16))**M.
TABCNT (060B)V      CURRENT LOGICAL OUTPUT POSITION.
TABPNT (00C4)V      POINTER USED FOR VARIOUS FP OPERATIONS.
TABPTR (00A0)V      IOSCAN NAME TABLE POINTER.
TAGPTR (0088)V      IOSCAN STACK POINTER.
TAN15  (E9B2)C      FP CONSTANT = TAN(15 DEGREES).
TANCOD (004A)C      TOKEN CODE FOR TAN.
TAPE   (D9C0)E      ROUTINE TO PROCESS MARK, FIND COMMANDS.
TAPF2  (E12F)E      ROUTINE TO PROCESS LIST, SAVE, OLD COMMANDS.
TAPFIL (E129)E      SIMIALAR TO TAPF2.
TCOL   (00A2)V      TEMPORARY MATRIX COLUMN COUNTER FOR I/O.
TEMPX  (04F2)V      TEMP. STORAGE OF FPN FOR GRAPHICS.
TEMPY  (0502)V      TEMP. STORAGE OF FPN FOR GRAPHICS.
TESTCS (AA52)E      DOES A CONDITIONAL UPCASE DEPENDING ON SET CASE/NOCASE.
TFLGS1 (0035)V      TRANSLATOR FLAGS BYTE.
```

IMM	ASS	DEF FLG	STR	PARM	RAND	OF	

```
THECOD (0089)C      CODE FOR THE TOKEN.
TLICOD (007A)C      CODE FOR TLI TOKEN.
TLIST  (E207)E      ROUTINE TO PROCESS THE TLIST COMMAND.
TMP1   (00A5)V      TEMP. STORAGE FOR THE DISPLAY DRIVER. VECTOR INFORMATION
TMP2   (00A6)V      SIMILAR TO TMP1.
```

TMP3 (00A7)V SIMILAR TO TMP1.

TMPHX (00AA)V CURRENT POSITION OF X AXIS DISPLAY IN TEKPOINTS.

TMPHY (00A8)V CURRENT POSITION OF Y AXIS DISPLAY IN TEKPOINTS.

TMPLX (00AB)V THE LOWER 8 BITS OF TMPHX.

TMPLY (00A9)V THE LOWER 8 BITS OF TMPHY.

TO2COD(0087)C CODE FOR TO2 TOKEN.

TOCOD (0086)C CODE FOR TO TOKEN.

TPOINT(001C)V TABLE POINTER.

TRACE (C232)E ROUTINE TO IMPLEMENT SET TRACE COMMAND.

TRACOD(0098)C CODE FOR TRA TOKEN.

TRANSL(A8B0)E TRANSLATOR MAINLINE.

TRAPS (FFF8)C BASE ADDRESS FOR SYSTEM TRAPS.

TRASH (011E)V I/O SYSTEM SCRATCH VARIABLE.

TRIG (EA2C)E ROUTINES TO COMPUTE SIN, COS, AND TAN.

TRN (0021)C BANK SWITCH CONTROL CONSTANT.

TRNCOD(001A)C CODE FOR TRN TOKEN.

TRUTH (EBCB)E ROUTINE TO DETERMINE IF FP NUMBER IS CLOSER TO 0 OR 1.

TST8NF(FF8C)E ROUTINE TO TEST FOR 8NFR'S (PHYSICAL END OF FILE).

TSTINT(93C4)E TEST INTERRUPT - EVALUATOR ROUTINE TO TEST FOR RAISED ON CONDITIONS AND PENDING USER KEYS.

TTY2 (0592)V KEYBOARD TTY LOCK FLAG.

TYPARG(CE77)E TYPE OF ARGUMENT - ROUTINE TO TEST AND SIMPLIFY FUNCTION ARGUMENTS ON THE STACK.

TYPASC(DA2E)C POINTER TO "ASCII" HEADER.

TYPBIN(DA26)C POINTER TO "BINARY" HEADER.

TYPCOD(003E)C CODE FOR TYP TOKEN.

TYPE (CE00)E ROUTINE TO PROCESS THE TYPE COMMAND.

TYPFIL(E874)E ENTRY CALL TO FILE SYSTEM FOR TYP N.

TYPIN (8033)E LINE EDITOR MAIN ENTRY POINT.

TYPLST(DA36)C POINTER TO "LAST" HEADER.

TYPNEW(DA1E)C POINTER TO "NEW" HEADER.

TYPRES(CF77)E TYPE OF RESULT - ROUTINE TO TEST FOR VALID RESULT AREAS FOR SYSTEM FUNCTIONS.

TYPSEC(DA5C)C POINTER TO "SECRET" HEADER.

UNADR (AF10)E ROUTINE TO UNADDRESS THE I/O SYSTEM. RESETS DISPLAY AND I/O LIGHT AND I/O VARIABLES.

UNCOMP(98AA)E UNCOMPILER, CONVERTS POSTFIX INTERNAL NOTATION LINES TO PREFIX-INFIX INTERNAL NOTATION.

UNDER (B3BC)E SAME AS ZANS - PUSHES 0 ON STACK.

UNICOD(0075)C CODE FOR UNI TOKEN.

UNIT (080C)C UNIT COMMAND I/O SYSTEM CONTROL CONSTANT.

UNLEX (9B90)E UNLEXICALIZE, CONVERTS INTERNAL NOTATION INTO ASCII.

UP (EE6E)E ROUTINE TO COMPUTE FP A**B.

UPCASE(AA5A)E CONVERTS INPUT DATA TO UPPERCASE.

UPCOD (0010)C CODE FOR UP TOKEN.

UPDLEN(ADCE)E ROUTINE TO UPDATE A STRING LENGTH AFTER INPUT/READ.

USICOD(0091)C CODE FOR USI TOKEN.

USING (A84C)E ROUTINE TO SET UP POINTERS FOR PRINT USING COMMAND

USRORG(0047)V USER ORIGIN - THE ADDRESS OF THE FIRST FREE BYTE OF RAM. THIS VARIABLE SHOULD BE USED INSTEAD OF PGMEND BY SYSTEM MODULES.

UTILS (A994)M SYSTEM UTILITY ROUTINES.

V Symbols

VAL (0090)C	BANK SWITCH CONTROL CONSTANT.
VALCOD(001F)C	CODE FOR VAL TOKEN.
VALTG (000C)C	VALUE TAG - STACK TAG FOR A FLOATING POINT NUMBER.
VECTOR(CB31)E	ROUTINE TO DISPLAY VECTOPS ON THE DISPALY.
VIECOD(006C)C	CODE FOR VIE TOKEN.
VIEW (0010)C	BANK SWITCH CONTROL CONSTANT.
VLDKEY(05A6)V	KEYBOARD HANDLER TEMPORARY KEY REGISTER.

W Symbols

WAICOD(007F)C CODE FOR WAI TOKEN.

WAIT (C250)E ROUTINE TO IMPLEMENT WAIT COMMAND.

WAIT. (C6E6)E ROUTINE TO WAIT 40MSEC.

WBYCOD(0078)C CODE FOR WBY TOKEN.

WBYTE (F188)E ROUTINE TO HANDLE WBYTE COMMAND.

WBYTVL(F1E2)E ROUTINE TO SEND ONE BYTE FROM THE WBYTE LIST.

WINCOD(0068)C CODE FOR WIN TOKEN.

WINDOW(0018)C BANK SWITCH CONTROL CONSTANT.

WREND.(FE4E)E ROUTINE TO FINISH WRITING A MAGTAPE RECORD.

WRICOD(0068)C CODE FOR WRI TOKEN.

WRISTG(CD3D)E ROUTINE TO WRITE A STRING.

WRITE (0F21)C WRITE COMMAND I/O SYSTEM CONTROL CONSTANT.

WRIVAL(CD6F)E ROUTINE TO WRITE A VALUE.

WRRS (FD25)E MAGTAPE DRIVER ROUTINE TO WRITE A PHYSICAL RECORD.

X Symbols

X0	(00D3)V	X0 ... X5 IS A SIX BYTE ARRAY FOR HOLDING FRACTIONS DURING FLOATING POINT OPERATIONS.
X1	(00D2)V	" "
X2	(00D1)V	" "
X3	(00D0)V	" "
X4	(00CF)V	" "
X5	(00CE)V	" "
XEQSP	(0049)V	EXECUTION STACK POINTER - SECOND STACK FOR RETURN POINTS.
XEQSTK	(0424)V	EXECUTION STACK SPACE - WORK AREA FOR EXECUTION STACK.
XFNBNK	(0469)V	EXTENDED FUNCTION BANKS - FOUR VARIABLES THAT HOLD THE BANK ADDRESSES OF THE EXTENDED FUNCTION ROMS.
XFNMAP	(8800)M	MODULE THAT HAS THE BANK HEADER FOR OVERFLOW ROM PACK.
XFRCTL	(C431)E	ROUTINE TO ATTACH AN APPROPRIATE I/O DRIVER GIVEN A.PRIM AND A TABLE POINTER. NOTE: THIS ROUTINE WILL ALSO ATTACH I/O DRIVERS FROM THE PIATBL.
XFRSCN	(E07C)E	MAGTAPE CONTROL ROUTINE TO INITIALIZE MAGTAPE OPERATIONAL REQUESTS: LIKE PRINT, OLD.
XLAST	(04C2)V	X AXIS LAST POSITION FPN IN USER UNITS.
XMAXW	(0482)V	PRESENT XMAX WINDOW (FPN).
XMINS	(04A2)V	PRESENT XMIN VIEWPORT (FPN).
XMINW	(047A)V	PRESENT XMIN WINDOW (FPN).
XNEW	(04EA)V	NEW POSITION FOR PLOTTING IN USER UNITS (FPN).
XSF	(049A)V	PRESENT X AXIS SCALE FACTOR. XSF := (XMAX WINDOW - XMIN WINDOW) / (XMAX VIEWPORT - XMIN VIEWPORT)

Y Symbols

Y0	(00CD)V	Y0 ... Y5 IS A SIX BYTE REGISTER FOR HOLDING FRACTIONS DURING FLOATING POINT OPERATIONS.
Y1	(00CC)V	" "
Y2	(00CB)V	" "
Y3	(00CA)V	" "
Y4	(00C9)V	" "
Y5	(00C8)V	" "
YAXIS	(00A4)V	AXIS FLIP-FLOP FLAG.
YLAST	(04D2)V	SIMILAR TO XLAST.
YMAXW	(0492)V	SIMILAR TO XMAXW.
YMINS	(04B2)V	SIMILAR TO XMINS.
YMINW	(048A)V	SIMILAR TO XMINW.
YNEW	(04FA)V	SIMILAR TO XNEW.
YSF	(04CA)V	SIMILAR TO XSF.

Z Symbols

ZANS (B3BC)E ROUTINE TO PUT A FLOATING POINT 0.0 RESULT ON STACK.

ZDRAW (1420)C DRAW COMMAND I/O SYSTEM CONTROL CONSTANT.

ZERCNT(0606)V COMMON ZERO COUNT.

ZERO1 (0600)V LEFT TRAILING ZERO COUNT.

ZERO2 (0601)V RIGHT LEADING ZERO COUNT.

ZERO3 (0605)V RIGHT TRAILING ZERO COUNT.

ZERSVH(0607)V ZERO SAVE ADDRESS, HIGH.

ZERSVL(0608)V ZERO SAVE ADDRESS, LOW.

ZGIN (18A0)C GIN COMMAND I/O SYSTEM CONTROL CONSTANT.

ZIX (B3C2)E ROUTINE TO OUTPUT A FP ZERO ON STACK.

ZIX4 (B3CA)E ROUTINE TO ZERO TOP 4 WORDS OF STACK.

ZMOVE (1520)C MOVE COMMAND I/O SYSTEM CONTROL CONSTANT.

ZRDRAW(9420)C RDRAW COMMAND I/O SYSTEM CONTROL CONSTANT.

ZRMOVE(9520)C RMOVE COMMAND I/O SYSTEM CONTROL CONSTANT.

ZX (0041)V ZERO X - A VARIABLE CONTAINING ZERO THE CLEAR X WITH

TABLE OF CONTENTS

INTRODUCTION

This document provides information pertaining to the 4051 system firmware. The following is a list of the parts.

1) Routine documentation--this is a package containing documentation and interface specifications on routines within the 4051 which may be of use to potential ROM writers.

2) I/O System variable definitions--this is an appendix to the routine documentation. This appendix defines the I/O system variables which must be used for most of the I/O routines. This appendix is referred to in the documentation as the normal I/O system variables.

ABS

FUNCTION: ABSOLUTE VALUE

INPUT: FP NUMBER ON TOP OF STACK.

OUTPUT: FP NUMBER ON TOP OF STACK.

QACS/QASN/QATN

FUNCTION: COMPUTES INVERSE TRIG FUNCTIONS

INPUT: INPUT IS IN STACK

OUTPUT: ANSWER ON STACK.
ERRCD CONTAINS ERROR CODE.

USES: R1,R1+1
TABPNT

CALLS: PSHRET
DOFP
SQR
ZANS (IN NORM)
FPDIV
CMPFPN
FPNEG
PSHFPN
FPADD
ABX
FPMUL
RTRN

NOTES: ASIN AND ACOS ARE COMPUTED FROM

ASN=ATAN(X/SQR(1-X*X))

ATN IS COMPUTED BY REDUCING X TO ARCTAN (15 DEGREES) OR LESS AND THEN APPLYING A CONTINUED FRACTION APPROXIMATION.

ADRDEV

FUNCTION: THIS ROUTINE ASSIGNS PRIMARY AND SECONDARY ADDRESSES AND IN GENERAL SETS UP THE I/O SYSTEM FOR THE CURRENT I/O REQUESTED BY BASIC.

INPUTS: 1)OPRADR - 2 BYTES; A CONSTANT THAT IS SET UP BY THE EVALUATOR FOR EVERY I/O STATEMENT EXCEPT POLL, RBYTE AND WBYTE. THIS CONSTANT IS DIFFERENT FOR EVERY I/O STATEMENT AND ITS VALUE MAY BE FOUND IN IOKONS.

2)OPTIONAL ADDRESS DATA IS STORED ON THE STACK SOMEWHERE AND IS MARKED BY AN ATSNTG.

3)A,STAT - ATVLD BIT 2 <4,> IS SET IF OPTIONAL ADDRESS INFORMATION IS ON THE STACK.

4)PPMODE - FLAG FOR PERCENT MODE DELIMITERS.
PPNUL : PERCENT MODE NULL CHARACTER.
