Quick Reference Guide

intelligent graphics 2647A

HEWLETT PACKARD

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright © 1978 by HEWLETT-PACKARD COMPANY

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

TABLE OF CONTENTS

BASIC INTERPRETER

Introduction ... 1
  General Description ... 1
  Installing Terminal BASIC ... 1
  Using Terminal BASIC ... 2
  Direct Computation ... 4
Commands ... 5
Statements ... 15

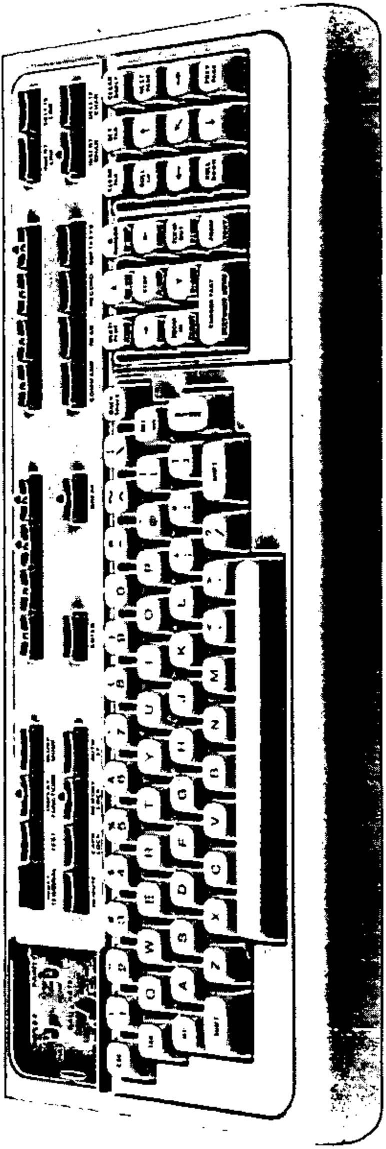
Computer Museum

A GRAPHICS LANGUAGE (AGL)

Introduction ... 43
AGL Terminology ... 45
Function Groups ... 53
Function Syntax ... 54
Set-Up Functions ... 55
Axis-Labeling Functions ... 56
Plotting Functions ... 80
Interactive Functions ... 87

TERMINAL FUNCTIONS

Alphanumeric Display Control ... A-1
Editing ... A-3
Terminal Control Group ... A-3
Additional Functions ... A-4
Terminal Control ... A-5
Graphics Control Sequences ... A-8
Display Control ... A-8
Graphics Label ... A-9
Vector Drawing Mode ... A-10
Plotting Commands ... A-11
Graphics Status ... A-12
Compatibility Mode ... A-13
ASCII Code Chart ... A-14
Device Control ... A-16
Multiplot ... A-35



PREFACE

2647A INTELLIGENT GRAPHICS TERMINAL

This quick reference guide contains a brief description of the terminal's functions. The functions are divided into three major groups:

- BASIC Interpreter
- Graphics functions (AGL)
- Terminal functions

BASIC INTERPRETER

The *BASIC Interpreter* allows you to use a powerful version of the BASIC language to write and execute application programs. These programs can interact with programs on a host computer or run independently with the terminal offline.

GRAPHICS FUNCTIONS

The terminal also contains a set of special *Graphics Functions* (AGL) that can be called from your BASIC program. The AGL functions provide high-level access to the terminal's graphics capabilities as well as scaling and labeling.

TERMINAL FUNCTIONS

Terminal functions consist of the various key and escape sequences that can be used to control the terminal directly. These include such operations as cursor, I/O device, and graphics control.

BASIC INTERPRETER

INTRODUCTION

GENERAL DESCRIPTION

Terminal BASIC provides you with the ability to do local programming in the BASIC language without a computer. Terminal BASIC is made up of a powerful and easy to use instruction set that contains most of the standard BASIC statements together with statements that allow your program to monitor and control terminal operation. Your program can interact with the terminal and its peripheral devices to perform a variety of application tasks. Programs can be written and run locally or in conjunction with a remote computer. The following modes of terminal operation are possible:

- Local operation without BASIC
- Remote operation without BASIC
- Local operation with BASIC
- Remote operation with BASIC

The first two modes of operation are explained in the terminal User and Reference manuals. This manual describes the remaining two modes of operation using BASIC.

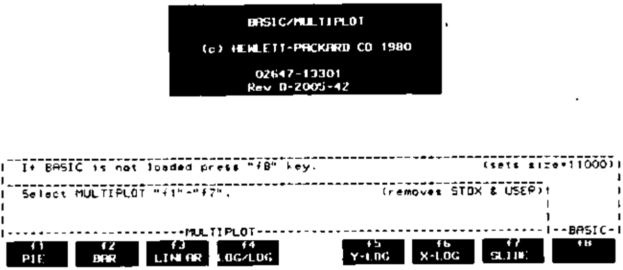
INSTALLING TERMINAL BASIC

To install Terminal BASIC, perform the following steps:

1. Place the Terminal BASIC/MULTIPLOT tape in the left tape drive.

2. Make sure that the REMOTE key is in the "up" position. (The terminal must be set for local operation.)

3. Press the READ key. This will cause the following display to appear on the screen:



```
BASIC/MULTIPLOT

(c) HEWLETT-PACKARD CO 1980

02647-13301
Rev B-2005-42
```

```
If BASIC is not loaded press "f8" key.                    (sets size=11000)

Select MULTIPLOT "f1"-"f7".                     (removes STDX & USEP)
```

4. To load BASIC press the F8 key. This will cause the BASIC Interpreter to be loaded into the terminal. The BASIC Interpreter will begin immediately by displaying the amount of workspace available for your program, followed by the BASIC prompt character ">".

Note that if the terminal is subsequently turned off, BASIC must be reloaded using steps 1 through 4.

USING TERMINAL BASIC

If the BASIC Interpreter is already loaded, you can access it by pressing the COMMAND key and typing "BASIC ⏎". This will cause the BASIC prompt ">" to appear. You can then enter BASIC commands or statements.

BASIC does not process input until the RETURN key is pressed. When the RETURN key is pressed, the line containing the cursor is read by the Interpreter. This allows you to edit the line using the cursor keys and the terminal editing functions before sending the statement to the Interpreter. Programs can be loaded from the keyboard, cartridge tape units, or remote computer.

If the BASIC statement is entered without line numbers it will be executed immediately. (Refer to Direct Computation.)

The terminal should be set for local operation (REMOTE key up) during program generation. Otherwise keyboard input will be sent out over the datacomm line to the computer and may cause undesired actions on the host computer.

If the terminal is set for local operation (REMOTE key up) the BASIC program can still request input from or send data to the datacomm line (see GETDCM and PUTDCM). While a program is running, keyboard input is monitored for the BASIC "break" character.

A running BASIC program can be halted with the "break" character (normally CONTROL-A). This has the same effect as a STOP statement. Direct computation can be performed while the program is stopped. You can then resume execution from the point at which the break occurred by entering the GO command.

When a program finishes execution (break character, STOP or END statements), the last values assigned to program variables are available for direct computation. The values for these variables are lost when you enter the next RUN command.

You can return to normal terminal operation by entering the EXIT command. Pressing the RESET key twice (full reset) will also cause normal terminal operation to be restored.

DIRECT COMPUTATION

Terminal BASIC can be used for direct computation without the need for a program. Numeric operations can be performed by entering a "?" followed by the numeric expression. Entering most BASIC language expressions or statements without a statement number will cause the operation to be executed as soon as the RETURN key is pressed.

Example: Calculate (2*(4+2)↑3).

```
?2*(4+2)^3 RETURN
432
```

Example: Set X=3, Y=−8*X, and print the absolute value of X*Y.

```
LET X=3 RETURN
LET Y=-8*X RETURN
PRINT ABS(X*Y) RETURN
72
```

The following statements cannot be used in direct mode:

```
CALL
DATA
FOR...NEXT
GOSUB
GOTO
IMAGE
ON...GOTO
ON ERROR
READ
RESTORE
RESUME
RETURN
SUB
SUBEND
WAKEUP
```

Note that when variables are assigned values using the LET statement, the values are not displayed. Only a PRINT statement or expression evaluation causes the output to be displayed. Statements or expressions entered without statement numbers are not stored. Values assigned to variables are lost if a program is subsequently run or if you exit BASIC. (You cannot assign a value to a variable in computational mode and pass it to a program run afterward.) You can pass direct computation values by using the GO command to resume a program interrupted by the break character or a STOP statement.