PPEOR : PERCENT MODE END-OF-LINE CHARACTER.
PPEOF : PERCENT MODE EOF CHARACTER.
IECRLF : OPTIONAL CR/LF DELIMITER MODE.

OUTPUTS : IOFUNC : SET TO CODE TO INDICATE PRESENT I/O FUNCTION.
A, STAT : SET APPROPRIATELY - SEE IOEQU FOR DEFINITION OF BITS.
ERRCD : SET IF ILLEGAL ADDRESS
A,PRIM : PRIMARY ADDRESS
A,SEC : SECONDARY ADDRESS
TABCNT =1 : COUNTER FOR PRINTF
IOSYSF =1 : I/O SYSTEM FLAG FOR FIRST ACCESS - GENERALLY; USED BY ROM PACKS (SPECIFICALLY FILE SYSTEM)
NULCHR : NULL CHARACTER - NORMALLY 255.
EOLCHR : END-OF-LINE CHARACTER NORMALLY <CR>
ETXCHR : EOF CHARACTER NORMALLY 255.
POINTR =65535 : FOR ON FULL
DR =65535 : FOR ON FULL
BLINK =0 : SO SCREEN I/O WILL WORK
NOOUT =0
CRSTAT =0 : INITIALIZE BUFFER FLAGS.

CALLS : ATPROC
MTPADR

NOTE : WILL ALSO TURN ON THE I/O LIGHT UNLESS PRIMARY ADDRESS IS 32, <DISPLAY> OR 34, <DATA>.

AFPITT

FUNCTION : CONVERTS AN ASCII STRING TO AN FPN.

INPUTS : R0 : POINTER TO FIRST CHARACTER OF STRING
R1 : POINTER TO DELIMITER OR ONE PAST LAST VALID CHARACTER.

OUTPUTS : FPC : RESULTANT NUMBER
R0 : POINTER TO DELIMITER USED
R1 : UN-MODIFIED

USES : POINT
CRSTAT

CALLS : PSHFPN
PULFPN
INPVAL

A(5-16)X AND D(5-13)X

FUNCTION : THESE ROUTINES ARE USED TO ADD OR SUBTRACT SMALL INTEGERS TO THE INDEX REGISTER. THE NUMBER IN THE NAME CALLED IS THE NUMBER ADDED TO IX.

INPUTS : X HAS ANY NUMBER.

OUTPUTS : IX IS ALTERED.

ASCFPN

FUNCTION : CONVERT AN INPUT STRING OF ASCII CHARACTERS INTO A FLOATING POINT NUMBER.
FORMAT IS:

(+/-)(DIGS)(.)(DIGS)(UPPER CASE E OR LOWER CASE E)(+/-)(DIGS)
OVERFLOW CAUSES LARGEST POSSIBLE NUMBER TO BE OUTPUT; NO ERROR OCCURS. UNDERFLOW CAUSES ZERO TO BE OUTPUT. IF TOO MANY DIGITS ARE SPECIFIED IN THE FRACTION THEY ARE IGNORED.

INPUTS : FIRST CHARACTER ASSUMED IN GLOBAL CHAR.
CALLS GETCHR TO PUT NEXT CHARACTER IN CHAR.

OUTPUTS : THE FLOATING POINT NUMBER IS PLACED ON THE STACK.
SETS SYSTEM FLAGS:

DIGFLG : 0 = DIGIT SEEN, <> 0 = NO DIGIT SEEN
ITSINT : 0 = INPUT NOT AN INTEGER <> 0 = INPUT A NON-NEGATIVE INTEGER
TFLGS1: SEE NOTES
EFLAG : <> 0 = ONLY AN E SEEN, 0 = ANYTHING ELSE
SIGNS : HIGH ORDER BIT = FRACTION NEGATIVE, LOW ORDER BIT = EXPONENT NEGATIVE.
ERRCD : IS CLEARED WHEN OUTPUT IS NORMAL.
EQU : <> 0 = NOT AN INTEGER

USES : CNT
X0...X5
T1...T4
DEXP,DEXP+1
PLTAB, THE TABLE OF POWERS OF 10
TABPNT

CALLS : GETCHR
PSHRET
TMULT
PSHFPN
NORMR (=PSHRET + NORM)
MAXANS (IN NORM)
ZANS (IN NORM)
FPDIV
FPMUL
A8X

NOTES: IF CNT=1 THEN IF NUMBER IS AN INTEGER THEN CLEAR BITS 2 AND 7 OF TFLGS1 ELSE SET EQU=1

BACKUP

FUNCTION : THIS ROUTINE USES R0 AS A PSEUDO STACK POINTER AND BACKS IT UP ONE STACK ENTRY. R0 HAS THE ADDRESS OF A VALUE TAG+1 FOR INPUT. IF THE BYTE IS NOT A LEGAL TAG SYSERR IS CALLED. IF THE TAG IS THE END OF STACK TAG R0 IS NOT ALTERED.

INPUTS : R0 IS THE ADDRESS OF A STACK TAG+1

OUTPUTS : R0 IS UPDATED TO REFLECT THE STACK ENTRY IS PASSED.

USES : R0

BELCAL

FUNCTION : BEEP THE 4051 BELL

BFRALC

FUNCTION : INITIALIZE BUFFER POINTERS BASED ON PRIMARY ADDRESS.

INPUTS : A,PRIM : PRIMARY ADDRESS
MTSREG : TO DETERMINE IF 256 OR 128 BYTE MAGTAPE RECORDS.
A,STAT : TO DETERMINE IF INPUT OR OUTPUT.

OUTPUTS : A,STRT : POINTER TO FIRST SLOT
A,END = A,STRT+1
NOTE :A,STRT + 1 IS ALSO CLEARED, THIS IS NECESSARY FOR PUTBYT
A,MAX : POINTER TO LAST SLOT.

CALLS : LDXX

BININ

FUNCTION : ATTACHES PROPER I/O DEVICE FOR BINARY INPUT.

INPUT : NONE

OUTPUT : ERRCD SET IF ERROR IN HANDLER.

USES : NOTHING

CALLS : XFRCTL
DATIN
FILEIN
IECIN
MTPIN
NIOPEV

NOTES : ALL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

BINOUT

FUNCTION : ATTACHES PROPER I/O DEVICE FOR BINARY OUTPUT.

INPUT : NONE

OUTPUT : ERRCD SET IF ERROR IN HANDLER.

USES : NOTHING

CALLS : XFRCTL
FILEOT
IECOUT
MTPOUT
NIODEV

NOTES : ALL I/O SYSTEM VARIABLES MUST BE SET CORRECTLY.

CMPFPN

FUNCTION: COMPARES TWO FLOATING POINT NUMBERS.

INPUTS: A POSITIVE FLOATING POINT NUMBER (Y) POINTED TO BY TABPNT. A FLOATING POINT NUMBER (X) ON THE STACK. THE STACK CONTAINS TWO BYTES ON TOP FOLLOWED BY THE TAG FOR X AND THEN X ITSELF.

X AND Y ARE NOT CHANGED.

OUTPUTS: ACC A = 0 IF X = Y
= 1 IF X > Y
= -1 IF X < Y

COMPR

FUNCTION: COLLECTS ALL STILL VALID STRINGS, ARRAYS, AND PROGRAM LINES AS FAR DOWN (TOWARD APPR. 0) IN MEMORY AS POSSIBLE, OR TO SAY THAT ANOTHER WAY COLLECTS ALL AVAILABLE SPACE BETWEEN PGMEND AND LSP AND MOVES IT ABOVE LSP INTO THE FREE SPACE AREA.

INPUTS: ITEM 1 ON STACK >=0 AND <=32767. THIS IS THE NUMBER OF BYTES WHICH MUST BE RECOVERED FOR THE COMPRESS TO BE CONSIDERED SUCCESSFUL. IF LESS THAN THIS ARE RECOVERED AN ERROR - ERWSFL - IS GENERATED.

OUTPUTS: NONE (FREE SPACE\)

USES: R0,R2-R6

RESTRICTIONS, SUGGESTIONS, AND NOTES: 1) ALL ENTRIES ON THE STACK MUST BE CORRECTLY TAGGED WHEN THIS ROUTINE IS CALLED.

2) THIS ROUTINE DOES NOT EXPECT TO FIND (IE WILL NOT HANDLE CORRECTLY) A POINTER TO STRING COUNT - PSCTG - ON THE STACK.

CROS

FUNCTION : TO PLACE ON THE CRT SCREEN THE GRAPHIC POINTER IN RESPONSE TO THE IMPUT FROM THE JOY STICK AND TO TERMINATE THE PROCESS WHEN A LEGAL CHARACTER IS INPUT FROM THE KEYBOARD.

INPUTS : ANALOG VOLTAGE FROM JOY STICK AND TERMINATING CHARACTER FROM KEYBOARD.

OUTPUTS : THE VALUE OF THE CRT DA"S AT THE TIME A LEGAL KEYBOARD CHARACTER IS INPUT, IN TMPHY, TMPLY, TMPHX, TMPLX, AND THE KEYBOARD CHARACTER INPUT IN THE "A" REGISTER ON EXIT.

USES : NO PSEUDO REGISTERS

CALLS : GENCUR, KBOOUT

CRTRST

FUNCTION : RESET THE CRT PIA"S AND RESET THE OLD STATUS OF THE DISPLAY AS WELL AS SERVICING ANY PENDING PAGE, COPY, HOME, REWIND REQUESTS. GENERALLY CALLED TO RESTORE CRT OPERATION AFTER THE MAGTAPE HAS BEEN USED.

CALLS : DSPENL - SERVICES PENDING REQUESTS
ENABLE - TO OFFSET THE DSABLE OF THE MAGTAPE SECTION.

CTLCHR

FUNCTION : THIS ROUTINE SENDS CHARACTERS TO THE CRT BUT WILL WAIT UNTIL THE PAGE IS CLEARED IF THE DISPLAY IS IN FULL STATUS.

CTLCHR - SENDS ALL CHARACTERS TO THE SCREEN

INPUT : ACC, A - ASCII CHARACTER

CALLS : SETBNK
PCHAR
DSPCPY
FULSCN

NOTES : THIS ROUTINE DOES SOME SPECIAL HANDLING FOR THE OPTIONAL DISPLAY HANDLERS LIKE THE OPTION 1 INTERFACE THRU THE 3 BYTE DSPVCT VARIABLE. IT ALSO HANDLES THE SPECIAL PAGE FULL MODES AS WELL AS THE ON FULL CONDITION.

DEFPNT

FUNCTION: TO OUTPUT STRING OR NUMBERS ACCORDING TO THE DEFAULT IMAGE STRINGS CONTAINED WITHIN DEFPNT.

DEFAULT STRINGS ARE AS FOLLOWS:

1. STRINGS (DT=0)

FA

2. FOR NUMBERS (DT=1) IN THE RANGE OF 10E-3<|N|<10E-7 THE DEFAULT IMAGE STRING IS -

FD.FD

FOR ALL OTHER NUMBERS, THE DEFAULT IMAGE STRING IS -

FE

INPUTS: DT, DP, DI

OUTPUTS: OUTPUTS FORMATTED DATA THROUGH PUTBYT IN ACCORDANCE WITH ABOVE INPUTS.

USES: R0,R3

CALLS: UPCASE, PUTBYT

NOTES: THE CALLER OF DEFPNT MUST INSURE THAT THE INPUTS STAY VALID DURING THE DATA OUTPUT SEQUENCE.

DIM

FUNCTION : DIM IS CALLED AS A RESULT OF THE DIMENSION TOKEN. ITS FUNCTION IS TO CLEAR THE BRACKET COUNT AND SET THE DIMENSION FLAG IN LOCAL FLAG TO 1.

INPUTS : NONE

OUTPUTS : BRKCNT = 0
LCLFLG = LCLFLG + DIMFLG

USES : <-

CALLS : <-

NOTES : <-

DIMSTR

FUNCTION : DIMSTR IS USED TO DIMENSION A STRING VARIABLE.

INPUTS : INT1 MUST HAVE THE LENGTH TO BE DIMENSIONED. "X" REGISTER MUST HAVE VARIABLE NAME TABLE POINTER.

OUTPUTS : MEMORY IS ALLOCATED BY CREATING A NEW DATA ENTRY AS PER THE LENGTH VALUE. THE STRING PARAMETERS ARE SET IN THE NAME TABLE.

USES : R7

CALLS : COMPR, DISBLE, ENABLE

NOTES/ERRORS : PART OF THE ALLOCATION ROUTINE MAY INVOLVE CONSIDERABLE TIME SINCE THE MEMORY COMPRESS ROUTINE MAY BE INVOLKED. AN ERROR WILL RESULT IF THERE IS NOT ENOUGH MEMORY TO DO THE DIMENSIONING.

DISBLE

FUNCTION : THIS ROUTINE IS USED TO INCREMENT THE SYSTEM INTERRUPT MASK COUNTER TO DISABLE SYSTEM LEVEL INTERRUPTS. THIS IS USED TO PROTECT FUNCTIONS LIKE MEMORY COMPRESS FROM BEING INTERRUPTED FOR AN ABORT OPERATION. IF A FUNCTION IS LEAVING THE SYSTEM DISABLED FOR LONG PERIODS OF TIME THE ROUTINE SAFE SHOULD BE CALLED TO LET ROM PACKS PROCESS INTERRUPTS.

RESTRICTIONS, NOTES: SEE ALSO ENABLE AND SAFE.

DOFP

```
FUNCTION :      FLOATING POINT INTERPRETER

INPUT :         A TYPICAL CALL IS:

                JSR DOFP
                .BYTE FOP1 + FAR1
                .BYTE FOP2 + FAR2
                .WORD ....
                .BYTE FRET
                FOR EACH OPERATOR BYTE, DOFP PERFORMS FIRST THE 5-BIT
                OPERATION FOPX, THEN THE 3-BIT OPERATION FARX,.WORD"S
                OF DATA MAY BE PLACED AFTER SOME FOP"S.  THE LAST FOP
                IS FRET.  INDIVIDUAL FOP"S ARE DESCRIBED BELOW.  THE
                FAR MAY BE:

                FA (CALL FPADD)
                FS (CALL FPSUB)
                FM (CALL FPMUL)
                FD (CALL FPDIV)

OUTPUT:         THE STACK IS MANIPULATED ACCORDING TO THE FOP"S.
                CONTROL RETURNS TO THE BYTE AFTER FRET.

USES :          R0,R0+1
                T4
                FPA
                FPB
                FPC

CALLS :         PSHFPN
                PULFPN
                A8X
                FPMUL
                FPDIV
                FPADD
                FPSUB
                JMPAX

FOP"S :         1. FHE: PUSH EXTENDED.
                .BYTE FHEX + FAR
                .WORD OPAD
                THE FPN AT THE ADDRESS GIVEN IN THE .WORD (E.G.,OPAD)
                IS PUSHED ON THE STACK.

                2. FHIN: PUSH INDIRECT
                SAME AS FHEX EXCEPT OPAD CONTAINS THE ADDRESS OF THE FP
                NUMBER RATHER THAN THE NUMBER.

                3. FRET: RETURN.

                4. FLEX: PULL EXTENDED.
```

DOFP (continued)

SAME AS FHEX EXCEPT THE FP NUMBER IS POPPED FROM THE STACK TO OPAD.

5. FSWP: SWAP TWO STACK ENTRIES.
THE TOP TWO STACK ENTRIES ARE SWAPPED.

6. FLIN: PULL INDIRECT.
SAME AS FHIN EXCEPT THE FP NUMBER IS POPPED FROM THE STACK TO THE ADDRESS IN OPAD.

7. DUPLICATE STACK ENTRY.
THE TOP STACK ENTRY IS DUPLICATED.

8. FSAV: SAVE IN TEMPORARY.
THE TOP OF THE STACK IS POPPED TO THE GLOBAL FPA.

9. FRES: RESTORE FROM TEMPORARY.
THE FP NUMBER IN FPA IS COPIED TO THE TOP OF THE STACK.

10. FART: NO OP.
USED WHEN ONLY THE FAR IS TO BE PERFORMED.

11. FHIM: PUSH IMMEDIATE.
THE OPERAND IMMEDIATELY FOLLOWING IN THE .WORD IS TO BE PUSHED.

DRBUSY

FUNCTION : WAIT UNTIL DISPLAY IS IN A READY STATE. ALL COPIES, PAGES, VECTORS DONE.

NOTES : THIS ROUTINE MUST BE CALLED BEFORE ANY NEW COMMANDS ARE GIVEN TO THE DISPLAY. THIS IS NORMALLY DONE BY CRT SERVICE ROUTINES BUT CHECK THE ROUTINES BEFORE ASSUMING ANYTHING.

DSMOVE/DSDRAW/DSGRAF

FUNCTION: CONVERTS NUMBERS FROM GDU"S TO TEKPOINT"S (10 BIT DAC INFO.) AND THEN DISPLAYS A LIGHT OR DARK VECTOR TO THAT POINT.

INPUTS: TMP1 : 0 - DRAW <DSDRAW SETS IT>
1 - MOVE <DSMOVE SETS IT>
POINT : POINTS TO FLOATING POINT GDU VALUE.

USES : YAXIS - AXIS FLIP-FLOP CONTROL.
TMP2, TMP3
TMPHX, TMPLX
TMPHY, TMPLY

CALLS : PSHFPN, A9X
FIX 1
VECTOR

NOTES : THIS ROUTINE IS CALLED TWICE FOR 1 VECTOR, THAT IS ONCE FOR EACH AXIS (YAXIS KEEPS TRACK). IT ALSO DROPS VECTORS THAT GO BEYOND SCREEN EDGES.

DSPAGE/DSHOME

FUNCTION: DSPAGE - PAGES/HOMES CRT
DSHOME - HOMES CRT

OUTPUT: CLEARS FULL

USES: TMP1
TMPHY,TMPHX
TMPLY,TMPLX

CALLS: MOVE (ROUTINE INTERNAL TO CRTDRV)
DELY1S

DSPCPY/DSPCP2

FUNCTION: ISSUES A DISPLAY COPY COMMAND TO THE HARD COPY CONNECTOR.

CALLS: DRBUSY
WAIT.

NOTES: DSPCPY CHECKS TO SEE IF THE SCREEN IS ENABLED, IF IT IS NOT IT WILL SET THE COPY PENDING BIT.

DSPCHR

FUNCTION: SENDS ASCII CHARACTERS TO THE SCREEN AS IN CTLCHR EXCEPT IT CONVERTS ALL CONTROL CHARACTERS EXCEPT <CR> TO THE READABLE BACKSPACE UNDERLINE FORM.

INPUT: ACC. A = ASCII CHARACTER

CALLS: CTLCHR

DSPGIN

FUNCTION: PERFORM INPUT FUNCTIONS FROM THE DISPLAY AS DETERMINED BY THE SECONDARY ADDRESS.

INPUTS: A,SEC : CURRENT SEC. ADDR.
24. = GIN <= RETURN CURRENT POSITION IN GDU'S
13. = INPUT <= RETURN SCREEN SIZE IN GDU'S.

YAXIS : AXIS FLIP-FLOP
0 = RETURN X
<>0 = RETURN Y
NOTE: TOGGLE HANDLED BY DSPGIN ROUTINE.

POINT : POINTER TO RESULTING VALUE.

OUTPUTS: VALUE STORED AT POINT.

USES: YAXIS
TMPHX, TMPLX : READ TO GET CURRENT POSITION
TMPHY, TMPLY

CALLS: NIODEV
FLOAT 1
DOFP
PULFPN

NOTE: ONLY I/O VARIABLE USED IS A,SEC.

DSPOUT

FUNCTION: PERFORMS THE BASIC DISPLAY FUNCTIONS AS DICTATED BY THE CURRENT SECONDARY ADDRESS AND A BUFFER FULL OF INFORMATION.

INPUT: A,SEC : DEFINES WHAT IS TO BE DONE TO THE CHARACTERS IN THE BUFFER.

A,PTR : POINTER TO THE FIRST VALID CHARACTER IN THE BUFFER.

A,END : POINTER TO THE LAST VALID CHARACTER.

USES: A,PTR : MOVING POINTER.

CALLS: PCHAR
CTLCHR
DSPCPY
DOIT

ENABLE

FUNCTION: THIS ROUTINE IS USED TO CONTROL SYSTEM INTERRUPTS. THE ROUTINE WILL DECREMENT THE SYSTEM INTERRUPT COUNT AND IF IT BECOMES ZERO, SYSTEM INTERRUPTS CAN BE PROCESSED. THE SYSTEM MASK IS NOT ZERO IF OPERATIONS ARE IN PROGRESS THAT CAN NOT BE INTERRUPTED SAFELY. THESE OPERATIONS HAVE SYSTEM POINTERS OR CONTROL TABLES IN A STATE THAT IS NOT CONTROLLED.

RESTRICTIONS, NOTES: THIS ROUTINE CAN CALL ANY INTERRUPT PROCESSOR SO THE VARIABLE LIST AND ROUTINE LIST CAN NOT BE GENERATED. SEE ALSO, DISABLE AND SAFE.

ETOX/ZZETOX

FUNCTION: EXPONENTIAL EXP(X)

INPUT: X ON STACK

OUTPUT: EXP(X) ON STACK

IF EXP(X) OVERFLOWS OR UNDERFLOWS IT IS REPLACED BY THE LARGEST POSITIVE NUMBER OR ZERO, RESPECTIVELY, AND EREXP IS PLACED IN ERRCD.

USES:
R3
R1
R1+1
R3+1
TABPNT

CALLS:
PSHRET
PSHFPN
FPMUL
ZANS
MAXANS
SLN
NORMR
DOFP
FPNEG
FPDIV
RTRN

METHOD: MULTIPLIES X BY LOG 2 (E) THEN PARTITIONS

/X/ = N + (M+Y) / 16)

WHERE N AND M ARE INTEGERS. 2**N IS EASY, 2**(M/16) IS BY TABLE LOOKUP IN TABAD, AND 2**(Y/16) IS COMPLETED BY A RATIONAL APPROXIMATION OF THE FORM.

2**X = (Q(X**2) + XP(X**2)) / (Q(X**2) - XP(X**2))

FOR 0<= X <= 1/16, NO SOURCE IS GIVEN FOR THE APPROXIMATION; P AND Q ARE BOTH OF DEGREE 1.

FIX1

FUNCTION: FIX FLOATING POINT TO 16 BIT INTEGER.

IF INPUT IS NEGATIVE, FIX (ABS(A)) IS COMPUTED AND ERFIXN IS SET IN ERRCD. IF THE INPUT EXCEEDS 2/16-1, THE LARGEST POSITIVE INTEGER IS RETURNED AND ERFIXOV IS SET IN ERRCD. IF THE INPUT IS LESS THAN 0.5, ZERO IS RETURNED. OTHERWISE THE INPUT IS ROUNDED TO THE NEAREST UNSIGNED 16 BIT INTEGER.

INPUT: A FLOATING POINT NUMBER. ON ENTRY, IX POINTS TO THE TAG BYTE FOR THAT FP NUMBER.

OUTPUT: THE 16 BIT INTEGER IS PLACED IN THE HIGH ORDER BYTES OF THE INPUT MANTISSA, IX+3 AND IX+4.

FLOAT1 ALIAS FLOAT2

FUNCTION: FLOAT A 16 BIT INTEGER.

AN UNNUMERALIZED FP NUMBER WITH EXPONENT 48 IS FORMED, THEN NORM IS CALLED.

INPUT: TAGGED 2 BYTE INTEGER ON STACK

OUTPUT: TAGGED FP RESULT IN STACK

CALLS: PSHRET, NORM

FMTCLN

FUNCTION: TO OUTPUT ANY REMAINING DATA THAT IS SPECIFIED IN THE IMAGE STRING AFTER ALL LIST OUPUT IS COMPLETE.

INPUTS: ISP, ISL

OUTPUTS: SEE FUNCTION

USES: R0-R3

CALLS: UPCASE, PUTBYT

NOTES: THE CALLER OF FMTCLN MUST INSURE THAT THE INPUTS REMAIN VALID DURING THE DATA OUTPUT SEQUENCE.

FMTINI

FUNCTION: TO INITIALIZE THE FORMAT OUTPUT PROCESS BY CLEARING ALL FLAGS, SET CURSER = TO FIRST CHARACTER.

INPUTS: ISL

OUTPUTS: SEE FUNCTION

USES: <-

CALLS: <-

NOTES: <-

FMTPNT

FUNCTION: TO OUTPUT NUMERIC AND STRING DATA TYPES ACCORDING TO THE IMAGE STRING AND ALSO TO OUTPUT DATA AS SPECIFIED WITHIN THE IMAGE STRING ITSELF.

INPUTS: ISP, ISL, DP, DL, DT

OUTPUTS: OUTPUTS FORMATTED DATA THROUGH PUTBYT IN ACCORDANCE WITH ABOVE INPUTS.

USES: R0 - R3

CALLS: UPCASE, PUTBYT

NOTES: THE CALLER OF FMTPNT MUST INSURE THAT THE INPUTS STAY VALID DURING THE DATA OUTPUT SEQUENCE.

FPADD

FUNCTION: FP ADD

TWO FLOATING POINT OPERANDS A AND B ARE ON THE STACK ALONG WITH THEIR TAGS, B IS ON TOP, A IS NEXT. THE RESULT R:=A+B REPLACES THEM ON THE STACK. NO GUARD BITS ARE USED. UNDERFLOWS AND OVERFLOWS ARE HANDLED IN THE STANDARD NORM ROUTINE.

INPUT: 2FP OPERANDS IN STACK

OUTPUT: ONE FP RESULT IN STACK

USES: X0 TO SAVE SIGN (B)

CALLS: PSHRET, RTRN
A9X
SHMR (RIGHT SHIFT MANTISSA)
FRTST (PART OF NORM)
NORM

FPCMP

FUNCTION: COMPARE TWO FP NUMBERS ON STACK.

INPUT: X, ON TOP OF STACK.
Y, NEXT ON STACK.

OUTPUT: IN ACCA:
-1 IF X > Y
+1 IF Y > X
0 IF X = Y

USES: TABPNT

CALLS: PSHRET
A9X=1
CMPFPN
RTRN

FPDIV

FUNCTION: FP DIVIDE

COMPUTES FP QUOTIENT OF 2 FP NUMBERS IN THE STACK. THE RESULT R:=A/B REPLACES A AND B. OVERFLOW, UNDERFLOW AND DIVIDE BY ZERO ARE HANDLED IN NORM. NO GUARD DIGITS USED.

INPUT: B, ON TOP OF STACK, WITH TAG
A, NEXT IN STACK, WITH TAG

OUTPUT: R, ON TOP OF STACK

USES: T1, T2, X0,..X5, Y0,...Y5

CALLS: PSHRET
A9X
MAXANS (IN NORM)
FPRET
NORM

FPMUL

FUNCTION: FP MULTIPLY

FORMS FP PRODUCT OF TWO FP NUMBERS ON THE STACK, THE RESULT R:=A*B REPLACES A AND B. NO GUARD DIGITS ARE USED. UNDERFLOWS AND OVERFLOWS ARE HANDLED BY NORM.

INPUT: B, ON TOP OF STACK, WITH TAG
A, NEXT ON STACK, WITH TAG

OUTPUT: R, RESULT ON TOP OF STACK

USES: T1, T2, X0...X5, Y0,...,Y5

CALLS: PSHRET
A9X
ZANS (IN NORM)
NORM

FPNASC

FUNCTION: REDUCE A FLOATING PRINT NUMBER X TO A FRACTION F AND AN EXPONENT E SUCH THAT X = F * 10^E AND 1/10<=F<1.

INPUTS: THE FP NUMBER IS ON THE STACK. THE TABLES OF POWERS OF 10,
PLTAB AND NEGTAB, ARE USED.

OUTPUTS: THE FP FRACTION MANTISSA, NOT NUMERALIZED IF 1/10<=F <1/2, IS IN X5...X5+5(X0).

THE EXPONENT E IS IN DEXP AND DEXP+1. IF X<0 THEN N5:="-"ELSE N5:=" ".

USES: X0...X5,Y1,Y0
N12, T4
TABPNT, MULPNT

CALLS: PSHRET
A9X
RTPN
CMPFPN
FPMUL
A8X
PULFPN
SHMR

FPNEG

FUNCTION: NEGATE A FLOATING POINT NUMBER: G = -F

INPUT: A FLOATING POINT NUMBER F ON TOP OF THE STACK.

OUTPUT: A FLOATING POINT NUMBER G = -F REPLACES F ON THE STACK.

CALLS: NONE

FPSUB

FUNCTION: FP SUBTRACT

REVERSES SIGN OF FP OPERAND ON TOP OF STACK AND GOES TO FPADD.