COMMANDS

Commands can be entered whenever a program is not executing. The EXIT and SET commands can be used as statements in a BASIC program. The SET SIZE version of the SET command cannot be used in a program.

The BASIC commands allow you to load and store program files, manipulate program listings, establish default parameters for the BASIC Interpreter, execute programs, and to return to normal terminal operation. The available BASIC commands are:

- AUTO
- CSAVE
- DELETE
- EXIT
- EXTEND
- GET
- GO
- LIST
- MERGE
- REMOVE
- RENUM
- RUN
- SAVE
- SCRATCH
- SET

AUTO

AUTO [starting line [,[increment]]]

The AUTO command causes the Interpreter to generate line numbers automatically as the program is entered. The numbering will begin with the starting line number given in the command. If no starting line number is given, the numbering will begin with line "10". If an increment is given, the line numbers will be stepped by this value. If no increment is given, the line numbers will be stepped by 10. If only the comma is entered, (with or without a starting line number), the program will be stepped using the last increment used by the AUTO command. (This value is set to 10 whenever BASIC is reloaded.)

If the AUTO command is used during program modification and generates a line number already present in the program, an error message is generated and AUTO is terminated. Entering the break character (Control-A) will turn off the autonumbering feature.

Example:

```
AUTO RETURN
10

AUTO 5,1 RETURN
5
```

CSAVE

CSAVE [range [TO]] [filename] [,SECURE]

The CSAVE command is similar to the SAVE command except that it causes a condensed version of the program to be kept. If SECURE is used, the kept program can be loaded but not listed by subsequent users.

Example:

```
CSAVE "RIGHT TAPE", SECURE Return
```

DELETE

DELETE [range]

The DELETE command allows you to delete lines from your program. The range gives the line numbers to be deleted.

<range> = first line to be deleted - last line to be deleted

If the last line number is not used, only the one indicated line will be deleted. If a starting line number is followed by a dash only, lines are deleted from the starting line to the end of the program.

Example: In the program resulting from the last RENUM example, a command of `DELETE 15-35` Return would result in:

```
10 LET X=0
40 END
```

The same result could be achieved by `DELETE 11,39` Return.

EXIT

EXIT

The EXIT command ends the BASIC Interpreter and restores normal terminal operation.

EXTEND

EXTEND <filename>

The EXTEND command allows you to add commands, statements, or functions to the Interpreter. These new commands and statements must be prepared according to a specific format. Refer to the BASIC Reference manual (part number 02647-90005) for a detailed description of the EXTEND command.

Example:

```
EXTEND "EDIT" RETURN
```

GET

GET [file name]

The GET command reads in a copy of a program from a file. If the file is not specified, the program is loaded from the last specified source device. The initial default source device is the left tape.

Examples:

```
GET RETURN
GET "RIGHT TAPE" RETURN
```

GO

GO

The GO command causes program execution to resume after the break character (normally a CONTROL—A) has been entered from the keyboard or a STOP statement has been executed. Execution resumes at the statement after the break occurred or following the STOP statement. If the terminal is waiting for input when a break occurs, the program will again request input. The input prompt (?) or string will also be reprinted.

The GO command is useful in debugging. It will allow you to interrupt an infinite loop, use direct computation statements to display or change variable values and then resume execution using the GO command.

Note that you cannot modify the program while the program is stopped and then resume execution. If statements are changed, you must rerun the program.

LIST

LIST [range] [[TO] filename]

The LIST command displays the current program in line number order. The range parameter can be used to list a sequential block of lines. If no range is specified, the entire program will be listed. The range parameter is of the following format:

<range> = first line to be listed - last line

The last line number is optional. If the last line number is not used, only the single specified line will be listed. If the first line is followed by a dash only, the program will be listed from the starting line to the end of the program.

Example:

```
LIST. 30-50
```

MERGE

MERGE [filename]

The MERGE command loads a copy of the specified file into the BASIC workspace. The new program is merged with any existing program. If there are conflicting line numbers, the old lines are replaced by the new ones. If a filename is not specified, the program is loaded from the current source device.