CALLS: FPADD

FUZZIE

FUNCTION: FUZZY COMPARE.

INPUTS: FP NUMBER B ON TOP OF STACK.
FP NUMBER A NEXT ON STACK.
FUZB, A GLOBAL FP NUMBER WITH THE ZERO-COMPARE FUZZY EXPONENT.

FUZA, A GLOBAL BYTE WITH THE NON-ZERO COMPARE FUZZ VALUE.

OUTPUTS: A AND B ARE DESTROYED.
A FP ZERO IS PLACED ON THE STACK.
ACC A AND T4 SET TO:

0 IF B FUZZY EQUAL A
+1 IF A > B
-1 IF A < B

USES: T4, TARPNT, FUZP

CALLS: FPSUB
CMPFPN
ZANS (IN NORM)
PSHRET

NOTES: FUZZY EQUAL OF NON-ZERO A TO 0.0 IS DEFINED BY COMPARING A TO A NUMBER WHOSE EXPONENT IS FUZB. IF A <= THAT NUMBER, THEN FUZZY EQUAL IS TRUE.

FUZZY EQUAL BETWEEN NON-ZERO A AND B IS DEFINED BY COMPUTING A-B AND COUNTING THE NUMBER OF NORMALIZE SHIFTS REQUIRED. IF THAT NUMBER IS >= FUZA, THEN FUZZY EQUAL IS TRUE.

GENCUR/DOIT

FUNCTION: GENERATES A WRITE-THRU CURSOR.

INPUT: CURSOR : ASCII CHARACTER TO FLASH.

USES: DISCNT - COUNTER

BLINK - TOGGLED TO GET BLINK ACTION. WHILE BLINK = 0, JUST IDLING FOR A WHILE.

CALLS: PCHAR

GETCHR

FUNCTION: EXTRACT ONE CHARACTER FROM THE CURRENT I/O DEVICE BUFFER. ALL DEVICES ARE HANDLED ON A BUFFERED BASIS. IF THE BUFFER IS EMPTY THEN A NEW BUFFER WILL BE REQUESTED FROM THE I/O DEVICE. A BUFFER REQUEST IS SATISFIED WHEN THE BUFFER BECOMES FULL OR AN END OF LINE CHARACTER IS ENCOUNTERED.

INPUTS: A. STAT : BFRSTT - BIT 5 <32.> SET IF BUFFER VALID (CONTAINS DATA).

A.PTR : POINTER TO NEXT VALID CHARACTER

A.END : POINTER TO FIRST NON-CHARACTER.

CRSTAT : BUFFER STATUS FLAGS AS SET BY I/O HANDLERS.

OUTPUTS: CHAR : CHARACTER READ
ACC.A : CHARACTER READ.
A.PTR : UPDATED
A.STAT : UPDATED.
CRSTAT : UPDATED

IF READ A DELIMITER CHARACTER THE CRVLD BIT WILL BE TURNED ON IN CRSTAT AND A <CR> WILL BE RETURNED AS THE CHARACTER REQUESTED.

ERRCD : SET IF ERROR IN I/O HANDLERS.

CALLS: NEWBFR

NOTE: ALL I/O SYSTEM VARIABLES ARE ASSUMED TO BE CORRECT.

IECIN

FUNCTION: TO SEND A BUFFER OF INFORMATION ALONG THE GPIB. IT WILL ALSO SEND MTA AND MSA IF REQUIRED.

INPUTS: A.STAT : BUSACT = BIT 4 <16.>
0 = SEND MTA AND MSA THEN SET BUSACT.
1 = ASSUME BUS SET UP.

A.PRIM : PRIMARY ADDRESS (MTA)

A.SEC : SECONDARY ADDRESS (MSA)

A.PTR : POINTER TO FIRST AVAILABLE CHARACTER POSITION IN BUFFER

A.MAX : POINTER TO LAST AVAILABLE CHARACTER POSITION IN BUFFER.

IOFUNC : CODE FOR PRESENT I/O FUNCTION, IF = READ <14.> THEN NO SEARCH FOR DELIMITERS WILL BE PERFORMED.

ETXCHR : 8 BIT END-OF-TEXT (OR EOF) CHARACTER. IF THIS CHARACTER IS DETECTED THE ETX BIT IN CRSTAT WILL BE SET AND INPUT FROM THE BUS WILL BE TERMINATED.

EOLCHR : 8 BIT END-OF-LINE CHARACTER. IF THIS CHARACTER IS DETECTED THE CRNORM BIT IN CRSTAT WILL BE SET AND INPUT FROM THE BUS WILL BE TERMINTED, AT LEAST UNTIL THIS ROUTINE IS CALLED AGAIN.

NULCHR : 7 BIT NULL CHARACTER. IF NULCHR >= 128. THEN NO CHARACTER WILL BE NULLED, OTHERWISE IF THE NULCHR IS DETECTED THE CHARACTER WILL NOT BE ENTERED INTO THE BUFFER.

CRSTAT : BUFFER STATUS FLAGS MUST BE 0 BEFORE CALLING THIS ROUTINE.

OUTPUTS: CRSTAT : BUFFER TERMINATION STATUS FLAG. POSSIBLE VALUES ARE ENUMERATED:

0 = BUFFER IS FULL AND NO TERMINATION CHARACTER WAS RECEIVED.

CREOI = AN EOI WAS RECEIVED
CRNORM = AN EOLCHR WAS RECEIVED
CRETX = AN ETXCHR WAS RECEIVED.

A.END : POINTER TO 1 PAST LAST VALID DATA CHARACTER.

NOTE: EOLCHR AND ETXCHR ARE NOT VALID DATA CHARACTERS

IECIN (continued)

THEY ARE DELIMITERS. IN OTHER WORDS:

IF CRSTAT = 0 OR CREOI
A,END WILL POINT TO 1 PAST THE LAST VALID DATA CHARACTER.

IF CRSTAT = CRNORM OR CRETX
A,END WILL POINT TO THAT DELIMITER.

ERRCD : SET IF NO BODY LISTENING DURING ADDRESSING SEQUENCES.

USES: CHAR : TEMPORARY CHARACTER STORAGE.

CALLS:
ATNON
ATNOFF
INTSRC
IECSND
INTACP
IECRD
SCR%$

NOTES: THIS ROUTINE WILL SET UP THE PIA'S FOR BUS TRANSFERS BUT OTHER ROUTINES MUST BE USED TO GET PIA'S OFF OF THE BUS. UNADRS WILL NORMALLY PERFORM THIS FUNCTION.

IECOUT

FUNCTION: TO OUTPUT A BUFFER FULL OF DATA ON THE GPIB. ALSO SEND PRIMARY LISTEN ADDRESS AND SECONDARY ADDRESS IF NECESSARY.

INPUTS: A.STAT : BUSACT - BIT 4 <16.>
0 - SEND MLA, MSA AND SET BIT 4
1 - ASSUME ALREADY ADDRESSED

A.PRIM : PRIMARY ADDRESS (MLA)

A.SEC : SECONDARY ADDRESS (MSA) NOTE: IF = 32 DON"T SEND A SEC. ADDRESS.

A.PTR : POINTING TO FIRST DATA CHARACTER

A.END : POINTING TO LAST DATA CHARACTER

NOOUT : SNDIT - BIT 7 <128>
0 - SEND DATA
1 - SEND DATA BUT ALSO SEND EOI WITH LAST BYTE.

OUTPUT: ERRCD SET IF NO DEVICES LISTENING TO BUS, NRFD AND NDAC SENSED HIGH.

USES: A.PTR AS A MOVING POINTER

CALLS: ATNON
INTSRC
IECSND
ATNOFF
EOION
EOIOFF

NOTE: THIS ROUTINE WILL INITIALIZE THE PIA"S TO GET ON THE BUS LIST SOME OTHER ROUTINE MUST GET OFF THE BUS. THE UNADRS ROUTINE WILL GENERALLY PERFORM THIS FUNCTION. THE ROUTINES DIRECTLY RESPONSIBLE FOR GETTING OFF THE BUS ARE

IECOFF AND IECIFC.

INAGTE

FUNCTION: CONVERTS AN INTEGER TO AN ASCII STRING WHERE THE STRING IS STORED IN R15-R20 AND HAS A NULL STORED AFTER THE LAST CHARACTER.

INPUTS: INDEX REG. IS INTEGER

OUTPUTS: STRING STORED IN R15 - R20 WITH A NULL AFTER THE LAST CHARACTER. R1 WILL POINT TO FIRST CHARACTER.

USES: R0, R1, R2, R15-R20

CALLS: INAITT

NOTE: THE ACTUAL STRING WILL START AT R16 BUT PART OF R15 IS USED BY INAITT.

INAITT/FPAITT

FUNCTION: INAITT CONVERT AN INTEGER TO AN ASCII STRING. FPAITT CONVERT FPN IN FPC TO AN ASCII STRING.

INPUTS: R0 - 2 BYTES : INTEGER TO CONVERT

R1 : POINTER TO FIRST CHARACTER OF RESULT STRING.
NOTE: CHARACTER R1-1. WILL BE CLEARED.

R2 : MAX. LENGTH OF STRING <72.

FPC : IF CALLING FPAITT THIS CONTAINS THE FPN.

OUTPUTS: R2 : POINTER TO LAST CHARACTER OF RESULT STRING.
R1 : POINTER TO FIRST CHARACTER OF RESULT STRING.

USES: FPC, NOUT
IOFLGS

CALLS: FLOAT1
PRIVAL
PULFPN
PSHFPN

INITMT

FUNCTION: INITIALIZE MAGTAPE PIA'S, REWIND IF NECESSARY, DISABLE CRT, DISABLE ABORT.

OUTPUTS: ERRCD : SET IF NO CART. PRESENT.
MTSTAT <- MTSTT2 <- RESET IF REWIND NEW TAPE
DSPSTT <- 128. ,, DISABLES DISPLAY.

CALLS: DRBUSY
DISBLE
PUPTAP
REWIND

INPSTG

FUNCTION: INPUT <INPUT> A STRING VARIABLE FROM THE CURRENT I/O DEVICE

INPUTS: POINT : POINTER TO FIRST CHARACTER POSITION TO RECEIVE DATA.

LENGTH : DIMENSION MAXIMUM LENGTH OF STRING.

OUTPUTS: STRING STORED STARTING AT POINT
CHRCNT : ACTUAL LENGTH READ.
ERRCD : SET IF ERRORS IN HANDLERS.

USES: COLCNT
TCOL - 2 BYTES
CRSTAT

CALLS: GETCHR

NOTES: ALL NORMAL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

INPVAL

FUNCTION: INPUT <INPUT> A VALUE FROM THE CURRENT I/O DEVICE.

INPUTS: A.PRIM : CHECKS IF CURRENT DEVICE IS THE DISPLAY, WHICH REQUIRES SPECIAL HANDLING.

POINT : POINTER TO RESULT VALUE LOCATION.

OUTPUTS: VALUE STORED IN RESULT LOCATION.
ERRCD : SET IF ERROR IN HANDLERS.

USES: CRSTAT
A.PTR
DIGFLG
FFLAG

CALLS: DSPGIN
A9X
GETCHP
ASCFPN
PULFPN

NOTES: ALL NORMAL I/O SYSTEM VARIABLES MUST BE SET CORRECTLY.

INT

FUNCTION: GETS THE INTEGRAL PART OF A FLOATING POINT NUMBER, RETURNING A FLOATING POINT NUMBER. DOES AN ALGOL-FORTRAN STYLE FIX:

INT(1.5)=1, INT(-1.5)=-2

INPUT: FP NUMBER IN STACK

OUTPUT: FP NUMBER IN STACK

USES: R1

CALLS: PSHRET
ZANS
A9X
PSHFPN
FPNEG
RTRN

INTMLA

FUNCTION: INTMLA MULTIPLIES TWO 16 BIT UNSIGNED INTEGERS AND PRODUCES A 32 BIT UNSIGNED RESULT AND THEN ADDS TO THE RESULT A 16 BIT UNSIGNED INTEGER.

INPUTS: ONE INTEGER IS IN R6.
THE SECOND INTEGER IS IN THE "X" REGISTER.
THE INTEGER TO BE ADDED TO THE RESULT IS IN THE "A" AND "B" REGISTERS WITH "A" BEING THE HIGH ORDER BYTE.

OUTPUTS: THE OUTPUT IS IN R6 AND R7 WITH R6 BEING THE HIGH ORDER BYTE AND R7+1 THE LOW ORDER BYTE (R6, R7 =(R6*X)+AB).

THE "X" REGISTER IS LOADED WITH R6 JUST BEFORE EXIT SO THE HIGH ORDER 16 BITS MAY BE CHECKED ON RETURN BY A BEQ OR BNE INSTRUCTION.

USES: R6, R7

CALLS: <-

NOTES: NO ERRORS OR OVERFLOW POSSIBLE.

INTMLC

FUNCTION: INTMLC MULTIPLIES TWO 16 BIT UNSIGNED INTEGERS AND PRODUCES A 32 BIT UNSIGNED RESULT.

INPUTS: ONE INTEGER IS IN R6.
THE SECOND INTEGER IS IN THE "X" REGISTER.

OUTPUTS: THE OUTPUT IS IN R6 AND R7 WITH R6 BEING THE HIGH ORDER BYTE AND R7+1 THE LOW ORDER BYTE (R6, R7=R6*X)
THE "X" REGISTER IS LOADED WITH R6 JUST BEFORE EXIT SO THE HIGH ORDER 16 BITS MAY BE CHECKED ON RETURN BY A BEQ OR BNE INSTRUCTION.

USES: R6, R7

CALLS: <-

NOTES: NO ERRORS OR OVERFLOW POSSIBLE.

IOCLNR

FUNCTION: CLEANS UP THE STACK AFTER THE PROCESSING OF I/O LISTS BY THE I/O SYSTEM. THE I/O SYSTEM PUSHES A SECOND EOLTG BEFORE IT BUILDS A NORMAL LIST ON TOP OF THE POST FIX INFORMATION. THIS ROUTINE SEARCHES FOR THE FIRST EOLTG AND RETRIEVES THAT RETURN ADDRESS. IT THEN SEARCHES FOR A SECOND EOLTG (THE ONE THE EVALUATOR PLACED) AND MOVES THE STACK POINTER TO THAT EOLTG. THE ROUTINE RETURNS TO THE RETURN ADDRESS RETRIEVED EARLY WITH THE FIRST EOLTG. NOTE: THIS ROUTINE IS JSR'D TOO, BUT THE RETURN ADDRESS IS NEVER USED EXCEPT FOR MAYBE DEBUGGING.