Example:

```
MERGE "LEFT TAPE" RETURN
```

REMOVE

REMOVE STD
or
REMOVE STDX
or
REMOVE USER

The REMOVE command is used to remove from the Interpreter any commands or statements that have been added using the EXTEND command. REMOVE STD removes the extensions prepared by Hewlett-Packard. This includes the AGL graphics statements.

REMOVE STDX removes the PRINT USING, PRINT # USING, IMAGE, CALL, SUB, SUBEND, COMMAND, and HP-IB statements from the BASIC Interpreter. In addition, error messages are abbreviated to error codes. The statements removed with REMOVE STDX provide an additional 5K bytes of user workspace.

REMOVE USER removes extensions prepared by the user.

Examples:

```
REMOVE STD ⏎
REMOVE USER ⏎
```

RENUM

RENUM [new starting line[,increment
[,old starting line [,old ending line]]]]

If the starting line number is not given, the first line will be numbered 10. The increment is optional and specifies the increment between line numbers. If no increment is given, a value of 10 will be used. Line numbers less than the old starting line number will not be renumbered. If the old starting line number is not specified, the program will be renumbered beginning with the first line. If an old ending line is specified, the renumber process will continue until the specified point in the program is reached. Line numbers after the old ending line are unchanged.

Example: Assume that the following program is present:

```
10 LET Y=0
11 LET X=0
17 INPUT X,Y
70 LET Z=X+Y
71 PRINT X,Y,Z
90 GOTO 17
999 END
```

Entering RENUM RETURN will cause the program to be renumbered to the following:

```
10 LET Y=0
20 LET X=0
30 INPUT X,Y
40 LET Z=X+Y
50 PRINT X,Y,Z
60 GOTO 30
70 END
```

Note that line numbers used in GOTO or GOSUB statements are automatically changed to the new line numbers.

Using the same program, RENUM 100 RETURN would cause the following:

```
100 LET X=0
110 LET Y=0
120 INPUT X,Y
130 LET Z=X+Y
140 PRINT X,Y,Z
150 GOTO 120
160 END
```

Specifying a statement increment, RENUM, 5 RETURN would result in the following:

```
10 LET X=0
15 LET Y=0
20 INPUT X,Y
25 LET Z=X+Y
30 PRINT X,Y,Z
35 GOTO 20
40 END
```

RUN

RUN [linenumber]
or
RUN filename [, linenumber]

The RUN command is used to execute your program. If the optional line number is not used, the program will begin executing from the first statement. If a line number is specified, program execution will begin with the indicated line.

When a filename is used, the program will first be loaded from the indicated file and then run. If no file name is specified and there is no program in the workspace, a program will be loaded from the current source file and then run.

Example: Execute the first program shown under the RENUM command beginning at the statement INPUT X,Y.

```
RUN 17↵
```

SAVE

SAVE [range] [[TO] filename] [,BASIC]

The SAVE command stores a copy of the current program on a file. If the file is not specified, the program is stored on the last specified destination device. The initial destination file is the right tape.

The BASIC option is used to save a copy of the current BASIC interpreter. This is useful when the interpreter has been modified using the EXTEND command. The range option can not be used with the BASIC option.

Examples:

```
SAVE ↵
SAVE "RIGHT TAPE", BASIC ↵
```

SCRATCH

SCRATCH

The SCRATCH command deletes the current program and variables from the workspace. SCRATCH can be abbreviated SCR.

Example:

```
SCRATCH↵
```

SET

SET <condition>

where <condition> is one of:

- **MULTIPLE**
- **SINGLE**
- **SHORT**
- **LONG**
- **INTEGER**
- **PROMPT**
- **KEY**
- **SIZE**

The SET command allows you to select the default modes of operation for the Interpreter.

The SET MULTIPLE command tells the BASIC interpreter whether or not to allow multiple statements per line number. The SET SINGLE command disables the interpreter's ability to use multiple statements in a line. The multiple statement capability can be reestablished with the SET MULTIPLE command or by reloading BASIC.

When the SET MULTIPLE command is entered, the interpreter will accept more than one statement in the same line. The statements must be separated by a "\" (backslash). The line has only a single line number at the left margin as for a normal statement. Multiple statements are executed in order from left to right. The multiple statment capability is initially on and is reset to on each time BASIC is entered.

```
100 READ A
200 PRINT A \ READ B \ PRINT B
300 FOR I=1 TO N \ A(I)=I^2 \ NEXT I
```

The main advantage of multiple statements is that you can save program storage space by not numbering each statement.

Example:

```
SET MULTIPLE RETURN
```

If multiple statements are entered and the multiple statement capability is not turned on, the entire line is rejected and an error message is displayed.

The SET SHORT, SET LONG, and SET INTEGER commands are used to set the default data type for numeric variables. The last default data type used continues in effect until changed. If BASIC is reloaded, the default type is set to SHORT.

Only one condition can be used in a SET command. For example, you cannot enter SET SINGLE, LONG.

Examples:

```
SET INTEGER
SET SINGLE
```

The SET PROMPT command allows you to change the prompt character. The default character is ">". Note that only a single character can be used.

Example:

```
SET PROMPT ="#"
```

The SET KEY command allows you to select the control character to be used as a local "break" character (a letter A–Z). Entering the "break" character will cause execution of the local BASIC program to stop. The default "break" character is CONTROL–A.

Example:

```
SET KEY ="B"
```

The SET SIZE command allows you to allocate terminal memory resources between the terminal display and the BASIC workspace. Decreasing the size of the display memory will increase the amount of memory available to your BASIC program. <size> is the number of bytes to be allocated to the workspace.

Note that the SET SIZE command cannot be executed from a BASIC program.

Example:

```
SET SIZE = 21000
```

STATEMENTS

This section contains descriptions of the statements available in Terminal BASIC. The statements are listed in alphabetic order. Each statement is shown with statement syntax, a brief description, and statement examples. Most statements can be used in direct computation mode. Some cannot. Refer to the BASIC Reference manual for additional information on direct computation mode.

Items in uppercase type must be included whenever the statement is used. The letters "LET" in the LET statement are an exception.

An ellipsis ("...") indicates items that can repeat indefinitely.

Lists consist of one or more of the items specified. Items within lists are separated by commas unless otherwise specified.

All punctuation marks (quotes, commas, colons, and semicolons) must be included.

ASSIGN

ASSIGN filename TO # filenumber
or
ASSIGN * TO # filenumber

The ASSIGN statement allows you to equate a filename to a logical filenumber. The filename can be a string or string expression. The filenumber is used by BASIC statements to select the file to be used in a file operation. This means that your program is independent of the name used for a data file. The file number must be between 1 and 127.

You can reuse the program with a different filename simply by changing the filename used in the ASSIGN statement. The ASSIGN statement allows you to reassign files while the program is running so that several files can be acted on in succession.

```
200 ASSIGN "DATA1" TO #3
300 ASSIGN "LEFT TAPE" TO #1
400 ASSIGN * TO #2
500 ASSIGN "TE*10" TO #1
600 ASSIGN "H#6" TO #2
```

CALL

CALL <subprogram name> [(<parameters>)]

The CALL statement is used to transfer control to a subprogram. The parameters can be numeric or string variables or expressions, or they can be logical file numbers. When a CALL statement is executed, the values of the CALL parameters are equated to similar variables in the SUB statement. (A SUB statement must be the first statement in the CALLed program.) Note that the variables in both the CALL statement and the SUB statement must be of the same type and order. Refer to the SUB statement for additional information.

```
10 FOR I=1 TO 10
20 INPUT A$
30 CALL TEXT(I,A$)
40 NEXT I
  .
  .
  .
100 SUB TEXT(J,B$)
110 PRINT "LINE#";J;TAB(10);B$
120 SUBEND
```

COMMAND

COMMAND commandstring [,variable]

The COMMAND statement allows you to execute terminal commands from your BASIC program. If a variable is specified, the execution status of the command is returned in the variable. If the return variable is not used and the command does not execute properly, your program will be terminated. If the variable is used and the command executes properly, the variable will be set to zero before execution of the next program statement. On failure of the command the variable will be set to one of several error codes depending on the nature of the error. Refer to the BASIC manual for a description of the error codes.

If quote characters are required within the command string, replace them with the apostrophe character (').