USES: DREXTA, R0

CALLS: LOCTG
DREXTA

KEYIN

FUNCTION: SET THE KEYBOARD UP FOR RESPONSE TO THE INPUT STATEMENT, COMPLETE WITH FLASHING\.

INPUT: A,SEC ; USED TO DETERMINE IF INPUT REQUESTED.

OUTPUT: A FULL DATA BUFFER DETIMITED BY A <CR>.

USES: CURSOR
BFLAG

CALLS: NIODEV
CIDLE

NOTE: A, END WILL BE SET UP BY TYPIN WITH THE ASSUMPTION THAT THE EDIT BUFFER (EDTBFR) IS BEING USED.

LOCTG (LOCATE TAG)

FUNCTION: THIS ROUTINE IS USED TO LOCATE A GIVEN TAG ON THE STACK. THE STACK ENTRIES ARE SEARCH FOR THE TAG AND IF IT IS NOT FOUND IN THE STACK THE RESULT POINTER IS SET TO ZERO. A PSUEDO STACK POINTER (R0) IS USED FOR THE STARTING POINT OF THE SEARCH.

INPUTS: R0 IS A POINTER TO THE STARTING LOCATION ON THE STACK-1.
ACC-A IS THE TAG YOU WISH TO FIND.

OUTPUTS: R0 IS UPDATED TO POINT TO THE FOUND ENTRY-1.

USES: R0 AND R1

CALLS: BACKUP.

LOCTGR

FUNCTION: LOCATE A TAG IN A RANGE OF VALUES. THIS ROUTINE IS VERY SIMILAR TO LOCTG EXCEPT IT WILL FIND THE FIRST OF ANY TAGS IN A GIVEN RANGE OF TAG VALUES.

INPUTS: R0 IS THE PSEUDO STACK POINTER, THE ADDRESS OF TAG-1.

ACC-A IS THE LOW VALUE.
ACC-B IS THE HIGH VALUE.

OUTPUTS: R0 HAS THE ADDRESS OF THE FIRST FOUND TAG-1.

USES: R0 AND R1

CALLS: BACKUP.

LOG/LGN/ZZLOG

FUNCTION: COMPUTE LOGE(X) OR LOG10(X)

INPUTS:
1. X ON STACK
2. GLOBAL CTKN. IF CTKN=LGNCOD, LOG10 IS COMPUTED; OTHERWISE LOGE.

OUTPUTS: LOG(X) ON STACK. IF X<=0, X IS RETURNED AND ERLOG IS PUT IN ERRCD.

USES: R3, R3+1

CALLS:
PSHRET
RTRN
DOFP
FLOAT 2 (=FLOAT 1)
FPMUL
FPADD
PSHFPN

METHOD: COMPUTES LOG 2(X) THEN MULTIPLIES BY LOGE(2) AND OPTIONALLY LOG10(E). THE EXPONENT OF X IS CONVERTED TO AN INTEGER; THE FRACTION IS CHANGED TO THE RANGE SQRT(0.5)<=W<=SQRT(2) AND T=(W-1)/(W+1) IS COMPUTED.

THEN LOG 2(W)=T*P(T'2)

WHERE P IS A POLYNOMIAL OF DEGREE 7 IN T'2, NO SOURCE IS GIVEN FOR P WHICH IS ASSERTED TO BE CHEBYSHEV.

MIN,MAX

FUNCTION: FIND EITHER MIN OR MAX OF TOP TWO FP NUMBERS ON STACK.

INPUT:
X, A FP NUMBER ON TOP OF STACK.
Y, A FP NUMBER NEXT ON STACK.
BOTH INPUTS ARE POPPED.

OUTPUT: MIN(X,Y) OR MAX (X,Y) ON STACK

CALLS:
PSHRET
FPCMP
A9X-1
PULFPN
A9X
RTRN

MKFILE

FUNCTION: MARKS MAGTAPE FILES ON THE INTERNAL MAGTAPE.

INPUTS: R8+1 : NUMBER OF FILES TO MARK

R9 : NUMBER OF RECORDS PER FILE - NOTE: MUST BE 3 OR GREATER.

A,STRT : POINTER TO FIRST CHARACTER LOCATION IN THE RECORD. NOTE: MUST BE MTBFR.

FILLOC : PRESENT FILE NUMBER

MTSTT2 : MTEFST - BIT 7 (128) HEAD IN GAP BIT.

MTSREG : TAPE CONFIGURATION STATUS. LENGTH/ CHECK SUM/ HEADER

OUTPUTS: ERRCD : SET IF TOO MANY FILES OR END OF MEDIUM.

FILLOC : MODIFIED TO NEW LOCATION <- NOTE: EXITS WITH HEAD JUST BEFORE #LAST# FILE.

USES: R0, R1, R2, R10
MTPTR
MTMAX
FILEND

CALLS: NER8
MTNULL
MODT88
INAITT
MTCHKS
MARKS
REWIND
MTCLOS
PUPTAP

NOTE: MAGTAPE MUST BE INITIALIZED.
ALSO, BEFORE CALLING, A TEST FOR THE WRITE LOCKED CONDITION SHOULD BE MADE. THE TAPE IS WRITE LOCKED IF PIAMTA BIT 3 (8.) IS SET. IF A FILE IS OPEN MTSTAT - MTEPOS: BIT 6(64) IS SET THEN THIS IS ALSO AN ERROR AND THE ROUTINE SHOULD NOT BE CALLED.

MODMTH

FUNCTION: UPDATES OR MODIFIES THE MAGTAPE HEADER RECORD IN MEMORY.

INPUTS: A,STRT ; POINTER TO BEGINNING OF HEADER RECORD.

ACC,B ; INDEX OFF OF A,STRT TO POSITON TO MODIFY.

INDEX REG. ; POINTER TO NEW HEADER TEXT.

ACC,A ; LENGTH OF NEW HEADER TEXT.

OUTPUTS: MODIFIED HEADER RECORD IN MEMORY.

USES: R0,R1

MTAFIN

FUNCTION: FINDS AND OPENS A FILE ON THE INTERNAL MAGTAPE.

INPUTS:
ACC,B ; FILE TO BE LOCATED
FILLOC-2 BYTES ; PRESENT FILE POSITION.
MTSREG ; TAPE CONFIGURATION FLAGS.
MTSTAT ; TAPE STATUS
MTSTT2 ; TAPE STATUS
A,STRT ; POINTER TO FIRST CHARACTER IN TAPE BUFFER.
A,MAX ; POINTER TO LAST CHARACTER IN TAPE BUFFER.
NOTE: IF 128. BYTE RECORDS A,STRT+128. = A,MAX

OUTPUTS:
FILLOC ; SET TO NEW FILE POSITION
MTSTAT ; SET TO NEW STATUS
MTSTT2 ; SET TO NEW STATUS
ERRCD ; SET IF ERROR
RECCNT-2 BYTES ; OPEN FILE - AVAILABLE RECORD COUNT.

NOTE : IF NO HEADER MODE RECCNT = 65535. MTMAX = A,MAX, MTPTR = A,STRT

USES:
TABPTR
FILEND
R0, R1, FPC

CALLS:
MTCLOS
REWIND
SERCHS
MTREAD
AFPITT
FIX1
A14X
MTBINT

NOTE: MAGTAPE MUST BE INITIALIZED.

MTCLOS

FUNCTION: CLOSES OPEN MAGTAPE FILES; WILL WRITE LAST RECORD IF THE MAGTAPE BUFFER CONTAINS WRITTEN INFORMATION.

INPUTS: MTSTT2; MAGTAPE STATUS - INCLUDES WRITE BIT.

MTPTR; POINTING TO NEXT CHARACTER. SPACE IN MAGTAPE BUFFER, ONLY NEEDED IF WRITING.

OUTPUTS: MTSTT2 ; MAGTAPE STATUS
MTSTAT<-0 ; MAGTAPE STATUS

CALLS: MTBSWP
MTWRIT
A9X

NOTE: MAGTAPE MUST BE INITIALIZED.

MTPADR

FUNCTION: INITIALIZES MAGTAPE AND CHECKS FOR VALIDITY OF I/O OPERATION VERSUS PRESENT MAGTAPE STATUS AND FILE. FILE HEADERS WILL BE MODIFIED IF APPROPRIATE.

INPUTS: A.SEC ; SECONDARY ADDRESS FOR PRESENT I/O OPERATION, THIS WILL DETERMINE WHAT ACTION WILL BE TAKEN.

OUTPUTS: ERRCD ; SET IF ERROR.
MAGTAPE INITIALIZED FOR MAGTAPE OPERATORS, CRT DISABLE, ABORT DISABLED.

CALLS: INITMT
BFRALC
XFRSCN
MTPPRI, MTPINP
MTPOLD, MTPSAV
MTPRD, MTPWRI

MTPIN

FUNCTION: GRABS CHARACTERS FROM MAGTAPE BUFFER FOR BASIC I/O.

INPUTS: MTPTR ; POINTING TO NEXT VALID CHARACTER.

MTMAX ; POINTING TO LAST VALID CHARACTER.

A.PTR ; POINTING TO FIRST CHARACTER OF RESULT STORAGE.

A.MAX ; POINTING TO LAST CHARACTER OF RESULT STORAGE.

MTSTT2 ; MAGTAPE STATUS

RECCNT ; AVAILABLE RECORD COUNT.

IOFUNC ; CURRENT I/O FUNCTION CODE IF = 14. THEN ASSUME DOING A READ AND WILL FILL FROM A.PTR THRU A.MAX. OTHERWISE SEARCH FOR LINE DELIMITERS AND SET UP A.END ADDORDINGLY.

ETXCHR ; INPUT EOF CHARACTER

NULCHR ; INPUT NULL CHARACTER

EOLCHR ; INPUT END-OF-LINE CHARACTER.

OUTPUTS: A.END ; POINTING TO DELIMITER CHARACTER OR IF ONE WAS NOT FOUND BEFORE REACHING A.MAX THEN IT WILL BE POINTING TO 1 PAST THE LAST VALID CHARACTER.

CRSTAT ; 0 IF NO DELIMITER FOUND
; CRNORM IF EOLCHR FOUND
; CREOT IF ETXCHR FOUND
NOTE: A <CR> WILL BE PLACED OVER THE ETXCHR

PNDEOF ; SET TO 128. IF LOGICAL OR PHYSICAL EOF ENCOUNTERED.

MTPTR ; POINTING TO NEXT VALID CHARACTER IN MAGTAPE BUFFER.

ERRCD; SET IF ERROR.

CALLS: MTRBFR
MTBSWP
MTREAD
MTRINT
PULFPN
SCRMB

NOTE: MAGTAPE MUST BE INITIALIZED.

MTPOUT

FUNCTION: TO TRANSFER DATA FROM BASIC'S I/O BUFFER TO THE MAGTAPE BUFFER AND EVENTUALLY TO THE MAGTAPE.

INPUTS:
MTSTT2 ; MAGTAPE STATUS.
RECCNT ; AVAILABLE RECORD COUNT
A.PTR ; POINTING TO FIRST CHARACTER TO XFER.
A.END ; POINTING TO LAST CHARACTER TO XFER.
MTPTR ; POINTING TO NEXT AVAILABLE SLOT IN MAGTAPE BUFFER.
MTMAX ; POINTER TO LAST AVAILABLE SLOT IN MAGTAPE BUFFER.

OUTPUTS:
MTSTT2 ; NEW MAGTAPE STATUS
PNDFLG =128,; IF RECCNT = 0 WHEN CALLED.

CALLS:
BAKARS
MTWRIT
PULFPN

NOTE: MAGTAPE MUST BE INITIALIZED.

NOTE: THIS ROUTINE WILL BACK UP TO LAST RECORD IF THE RECORD IN MEMORY WAS PARTIALLY READ.

MTREAD

FUNCTION: READS A RECORD OFF OF THE MAGTAPE INTO THE MAGTAPE BUFFER.

INPUT: A.STRT ; POINTING TO FIRST CHARACTER OF MAGTAPE BUFFER.

A.MAX ; POINTING TO LAST CHARACTER OF MAGTAPE BUFFER. NOTE: IF USING 128. BYTE RECORDS A.MAX MUST BE SET ACCORDINGLY.

MTSTT2 ; MAGTAPE STATUS

OUTPUTS: RECORD STORED IN BUFFER

MTSTT2 ; NEW MAGTAPE STATUS

RECCNT <= 0 IF PHYSICAL END OF FILE ENCOUNTERED (8NFR"S)

FILLOC ; MODIFIED IF RECCNT <=0

MTEPR ; NUMBER OF RE-READS IF READ ERRORS DETECTED (MAX 9).

ERRCD ; SET OF FATAL READ (10 RE-READS)

USES: M1PTR
MTMAX

CALLS: BAKARS
READRS
MTWAIT
PUPTAP
DELY1S
MTCHKS

NOTE: MAGTAPE MUST BE INITIALIZED.

MTWRIT

FUNCTION: WRITE A BUFFER ONTO THE MAGTAPE.

INPUTS: RECCNT ; AVAILABLE RECORD COUNT FOR OPEN FILE. NOTE: IF IT IS ALREADY 0 DON'T CALL THIS ROUTINE.

A,STRT ; POINTING TO FIRST CHARACTER

A,MAX ; POINTING TO LAST CHARACTER; MUST BE ON A RECORD BOUND.

MTSREG ; TAPE CONFIGURATION.

OUTPUTS: RECCNT ; MODIFIED
MTSTT2 ; MAGTAPE STATUS
EPRCD ; SET IF WRITE LOCKED.

USES: MTPTR, MTMAX, MTFLGS

CALLS: MTCHKS
WPRS
MTWAIT
WREND.
PUPTAP

NOTE: MAGTAPE MUST BE INITIALIZED

NEWBFR

FUNCTION: GET A NEW BUFFER FULL OF DATA FOR INPUT PROCESSING.

INPUTS: PPMODE : PERCENT MODE FLAG - SET IF IN PERCENT MODE

A.STRT : POINTER TO FIRST CHARACTER OF THE BUFFER.

A.MAX : POINTER TO THE LAST CHARACTER OF THE BUFFER. NOTE: ONE POSITION PAST THAT INDICATED BY A.MAX MAY BE USED FOR DELIMITER STORAGE. SO WATCH OUT.