Examples:

```
10 COMMAND "RE L"    ! REWIND THE LEFT TAPE
20 COMMAND A$
```

DATA

DATA datalist

Provides data to be read by READ statements. DATA statements may be located anywhere in the program; all DATA statements in a program are considered part of a single continuous list of data for that program. Data items may be numeric or string constants.

```
10 DATA 10
20 DATA "CALIFORNIA"
30 DATA 405,"OREGON",999
```

DIM

DIM itemlist [stringlength or arraysize]

Specifies the maximum length of strings and the maximum array size of numeric or string variables. String lengths are enclosed in brackets; array dimensions in parentheses. Numeric arrays declared in DIM statements assume the default numeric type. (Refer to LONG, INTEGER, and SHORT for additional information.)

```
70 DIM A$[25]
90 DIM K(18)
130 DIM P$(11,11)[25]
```

END

END

Terminates execution of the program. The END statement can be placed anywhere in a program except following a SUBEND statement.

```
90  END
```

ERROR

ERROR errornumber [, message]

This statement causes an immediate branch to the error handling routine. The error code placed in ERRN is the integer value of the errornumber parameter. Errornumber can be any integer expression. The ERROR statement can be used to cause application error conditions to be acted on by the same routines that would normally handle programming errors. Make sure that the error numbers that you use are greater than those used by the BASIC Interpreter or you will be unable to tell an application error from a programming error. (See the ON ERROR statement for a list of error numbers.) If no message exists for the error, a special message will be printed to indicate that there is no standard error message.

The optional message can be any string expression. If the message parameter is used, the user defined message will be displayed, replacing any standard message.

```
100 ERROR N+40
```

FOR...NEXT

FOR variable = initial TO final [STEP size]

```
  .
  .
  .
```

NEXT variable

The FOR...NEXT statements allow repetition of a group of statements between FOR and NEXT. The number of repetitions is determined by the initial and final values of a loop variable, and by an optional STEP size. The loop variable cannot be type LONG.

```
110 FOR I=1 TO 5
120 FOR J=1 TO -3 STEP -1
130 PRINT J
140 NEXT J
150 NEXT I
```

GETDCM ON/OFF

GETDCM ON
or
GETDCM OFF

The GETDCM statement enables and disables the operation of the GETDCM function. GETDCM ON enables the GETDCM function (described later in this manual) and disables interrupts from the interrupt keys. (Refer to SET KEY and ON KEY for additional information on interrupt keys.) GETDCM OFF disables the GETDCM function and enables the interrupt keys.

```
10 GETDCM ON
```

GETKBD ON/OFF

GETKBD ON
or
GETKBD OFF

The GETKBD statement enables and disables the operation of the GETKBD function. GETKBD ON enables the GETKBD function and disables program interrupts from the keyboard. (Refer to SET KEY and ON KEY for additional information.) GETKBD OFF disables the GETKBD function and enables the interrupt keys.

```
20 GETKBD ON
```

GOSUB

GOSUB line number

Transfers control to the specified statement. The line number must refer to the first statement of a subroutine within the current program unit. A RETURN statement is required in the subroutine to return control to the statement following the GOSUB statement.

The GOSUB statement differs from the CALL statement in that the GOSUB statement can not pass parameters to the subroutine.

```
80 GOSUB 1000
90 REM CONTINUE PROCESSING
  .
  .
  .
1000 REM: START OF SUBROUTINE
1010 RETURN
```

GOTO

GOTO line number

Transfers control to the specified statement in the current program unit.

```
160 GOTO 200
```

IF...THEN...ELSE

IF expression THEN statement [ELSE statement]

Evaluates a conditional expression and specifies an action to be taken if the condition is true. A conditional expression is considered true if its value is non−zero, false if its value is zero. The action may be a transfer to a line number or one or more executable statements. Refer to the BASIC Reference manual for detailed information on the use of multiple statements with IF...THEN...ELSE statements.

```
180 IF A<0 THEN 400
190 IF A=B THEN PRINT B
```

IMAGE

IMAGE formatstring

The IMAGE statement provides format specifications to be used in a PRINT USING statement. The PRINT USING statement must reference the line number of the IMAGE statement to be used.

```
200 PRINT USING 300; A$,B,C$
300 IMAGE "MONTH",3A,DD.DD,4A
```

INPUT

INPUT [“message”,] variablelist

The INPUT statement requests input to one or more variables by displaying a “?” prompt. The program will then accept string or numeric data from the keyboard. (See also KEYCDE.) If an optional message is used, the message replaces the “?” prompt. Whether a basic prompt or message appears on the screen, input of data begins one column beyond the prompt or message. The basic prompt or message is not input with the data.

```
10 INPUT A$,B
20 input "ENTER TIME",T
```