OUTPUTS: CRSTAT : SET BY I/O HANDLERS

A.PTR : SET TO FIRST CHARACTER

A.END : SET TO LAST VALID CHARACTER. (NOTE: END OF LINE DELIMITERS ARE NOT CONSIDERED TO BE VALID CHARACTERS.

A.STAT: BFRSTT - BIT 5 <32.> SET
ERRCD: SET BY I/O HANDLERS

USES: SCRATCH AREA WHEN PURGING UNDER PERCENT MODE.

CALLS: NXBFR <- REALLY PART OF NEWBFR
PSHFPN
PULFPN
XFRCTL
IECIN
KEYIN
NIODEV
MTPIN

NOTE: ALL NORMAL I/O SYSTEM VARIABLES MUST BE SET CORRECTLY.

NFR8

FUNCTION: TO SEARCH IN REVERSE FOR AN 8NFR FILE MARK.

CALLS: BAKARS
TST8NF

NOTE: MAGTAPE MUST BE INITIALIZED.

NORM

FUNCTION: NORMALIZE FLOATING POINT NUMBER ON STACK, CHECKS FOR OVERFLOW, UNDERFLOW, AND ZERO RESULT. UNDERFLOW GOES TO ZERO. OVERFLOW IS REPLACED BY THE LARGEST POSSIBLE NUMBER.

NORM IS REACHED BY A JMP AFTER PUTTING THE RETURN ADDRESS IN FNRET; I.E., LAST INSTRUCTION IN NORM IS A JMP TO RTRN.

INPUT: FP NUMBER ON STACK.

OUTPUT: NORMALIZED OR ZERO FP NUMBER ON STACK. IN THE EVENT OF OVERFLOW, ERFPOV IS PLACED IN ERRCD. THE NUMBER OF LEFT SHIFTS REQUIRED TO NORMALIZE IS KEPT IN THE GLOBAL FUZR FOR SUBSEQUENT USE IN FUZZY COMPARE.

OUTBFR

FUNCTION: ATTACH DEVICE HANDLER TO OUTPUT CURRENT OUTPUT BUFFER. ALSO CONVERT CONTROL CHARACTERS TO #LIST# FORMAT IF REQUIRED.

INPUTS: NOOUT: SNDIT : BIT 7 (128.) SEND EOI WITH LAST BYTE IF SET.

. LSTFMT : BIT 7 (2.) SET IF #LIST# FORMAT CONTROL CHARACTER.

. NOWRIT : BIT 0 (1.) SET IF SHOULDN"T WRITE THE BUFFER. AND ERROR WILL BE SET.

A.PTR : POINTER TO FIRST CHARACTER SLOT.

A.END : POINTER TO MOST RECENTLY USED CHARACTER SLOT USED BY PUTBYT.

IOFUNC : CODE FOR PRESENT I/O FUNCTION.

A.STRT : POINTER TO FIRST CHARACTER SLOT.

IECRLF : CR VS. CR/LF FLAG.

OUTPUTS: A.PTR, A.END; INITIALIZED FOR NEW BUFFER.

ERRCD ; SET IF ERROR IN I/O HANDLER, OR NOWRIT BIT SET IN NOOUT.

USES: SCRTCH SPACE

CALLS: SCRMD
XFRCTL
FILEOT
NIODEV
IECOUT
DSPOUT
MTPOUT
NOACT

NOTE: ALL I/O SYSTEM VARIABLES MUST BE SET CORRECTLY.

PCHAR

FUNCTION: DISPLAY A CHARACTER ON THE CRT, THIS INCLUDES CONTROL CHARACTERS.

INPUTS: ACC, A - ASCII CHARACTER.

CALLS: DRBUSY
WAIT.
PCHR.1-THE ACTUAL CHARACTER POINTER
DSPCPZ
PAGE - INTERNAL TO CRTDRV
HOME - INTERNAL TO CRTDRV
MTPRWD
CRTRST

NOTES: THE STATUS OF THE DISPLAY IS KEPT IN THE VARIABLE DSPSTT. DSPSTT CONTAINS THE FOLLOWING INFORMATION IN THE FORM OF A BIT PATTERN.

DSPDIS - SET IF DISPLAY DISABLED
HOMFLG - SET IF HOME PENDING
PAGFLG - SET IF PAGE PENDING
RWNDFG - SET IF REWIND PENDING
CPYFLG - SET IF COPY PENDING.

PCHR.1

FUNCTION: THIS ROUTINE PAINTS THE CHARACTERS ON THE CRT. THEY ARE STORED OR WRITTEN THRU AS DISCRIBED BY BLINK. ALSO DISCUSSES THE EXTERNAL ROM PACK CHARACTER POINT CAPABILITY.

INPUT: BLINK : 0 - STORE MODE
1 - WRITE-THRU MODE

NOTES: 1) TO USE THE EXTERNAL CHARACTER FONT CAPABILITY ONE MUST STORE THE BANK NUMBER AND SERVICE ROUTINE ADDRESS IN THE 3 BYTE CHRVCT VARIABLE. CONTROL WILL BE PASSED TO THIS ROUTINE ANYTIME A PRINTABLE CHARACTER REQUEST IS PROCESSD THRU PCHR.1. PLEASE REFER TO THE CODE IN CRTDRV FOR MORE INFORMATION.

2)DON'T BLINK CONTROL CHARACTERS.

PRINTF

THE MODULE PRINT F HAS SEVERAL ROUTINES AND THE MODULE'S FUNCTION WITHIN THE SYSTEM IS TO HANDLE CONVERSIONS FROM INTERNAL FORMAT TO ASCII FOR BOTH NUMBERS AND STRINGS.

FOR NUMBERS, A TRUE CONVERSION FROM BINARY FLOATING POINT TO ASCII IS MADE. FOR STRINGS, IT USUALLY ENTAILS TAKING THEM DIRECTLY FROM MEMORY AND OUTPUTTING THEM.

THERE ARE TWO DISTINCT PROCESSES INVOLVED:

1)THE FIRST IS WHEN CONVERSIONS ARE GOVERNED BY AN IMAGE STRING SUPPLIED BY THE USER.

2)THE SECOND IS WHEN CONVERSIONS ARE GOVERNED BY DEFAULT IMAGE STRING

ALL OUTPUT FOR BOTH PROCESSES IS PASSED THROUGH AN EXTERNAL ROUTINE - PUTBYT; THE USER OF PRINTF MUST HAVE AN INTIMATE KNOWLEDGE OF PUTBYT SINCE, BEFORE PRINTF CAN BE USED, PUTBYT MUST BE SET UP PROPERLY.

ALSO, FOR NUMBER OUTPUT, FPNASC MUST BE CALLED FOR EACH NUMBER OUTPUT BEFORE BEFORE PASSING CONTROL TO PRINTF EACH TIME.

PRINTF CONSISTS OF THE FOLLOWING USER CALLABLE ROUTINES:

A)FMTINI (FORMAT INITIALIZE)
B)FMTPNT (FORMAT PRINT; USING LIST OUTPUTS)
C)DEFPNT (DEFAULT PRINT, NON-USING OUTPUTS (PRINT LIST, STR, ETC))
D)FMTCLN (USED TO OUTPUT ANY REMAINING SPECIFIER'S THAT DO NOT REQUIRE DATA ITEMS FROM THE PRINT LIST)

THE DESCRIPTIONS FOR EACH ROUTINE FOLLOW THIS INTRODUCTORY VERBAGE.

THE GENERAL PROCEDURE FOR OUTPUTTING USING IMAGE STRING SUPPLIED CONTROL IS AS FOLLOWS:

1. CALL FMTINI TO INITIALIZE FMTPNT/FMTCLN.
2. SET UP PUTBYT.
3. FOR EACH STRING OUTPUT ENCOUNTERED, SIMPLY CALL FMTPNT.
4. FOR EACH NUMERIC OUTPUT ENCOUNTERED, CALL FPNASC FOLLOWED BY A CALL TO FMTPNT.

PRINTF (continued)

5. WHEN ALL ITEMS IN THE LIST ARE EXAUSTED, CALL FMTCLN.

THE GENERAL PROCEDURE FOR DEFAULT PRINTING IS (OUTPUT NOT CONTROLLED BY A USERS IMAGE STRING):

1. SET UP PUTBYT.
2. FOR EACH STRING OUTPUT ENCOUNTERED, CALL DEFPNT.
3. FOR EACH NUMERIC OUTPUT ENCOUNTERED, CALL FPNASC FOLLOWED BY A CALL TO DEFPNT.

THE NUMBER OF ERROR CONDITIONS ARE NUMEROUS AND ARE DEPENDANT ON THE IMAGE STRING AND DATA VALUE RELATIONSHIPS AS WELL AS SYNTAX OF THE IMAGE STRING. A DETAILED LIST OF ERRORS THAT CAN OCCUR CAN BE FOUND IN THE USERS MANUAL.

THE COMMON COMMUNICATON LOCATIONS ARE AS FOLLOWS:

ISP - IMAGE STRING POINTER; 2 BYTE IMAGE STRING POINTER; POINTS TO THE FIRST CHARACTER IN THE IMAGE STRING AT ALL TIMES.
ISL - IMAGE STRING LENGTH; 2 BYTE INTEGER EQUAL TO THE NUMBER OF CHARACTERS IN THE IMAGE STRING.
DT - DATA TYPE; SINGLE BYTE FLAG WHERE DT=0 FOR STRING AND 1 FOR NUMERIC;DT=1 IMPLIES FPNASC HAS BEEN CALLED.
DP - DATA POINTER; 2 BYTE STRING POINTER; POINTS TO THE FIRST CHARACTER IN THE STRING.
DL - DATA LENGTH; 2 BYTE INTEGER EQUAL TO THE NUMBER OF CHARACTERS IN THE STRING.

PRISTG

FUNCTION: PRINT A STRING. IF NORMAL XFER THE STRING WILL BE ENTERED INTO THE OUTPUT BUFFER.

INPUTS: POINT ; POINTER TO STRING
LENGTH ; LENGTH OF STRING

USES: DP, DL, DT

CALLS: PRISPC

NOTE: ALL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

PRIVAL

FUNCTION: PRINT A VALUE. IF A NORMAL XFER THE VALUE WILL BE ENTERED AS ASCII CHARACTERS IN THE OUTPUT BUFFER.

INPUTS: POINT ; POINTS TO VALUE

USES: DT

CALLS: PRISPC

NOTE: ALL NORMAL I/O SYSTEM VAIABLES MUST BE SET UP CORRECTLY.

PSHFPN

FUNCTION: PUSH FLOATING

(1) THE FLOATING PRINT NUMBER IN IX+0,..,IX+7 IS PUSHED ON THE STACK.

(2) #VALTG# TAG IS PUSHED ON THE STACK.

(3) RETURN TO CALLER VIA DREXTB.

INPUTS: FLOATING POINT NUMBER POINTED TO BY IX, TOP 2 BYTES OF STACK.

OUTPUTS: FLOATING POINT NUMBER ON THE STACK. TAG VALTG ON STACK.

PSHRET

FUNCTION: THIS ROUTINE IS USED TO SAVE THE RETURN ADDRESS OFF THE SYSTEM STACK ONTO A SECONDARY TEN ELEMENT STACK.

INPUTS: ADDRESSES ON THE STACK.

OUTPUTS: NEW ENTRY ON SECONDARY STACK.

PULFPN

FUNCTION: POP FLOATING

(1) THE TAG IS THROWN AWAY.

(2) THE FLOATING POINT NUMBER ON THE STACK IS POPPED TO IX+0...IX+7.

(3) RETURN VIA DREXTB.

INPUT STACK: (TOP)

1 BYTE TAG
8 BYTES FP NUMBER

OUTPUT: FP NUMBER IN IX+0...IX+7

PUTBYT

FUNCTION: ENTERS A BYTE INTO THE CURRENT OUTPUT BUFFER. IF THAT BUFFER BECOMES FULL OR AN EOLCHR<CR> WAS IN THE BUFFER THEN THAT BUFFER IS TRANSMITTED TO THE CURRENT I/O DIVICE AND A FRESH BUFFER IS ESTABLISHED.

INPUTS: ACC.A ; THE CHARACTER TO BE PUT.
A.END ; POINTER TO LAST CHARACTER PUT.
A.MAX ; POINTER TO LAST SLOT IN THE OUTPUT BUFFER.
TABCNT ; CHARACTER COUNTER FOR PRINTF

OUTPUTS: ERRCD ; SET IF ERROR IN I/O HANDLERS
TABCNT ; MODIFIED CHARACTER COUNTER.
A.END ; UPDATED.

CALLS: OUTBFR

NOTE: ALL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

QCROSS

FUNCTION: QCROSS IS INVOLKED AS A RESULT OF THE #POINTER# COMMAND. ITS FUNCTION IS TO POSITION THE GRAPHIC POINTER IN RESPONSE TO THE JOY STICK INPUT AND EXIT THE PROCESS UPON LEGAL INPUT FROM THE KEYBOARD.

INPUTS: ANALOG VOLTAGE FROM JOY STICK AND TERMINATING CHARACTER FROM KEYBOARD.

OUTPUTS: QCROSS CONTINUOUSLY SLEWS THE CRT DA"S SO AS TO TRACK THE JOYSTICKS ANALOG INPUT VOLTAGE UNTIL SUCH TIME AS A LEGAL CHARACTER IS INPUT FROM THE KEYBOARD. THE OUTPUT THEN IS THE DA"S VALUE AND A SINGLE CHARACTER STRING CORRESPONDING TO THE INPUT FROM THE KEYBOARD. WHILE THE TRACKING IS IN PROCESS, THE GRAPHIC POINTER IS DISPLAYED ON THE SCREEN. QCROSS THEN JUMPS TO GINSET FOR THE COMPLETION OF THE #POINTER# COMMAND.

USES: NO PSEUDO REGISTERS

CALLS: CROS, DIMSTR, ADRDEV, A7X, GINSET

RESTRICTIONS, ERRORS, NOTES: IF THE STRING VARIABLE HAS NOT BEEN PREVIOUSLY DIMENSIONED, DIMSTR CAN RETURN A MEMORY FULL ERROR.

REAHDR

FUNCTION: READS <READ> A 2-BYTE BINARY HEADER FROM THE CURRENT I/O DEVICE AND CHECKS IT FOR VALIDITY.

INPUTS: NONE

OUTPUTS: R0 : CONTAINS LENGTH INFORMATION IF IT WAS A STRING HEADER.