INTEGER

INTEGER itemlist (arraysize)

Declares that the variables in the itemlist are type INTEGER. If arraysize is specified, the maximum size for the array is also set. Integers must be in the range −32K to +32K.

```
20 INTEGER B(20)
```

KEYCDE

KEYCDE (X, keynumber, new keycode)

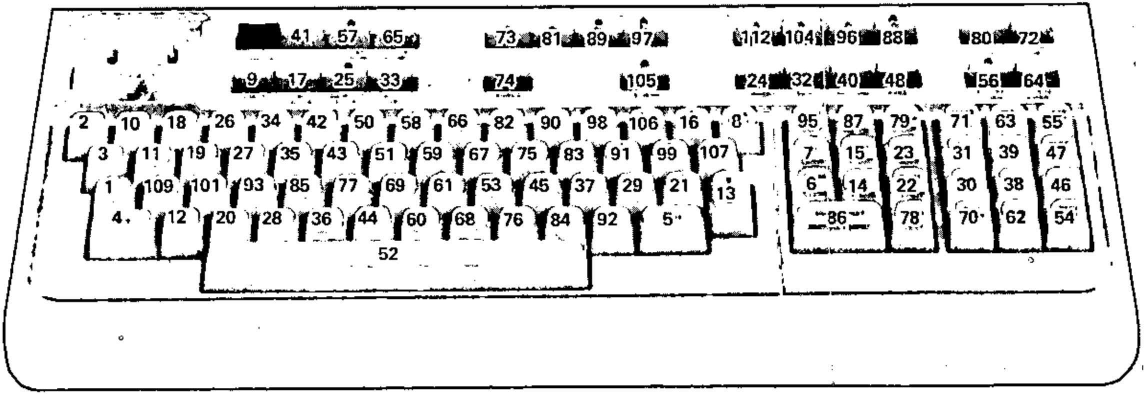
KEYCDE assigns the code values to be used for decoding the keyboard. Each of the keys on the keyboard is mapped to a code value (normally ASCII). The keys are numbered according to the diagram shown in figure 3—1. When a key is pressed, its location number is returned to the terminal. The number is used to select the proper ASCII code out of a table. For example, key number 109 would normally be the "A" key. Element U(109) would be equal to 65 (the ASCII decimal code for A).

The valid key numbers are 0—112. Key codes can be between 0 and 255. The left and right shift keys, CNTL, AUTO LF, BLOCK MODE, CAPS LOCK, REMOTE, and RESET keys cannot be redefined.

There are three possible code tables from which the key code can be obtained. The X parameter in the statement is used to indicate which one of the tables is being defined with the current array. X selects the key code table as follows:

X = 0 Unshift character codes
1 Left shift key characters
2 Right shift key characters
3 Left and right shift characters (defines both shift tables)

When the left shift key (#4 in figure 3—1) is pressed in addition to the character key, the left shift array will be used to select the character code. The right shift key (#5 in figure 3—1) will cause the character to be selected from the right shift table. Table 3—1 lists the default values for the key code tables.