ACCA : 0 = VALUE HEADER
1 = STRING HEADER
-1,-2 = ILLEGAL HEADER
-3 = I/O ERROR OR EOF.

ERRCD : SET IF ERROR.

USES: R0

CALLS: BINBYT
BININ

NOTES: REQUIRES ALL NORMAL I/O SYSTEM VARIABLES BE SET UP CORRECTLY.

REASTG

FUNCTION: READS <READ> A STRING FROM THE CURRENT I/O DEVICE.

INPUTS: LENGTH : DIMENSIONED STRING LENGTH
POINT : POINTER TO LOCATION OF RESULT STRING.

OUTPUTS: 1)CHRCNT : NEW RESULT STRING LENGTH.
2) THE STRING IS STORED IN MEMORY STARTING AT POINT.
3) ERRCD IS SET IF ERROR OCCURED.

USES: R0
A.PTR
A.MAX
SCRTCH SPACE <SCRTCH=SCRTCH+255.>

CALLS: READHDR
BININ

NOTES: ALL NORMAL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

REAVAL

FUNCTION: READ <READ> A VALUE FROM THE CURRENT I/O DEVICE.

INPUTS: POINT - POINTS TO LOCATION THAT IS TO RECEIVE THE VALUE.

OUTPUTS: READ VALUE.
ERRCD SET IF ERROR
EOF SET IF EOF ENCOUNTERED

USES: A.MAX
A.PTR

CALLS: A7X
BININ

NOTES: REQUIRES THAT ALL NORMAL I/O SYSTEM CONTROL VARIABLES BE SET UP CORRECTLY.

RENUM

FUNCTION: RENUM IS USED TO RENUMBER A PROGRAM POINTED TO BY PGMPTR.

INPUTS: R0=STARTING NUMBER
R1=INCREMENT VALUE
R2=STARTING STATEMENT NUMBER

OUTPUTS: A RENUMBERED PROGRAM IN ACCORDANCE WITH THE INPUTS.

USES: R0,R1,R2,R3,R4,R6,R7

CALLS: PSHRET, DISBLE, ENABLE, RTRN, PUSHX, GETLAR, GETSMA, A9X, INCXN, A8X

ERRORS: SAME AS FOR RENUMB

RENUMB

FUNCTION: RENUMB IS INVOLKED AS A RESULT OF THE BASIC #RENUMBER# COMMAND; IT IS USED TO RENUMBER A PROGRAM POINTED TO BY PGMPTR.

INPUTS: IT ACCEPTS UP TO THREE VALUES ON THE STACK AND SETS DEFAULTS FOR VALUES NOT SEEN AS FOLLOWS:

STARTING NUMBER = 100
INCREMENT VALUE = 10
STARTING STATEMENT NUMBER = 100

OUTPUTS: THE OUTPUT IS ESSENTIALLY A RENUMBERED PROGRAM FOR A PROGRAM THAT HAS NO RENUMBERABLE STATEMENT NUMBERS, RENUMB IS THE SAME AS A NO-OP.

USES: R0,R1,R2,R3,R4

CALLS: PSHRET, RENUM, TYPARG, FIX1

NOTES AND ERRORS: FOR A SUCCESSFUL RENUMBER, THE STACK IS LEFT CLEAN.

SINCE RENUMB MUST MODIFY CRITICAL AREAS OF A USERS PROGRAM, THROUGH RENUM, BREAKS AND ABORTS ARE DISABLED FROM TIME TO TIME DURING THE RENUMBER PROCESS.

ERRORS WILL RESULT UNDER THE FOLLOWING CONDITIONS:

1. IF ANY VALUE IS OUTSIDE, OR IS INCREMENTED OUTSIDE, THE RANGE OF 0-65,535.
2. IF THE STARTING NUMBER OR INCREMENT VALUE IS 0.
3. IF THE INITIAL PARAMETERS ARE SUCH THAT STATEMENT REPLACEMENT OR INTERLACING WILL OCCUR.

QRND

FUNCTION: GENERATES RANDOM NUMBERS WITH A UNIFORM DISTRIBUTION IN (0,1).

IF THE INPUT > 0 OR <=1> ANOTHER RANDOM NUMBER IS GENERATED (AND REPLACES THE KERNEL).

IF THE INPUT=0 THE KERNEL IS RESET TO A RANDOM VALUE.

IF 0> INPUT > -1, THE KERNEL IS RESET TO THE INPUT.

INPUT: FLOATING POINT NUMBER POINTED TO BY IX.

OUTPUT: FP NUMBER ON STACK

USES: KERNEL
R4,R7
T1,T2

CALLS: PSHRET
SHVR
PULFPN
SETRND
A9X
PSHFPN
INTMLC
INTMLA
NORM

NOTES: THE NEXT KERNEL (AND OUTPUT) IS GENERATED BY MULTIPLYING THE OLD KERNEL *5'17 MODULE 2'48.

RNDATA

FUNCTION: ROUNDS A FLOATING POINT NUMBER AND CONVERTS TO ASCII.

INPUTS: ACC A : NUMBER OF OUTPUT DIGITS DESIRED

X5...X0: FP MANTISSA, NOT NECESSARILY NORMALIZED.

DEXP, DEXP+1: POWER OF TEN WHICH MULTILIES THE MANTISSA.

OUTPUTS: N12,N11...N1 12 DIGIT ASCII FIELD

ES: EXPONENT SIGN, + OR - ASCII

E3,E2,E1: 3 DIGIT ASCII EXPONENT FIELD

USES: T4,T1

CALLS: TMULT

NOTE: 40 BASE 10 IS ADDED TO THE LAST BYTE OF THE MANTISSA PRIOR TO EXTRACTION OF THE NEXT DIGIT, THIS SOMETIMES CAUSES ROUNDING.

RPN

FUNCTION: RPN IS INVOLKED WHENEVER DIMENSIONING OR SUBSCRIPTING IS REQUIRED. ITS BASIC FLOW IS AS FOLLOWS:

FLOWCHART

SINCE THE PROCESSES ARE ALL QUITE DIFFERENT, A SEPERATE #INPUTS THRU NOTES# SECTION WILL BE PROVIDED FOR EACH PROCESS-
1) FRONT-END PROCESSING (FEP)
2) SUBSCRIPTING (SUB)
3) NEW ARRAY DIMENSIONING (NAD)
4) ARRAY RE-DIMENSIONING (ARD)
5) NEW STRING DIMENSIONING (NSD)
6) STRING RE-DIMENSIONING (SRD)

FEP INPUTS: BRKCNT AND DIMFLG ARE USED TO DETERMINE WHETHER DIMENSIONING OR SUBSCRIPTING IS TAKING PLACE.

ONE VALUE MUST BE ON THE STACK AND AN OPTIONAL SECOND VALUE MAY BE THERE; THE VALUE(S) ARE FOLLOWED BY A POINTER TO THE NAME TABLE.

IF ONLY ONE VALUE IS ON THE STACK, IT IS TAKEN AS THE ROW VALUE; IF TWO VALUES ARE ON THE STACK, THE FIRST VALUE POPPED IS THE COLUMN VALUE.

FEP OUTPUTS: INT1=ROW VALUE/STRING LENGTH

INT2= COLUMN VALUE (IF PRESENT).

DIMCNT= IS NON-ZERO IF INT2 IS VALID; OTHERWISE 0.

TPOINT= ABSOLUTE ADDRESS OF NAME TABLE ENTRY.

"X" REGISTER= ABSOLUTE ADDRESS OF NAME TABLE ENTRY.

"A" REGISTER= NAME TABLE POINTER STACK TAG.

FEP USES: R0,R4

FEP CALLS: PSHRET, TYPARG, FIX1

FEP ERRORS: AN ERROR WILL OCCUR IF THE VALUE(S) ARE NOT WITHIN THE RANGE 1<= V <= 65,535.

SUB INPUTS: SAME AS FRONT-END OUTPUTS.

SUB OUTPUT: PAETG ENTRY ON THE STACK (ABSOLUTE POINTER TO THE ARRAY ELEMENT PLUS SOME ASSUNDRY INFORMATION).

RPN (continued)

SUB USES: R7

SUB CALLS: INTMLA

SUB ERRORS: ERRORS OCCUR FOR THE FOLLOWING REASONS:

1)A SIMPLE VARIABLE HAS BEEN SUBSCRIPTED.
2) THE ROW VALUE IS OUT OF THE DIMENSIONED RANGE.
3) THE COLUMN VALUE IS OUT OF SYNC OR OUT OF RANGE WITH THE DIMENSIONED PARAMETERS.

NAD INPUTS: SAME AS FEP OUTPUTS.

NAD OUTPUTS: MEMORY IS ALLOCATED BY CREATING A NEW DATA ENTRY AS PER THE ROW AND COLUMN VALUES (DEFAULT COLUMN VALUE=1). THE ARRAY PARAMETERS ARE SET IN THE NAME TABLE. THE UNDEFINED BIT IS SET IN ALL VALUES.

NAD USES: R7

NAD CALLS: INTMLA, ARX, COMPR, DISBLE, PSHRET, ENABLE

NAD NOTES/ERROR: PART OF THE ALLOCATION ROUTINE MAY INVOLVE CONSIDERABLE TIME SINCE THE MEMORY COMPRESS ROUTINE MAY BE INVOLKED, ALSO, DURING THE TIME ACTUAL ALLOCATION IS TAKING PLACE, HARDWARE INTERRUPTS ARE DISABLED. IN ADDITION, DISBLE IS INVOLKED FOR BOTH THE ALLOCATION PROCESS AND DURING THE TIME THE UNDEFINED BITS ARE BEING SET IN THE VALUES.

ERRORS WILL RESULT IF THE USER ATTEMPTS TO DIMENSION A SIMPLE DEFINED VARIABLE OR IF THERE IS NOT ENOUGH MEMORY TO DO THE DIMENSIONING.

ARD INPUTS: SAME AS FEP OUTPUTS

ARD OUTPUTS: THE NEW ROW AND COLUMN VALUES ARE PUT IN THE NAME TABLE AND THE ALL OK BIT IS CLEARED.

ARD USES: R7

ARD CALLS: INTMLA, PSHRET

ARD NOTES/ERRORS: HARDWARE INTERRUPTS ARE DISABLED DURING THE SHORT TIME NEW VALUES ARE BEING UPDATED IN THE NAME TABLE.

AN ERROR WILL OCCUR IF THE USER TRIES TO REDIMENSION THE ARRAY IN SUCH A WAY AS TO INCREASE THE AMOUNT OF MEMORY ORIGINALLY ALLOCATED.

NSD INPUTS: SAME AS FEP OUTPUTS

NSD OUTPUTS: MEMORY IS ALLOCATED BY CREATING A NEW DATA ENTRY AS PER THE LENGTH VALUE. THE STRING PARAMETERS ARE SET IN THE NAME TABLE.

RPN (continued)

NSD USES: <-

NSD CALLS: PSHRET, DIMSTR

NSD NOTES: DIMSTR IS WHAT DOES MOST OF THE WORK FOR NAD.

SRD INPUTS: SAME AS FEP OUTPUTS

SRD OUTPUTS: THE NEW LENGTH IS PUT INTO THE NAME TABLE AND THE WORKING LENGTH IS SET EQUAL TO NEW LENGTH IF NEW LENGTH IS SHORTER THAN CURRENT WORKING LENGTH.

SRD USES: <-

SRD CALLS: PSHRET, DISBLE, ENABLE

SRD NOTES: AN ERROR WILL OCCUR IF THE USER TRIES TO REDIMENSION THE STRING IN SUCH A WAY AS TO INCREASE THE AMOUNT OF MEMORY ORIGINALLY ALLOCATED.

RTRN

FUNCTION: PULLS ONE SAVED RETURN ADDRESS OFF OF SECONDARY STACK AND PASSES CONTROL TO THAT POINT.

RESTRICTION, NOTES: YOU SHOULD JUMP TO THIS ROUTINE...

SAFE

FUNCTION: THIS ROUTINE IS USED TO LET ROM PACKS SERVICE INTERRUPTS WHEN THE SYSTEM MAY NOT BE SAFE. THE INTERRUPT ROUTINES WILL BE CALLED WHEN SAFE IS CALLED BUT THE SYSTEM INTERRUPT COUNTER IS NOT AFFECTED. SAFE SHOULD ONLY BE CALLED IF ALL SYSTEM POINTERS AND TABLES ARE IN CORRECT FORM. THE ABORT FUNCTION CAN NOT BE INITIATED DURING THE TIME SAFE IS ACTIVE.

RESTRICTION, NOTES: THIS ROUTINE CAN CALL ALL INTERRUPT PROCESSORS SO A LIST OF VARIABLES AND ROUTINES CAN NOT BE GENERATED. SEE ALSO DISBLE AND ENABLE.

SETRND

FUNCTION: SETS THE KERNEL OF THE RANDOM NUMBER GENERATER TO ITS DEFAULT VALUE.

OUTPUT: THE GLOBAL KERNEL IS SET TO ITS DEFAULT.

CALLS: PSHFPN
PULFPN

SHMR

FUNCTION: SHIFT MANTISSA RIGHT IN PLACE.

INPUT: ACC A CONTAINS THE NUMBER OF SHIFTS REQUIRED. IT IS DESTROYED. THE FP MANTISSA IS IN IX+3...IX+8.

OUTPUT: THE SHIFTED FP MANTISSA IS IN IX+3...IX+8.

SIG

FUNCTION: SIGNUM

INPUT: FP NUMBER X ON TOP OF STACK.

OUTPUT: FP 0.0 IF ON STACK X = 0
FP +1.0 IF ON STACK X > 0
FP -1.0 IF ON STACK X < 0

SLN

FUNCTION: LEFT SHIFT MANTISSA

INPUT: ACC A : NUMBER OF BIT POSITIONS TO SHIFT

IX : FP MANTISSA IS IN IX+3...IX+8

OUTPUT: ACC B : CONTAINS BITS SHIFTED OUT OF MANTISSA

IX+3...IX+8 CONTAIN THE REST OF THE SHIFTED MANTISSA.

SQR

FUNCTION: SQUARE ROOT

INPUT: FLOATING POINT NUMBER X ON STACK

OUTPUT: SQR(ABS(X)) ON STACK
EPSQR IS PLACED IN ERRCD IF X<0.0

USES: T1,R1

CALLS: PSHRET
DOFP
RTRN

NOTE: STARTS BY GETTING SQUARE ROOT TO 7 BITS BY THE DIVISION- LIKE AL

STKBLD

FUNCTION: BUILDS A #NORMAL FIX# STACK OF POINTERS OUT OF A #POST FIX# STACK ENTRY. FOR EXAMPLE: PRINT 1,2,3 LOOKS LIKE 3,2,1 ON THE STACK.

INPUT: R0 - POINTS TO BEGINNING OF #POST FIX# STACK

USES: DREXTB+1,+2

CALLS: CLRARG
DREXTB

NOTES: CLRARG IS USED TO GET NEXT TAG ON POST FIX STACK. IF THAT TAG IS AN EOL OR AN ATSNTG THE ROUTINE EXITS. IF A SEMITG IS ENCOUNTERED A SEMITG IS PUSHED ON THE #NORMAL# STACK. ALL OTHER ITEMS ARE TAGGED AS BAKSTG AND INCLUDES THE RESULTING BYTE FROM CLRARG AND AN INTERSTACK POINTER THAT POINTS TO THE ORIGINAL ENTRY IN THE POST FIX STACK. IT ALSO SKIPS ITM2TG WHICH IS USUALLY USED BY THE I/O SYSTEM AS ITS OWN EOL TAG.

SYMTAB

FUNCTION(S): SEARCHES THE SYMBOL TABLE FOR THE ENTRY CORRESPONDING TO THE SPECIFIED NAME. IF THE ENTRY IS FOUND A POINTER TO IT IS RETURNED; IF THE ENTRY IS NOT FOUND ONE IS CREATED.

INPUTS: PUT NAME IN R3 (TWO BYTES)
SET DEFFLG (BIT 'H20 IN TFLGS1) TO 0
SET STR (STRING)(BIT 'H10 IN TFLGS2) TO 0 OR 1
AS DESIRED, 1 IMPLIES STRING.

NOTE: TFLGS1 SHOULD BE RESTORED TO ITS ORIGINAL STATE IF THE TRANSLATOR WAS (MIGHT HAVE BEEN) ACTIVE WHEN THE ROM PACK GAINED CONTROL.

OUTPUTS: RETURNS A POINTER TO THE SYMBOL TABLE (NAME TABLE OR NT) IN R3

USES: R0, R3, R7, AND R8

CALLS: MAY CALL COMPR

RESTRICTIONS, SUGGESTIONS, NOTES:
1)THIS ROUTINE MAY CALL COMPR IF IT NEEDS TO CREATE A NEW ENTRY

2)USER NAMES ARE OF THE FORM
<LETTER><ASCII BLANK> OR
<LETTER><DIGIT> OR
<LETTER> &

NONE OF THESE FORMS SHOULD NORMALLY BE USED BY A ROM PACK TO AVOID COLLISIONS.

3)BECAUSE OF 1) AND THE NOTE PREVIOUS NEW ENTRIES FOR ROM PACKS SHOULD BE CREATED AT POWERUP TIME.

TMULT

FUNCTION: MULTIPLIES FP MANTISSA BY 10

INPUT: FP MANTISSA IN X5...X0, NEED NOT BE NORMALIZED.
ACC A, OFFSET, DESTROYED

OUTPUT: OVERFLOWED BITS INTO ACC B
REMAINDER OF MANTISSA IN X5...X0.
THE OFFSET THAT WAS IN ACC A IS ADDED TO THE REMAINDER FOR ROUNDING.

USES: Y0...Y5

TRIG

FUNCTION: COMPUTES SIN, COS AND TAN

INPUTS: 1. CONV1 POINTS TO DEGCON, GRDCON, OR RADCON ACCORDING TO THE TRIGONOMETRIC MODE

2. THE ARGUMENT X IS ON THE STACK

3. GLOBAL CTKN DETERMINES FUNCTION.

OUTPUTS: THE RESULT IS RETURNED ON THE STACK

1. IF THE ARGUMENT EXCEEDS ABOUT 40000 RADIANS A ZERO RESULT IS STACKED AND ERTRNG IS PLACED IN ERRC.

2. IF TAN(X)=INFINITY THEN THE LARGEST FP NUMBER IS RETURNED; NO ERRCD IS SET.

APPROXIMATIONS 1. COS (X)=P(X'2) WHERE P, POLYNOMINAL OF DEGREE 6 IN X'2 WAS USED, CLAIMED TO BE A CHEBYSHEV APPROXIMANT, SIMILAR TO HART ?3823 WHICH HAS 16.25 DIGITS PRECISION

2. SIN(X)=X*P(X'2) WHERE P IS OF DEGREE 5. SIMILAR TO HART ?3044 WHICH IS GOOD TO 17.48 DIGITS.

3. TAN(X)=X*P(X'2)/Q(X'2) WHERE P IS DEGREE 3 AND Q IS DEGREE 4. SIMILAR BUT NOT THE SAME AS HART ?4286.

USES: R1,R1+1
T4

CALLS: PSHRET
PSHFPN
FPMUL
ZANS
SLN
NORMR
FPSUB
DOFP
FPNEG
RTRN

FPDIV

TYPARG (TYPE OF ARGUMENT)

FUNCTION: TO TEST AND SIMPLIFY ARGUMENTS ON THE STACK, THIS ROUTINE USES R0 AS A PSEUDO STACK POINTER AND R4 TO RECORD THE RESULTS IT FINDS AND THE ACTIONS TAKEN. IF IT FINDS AN ARRAY IT WILL CALL MATSIZ TO INSPECT THE ARRAY FOR UNDEFINED ENTRIES AND TO PLACE THE END OF DATA ADDRESS ON THE STACK. MATSIZ ALSO PUTS THE SHAPE OF THE ARRAY ON THE STACK IN THE EXISTING STACK ENTRY. IF A LITERAL OR NAME TABLE STRING ENTRY IS FOUND IT IS CONVERTED TO A #POINTER TO STRING COUNT ENTRY.# IF A SIMPLE VARIABLE OR SUBSCRIPTED VARIABLE IS FOUND IT WILL BE CONVERTED TO A CONSTANT. THE ENTRIES FOR CONSTANTS AND POINTERS ARE ONLY NOTED IN R4 FOR THE CALLER. IF ANY STACK ENTRY IS FOUND FOR AN ARGUMENT R0 IS UPDATED TO POINT TO THE NEXT STACK ENTRY. IF THE END OF STACK IS FOUND R0 IS SET TO ZERO.

R4 BIT PATTERN MEANINGS:

RR AA BB CC

RR = BITS USED BY TTYPRES
AA = TYPE OF FIRST OPERAND.
BB = TYPE OF SECOND OPERAND.
CC = TYPE OF THIRD OPERAND.

BIT PATTERN MEANINGS:

00 = SCALER ENTRY.
01 = ARRAY ENTRY.
10 = STRING ENTRY.
11 = INVALID OPERAND. (NOTE ERUNDF MAY BE SET IN THIS CASE.)

INPUTS: R0 = A PSEUDO STACK POINTER, TAG ADDRESS-1.
R4 = HOLDING AREA ACCUMULATING INFORMATION ABOUT OPERANDS.

OUTPUTS: R0 = UPDATED IF A VALID STACK ENTRY WAS FOUND.
R4 = THE HIGH ORDER BYTE IS SHIFTED RIGHT TWO BITS AND INFORMATION ABOUT THE TYPE OF THE FOUND OPERAND IS PLACED IN BITS 4 AND 5.

USES: R0,R1, R2, R3, R4

CALLS: MATSIZ

RESTRICTIONS, NOTES: IF A POINTER TO STRING ENTRY IS ON THE STACK MEMORY COMPRESS CAN'T BE CALLED.

TYPE

FUNCTION: PREFORMS THE TYP(N) FUNCTION

INPUT: N = FILE NUMBER IS ON THE STACK.

OUTPUT: TYP (N) = IS RETURNED ON THE STACK.

USES: R0

CALLS: A9X
CRTRST
FIX 1
FLOAT 1
INITMT
MTOPEN
MTRBFR
PSHRET
RTRN
TYPFIL

TYPRES (TYPE OF RESULT)

FUNCTION: TEST FOR VALID RESULT AREA ON STACK. THIS ROUTINE WORKS IN CONJUNCTION WITH TYPARG TO TEST FOR A VALID STACK ENTRY TO STORE THE RESULT OF A FUNCTION. THE TYPE OF ITEM FOUND IS NOTED IN R4 FOR THE CALLER TO EXAMINE. IF AN ARRAY ENTRY IS FOUND THE END OF DATA ADDRESS AND SHAPE INFORMATION IS PLACED IN THE STACK ENTRY FOR THE ARRAY. IF A STRING IS FOUND AND IT IS NOT DIMENSIONED TYPRES WILL DIMENSION IT TO 72 BYTES.

INPUTS: R0 = A PSEUDO STACK POINTER (ADDRESS OF TAG-1).
R4 = WORK AREA FOR BIT PATERN DESCRIBING WHAT WAS FOUND.

OUTPUTS: R4 = BITS 6 AND 7 ARE SET TO DESCRIBE WHAT HAPPENED.

USES: R0, R4, AND LSP.

CALLS: MATSIZ, DISBLE AND ENABLE.

RESTRICTION, NOTES: IF TYPRES ALLOCATES A STRING, STACK SPACE IS USED.

UNADR

FUNCTION: RESET I/O SYSTEM AFTER AN I/O STATEMENT, GETS OFF OF GPIB.

INPUTS:
```
IOFUNC
A,STAT
A,PRIM
CRSTAT
```

OUTPUTS:
```
A,STAT=0
YAXIS=0
IOFLGS=0
PPMODE=0
NODOUT=0
A,PRIM=255,
EOLCHR=15, <CR>
ISR = 65535, ; FOR ON PAGE
```

CALLS:
```
MEMBER
ATNON
INTSRC
IFCSND
IFCOFF
CRTRST
```

NOTE: IF DOING INPUT THEN THIS ROUTINE WILL SCAN FOR AN END OF RECORD IF ONE WAS NOT ALREADY ENCOUNTERED AS INDICATED BY CRSTAT.

UP

FUNCTION: A*B

INPUTS: B ON TOP OF STACK
A NEXT ON STACK

OUTPUT: A*B ON STACK UNLESS:

1. 0*0 = RESULT 0 AND ERRCD:= ERUP

2. 0*B = RESULT 0 (NO ERROR GIVEN, EVEN IF B<0\)

3. A*0 = RESULT 1 (NO ERROR)

4. IF B IS AN INTEGER <256
THEN A*ABS(B) IS COMPUTED BY FORMING POWERS OF A, E.G.
A, A*2, A*4, A*8, AND THEN MULTIPLYING THEM TOGETHER AS
NECESSARY. THEN IF B<0 THE INVERSE IS TAKEN. NO
CHECKS FOR OVERFLOW OR UNDERFLOW ARE MADE BY ERUP;
UNDERFLOWS WILL BE SET TO 0, OVERFLOWS TO THE LARGEST
REAL AND ERRCD:=ERFPOV

5. IF B IS NOT AN INTEGER <256 THEN
. 5.1 IF A<0 THEN BOTH OPERANDS ARE LEFT ON STACK AND
ERRCD:=ERUPN
. 5.2 OTHERWISE A*B = EXP (B*LOGE(A)). OVER/UNDERFLOWS
ARE HANDLED AND ZZETOX
. IF ANY OVERFLOW ERRORS ARE NOTED THEN
. ERRCD:=ERVP.

USES: R3
R1, R1+1

CALLS: PSHRET
A9X
PSHFPN
FPMUL
DOFP
RTRN
LOG
ZZETOX

UPCASE

FUNCTION: UPCASES ANY INPUT CHARACTER,
LOWER CASE A-Z TO UPPER CASE A-Z.
RIGHT BRACE TO)
LEFT BRACE TO (
RIGHT BRACKET TO)
LEFT BRACKET TO (

INPUTS: ACC-A HAS ASCII INPUT CODE.

OUTPUTS: ACC-B HAS ASCII OUTPUT CODE.

RESTRICION, NOTES: TESTCS IS A SPECIAL ENTRY POINT TO TEST THE STATE OF THE CASE/NOCASE SYSTEM STATUS AND EXIT OR FALL INTO UPCASE ACCORDINGLY.

VECTOR

FUNCTION: PUT VECTORS ON THE SCREEN

INPUT: TMP1:0 - DRAW
1 - MOVE

TMPHY: 2 BYTES <10 BITS> Y DATA - SCREEN POINTS

TMPHX: 2 BYTES <10 BITS> X DATA - SCREEN POINTS

CALLS: DRBUSY
WAIT.

NOTES: DISPLAY MUST BE ENABLED (NOT IN MAGTAPE MODE)

WRISTG

FUNCTION: WRITES <WRITE> A STRING ON TO THE CURRENT I/O DEVICE.

INPUTS: POINT : POINTS TO BEGINNING OF STRING
LENGTH : LENGTH OF STRING

OUTPUTS: ERRCD:SET IF ERROR

USES: R0

CALLS: BINOUT

NOTES: ALL NORMAL I/O SYSTEM VARIABLES MUST BE SET UP CORRECTLY.

WRIVAL

FUNCTION: WRITES <WRITE> A VALUE ONTO THE CURRENT I/O DIVICE

INPUTS: POINT : POINTS TO VALUE

OUTPUTS : ERRCD : SET IF ERROR IN DEVICE HANDLERS.

USES: A,PTR
A,END

CALLS: BINOUT
A7X

XFRCTL

FUNCTION: ATTACH AN I/O HANDLER FOR THE PRESENT A.PRIM DEVICE. WILL PERFORM SEARCH FOR DEVICE HANDLERS IN ROM PACKS.

INPUTS: A.PRIM ; PRIMARY ADDRESS

INDEX REG. ; POINTING TO A TABLE AS DEFINED BELOW

A.PRIM=0 - INDICATES FILE HANDLER

1-30 - INDICATES IEC BUS HANDLER

31 - INDICATES KEYBOARD HANDLER

32 - INDICATES DISPLAY HANDLER

33 - INDICATES MAGTAPE HANDLER

34 - INDICATES DATA STATEMENT HANDLER

35 - INDICATES 2ND MAGTAPE HANDLER

36-255. WILL RUN PIATBL FOR HANDLER.

OUTPUT: ERRCD ; SET IF CAN"T FIND DEVICE HANDLER OR DEVICE HANDLER CAUSES AN ERROR.

USES: BANK
PIATBL

CALLS: JMPX
NIODEV
A6X
SETBNK