UNUSED KEYNUMBERS: 49 94 100 102 103 108 110 111

Figure 3—1. Terminal Keyboard Layout

Table 3—1. Default Keyboard Codes

Key #	UNSHIFT		SHIFT	
	Code	Key	Code	Key
1	0	CNTL	same	
2	27	ESC	same	
3	9	TAB	same	
4	0	L.SHIFT	same	
5	0	R.SHIFT	same	
6	139	ZOOM IN	141	CLEAR
7	165	L.G.CURSOR	144	DRAW
8	8	BACKSPACE	same	
9	134	REMOTE	same	
10	49	1	33	!
11	113	q	81	Q
12	122	z	90	Z
13	239	▬	same	
14	164	D.G.CURSOR	150	T ANG
15	135	STOP	143	A DSP
16	92	\	124	¦
17	131	CAPS LOCK	same	
18	50	2	34	"
19	119	w	87	W
20	120	x	88	X
21	93	]	125	}
22	140	ZOOM OUT	151	T SZE
23	163	R.G.CURSOR	145	MOVE
24	155	COMMAND	same	
25	128	MEMORY LOCK	same	
26	51	3	35	#
27	101	e	69	E
28	99	c	67	C
29	58	:	42	*
30	196	L ARROW	same	
31	211	ROLL UP	same	
32	156	READ	same	
33	132	AUTO LF	same	
34	52	4	36	$
35	114	r	82	R
36	118	v	86	V
37	59	;	43	+
38	232	HOME UP	same	
39	193	UP ARROW	same	
40	157	RECORD	same	
41	250	TEST	same	
42	53	5	37	%
43	116	t	84	T
44	98	b	66	B
45	108	l	76	L
46	195	R ARROW	same	
47	213	NEXT PAGE	same	
48	158	SOFTKEYS	same	
49	0	(not used)		
50	54	6	38	&
51	121	y	89	Y
52	32	SPACE	same	
53	107	k	75	K
54	214	PREV PAGE	same	
55	202	CLEAR DSPLY	same	
56	130	INSERT CHAR	same	

Table 3–1. Default Keyboard Codes

Key #	UNSHIFT		SHIFT	
	Code	Key	Code	Key
57	129	DSPLY FNCTNS	same	
58	55	7	39	'
59	117	u	85	U
60	110	n	78	N
61	106	j	74	J
62	194	DOWN ARROW	same	
63	177	SET TAB	same	
64	208	DELETE CHAR	same	
65	133	BLOCK MODE	same	
66	56	8	40	(
67	105	i	73	I
68	109	m	77	M
69	104	h	72	H
70	212	ROLL DOWN	same	
71	178	CLEAR TAB	same	
72	205	DELETE LINE	same	
73	240	F1	same	
74	152	ENTER	same	
75	111	o	79	O
76	44	,	60	<
77	103	g	71	G
78	138	ZOOM	149	TEXT
79	136	G.CURSOR	142	G.DISPLAY
80	204	INSERT LINE	same	
81	241	F2	same	
82	57	9	41	)
83	112	p	80	P
84	46	.	62	>
85	102	f	70	F
86	166	CURSOR FAST	146	MULT MENU
87	162	U.G.CURSOR	137	RB LN
88	247	F8	same	
89	242	F3	same	
90	48	0	same	
91	64	@	96	`
92	47	/	63	?
93	100	d	68	D
94	0	(not used)		
95	147	MULTPLOT	148	AXES
96	246	F7	same	
97	243	F4	same	
98	45	–	61	=
99	91	[	123	{
100	0	(not used)		
101	115	s	83	S
102	0	(not used)		
103	0	(not used)		
104	245	F6	same	
105	153	BREAK	same	
106	94	^	126	~
107	95	_	127	●
108	0	(not used)		
109	97	a	65	A
110	0	(not used)		
111	0	(not used)		
112	244	F5	same	

LET

LET variablelist = expression

The let statement assigns a value to a variable. The keyword LET may be omitted. Multiple variables can be assigned a common value by using a comma to separate the variables.

```
10 LET X = 10
20 Y=20
30 X,Y = 30
```

LINPUT

LINPUT ["message",] stringvariable

An entire line of input is accepted as a single string. The data can be assigned to a string or substring. Extra characters are truncated if the destination string is not large enough. If the optional message is used, the message will be printed on the display in place of the prompt. Whether a basic prompt or message appears on the screen, input of data begins one column beyond the last prompt or message. The basic prompt or message is not input with the data.

```
30 LINPUT H$
90 LINPUT "Name:", N$
```

LINPUT

LINPUT # filenumber [BYTE count]; stringvariable

The LINPUT # statement is similar to LINPUT except that the line of input data is read from the file specified by the file number. File numbers are assigned using the ASSIGN statement.

If the optional byte count is used, only the number of bytes (characters) specified will be read into the string variable. If a byte count of 0 is given, the length of the string as determined from dimension or data type declarations will be used to determine the number of characters to be input.

```
10 LINPUT #3;B$
20 LINPUT #4 BYTE 30; A$
```

LONG

LONG variablelist

The LONG statement is used to declare the items in the variable list to be of type LONG.

```
600 LONG A,B(4,4)
```

NEXT(See FOR...NEXT)

OFF KEY

OFF KEY # keycode

Disables any interrupt associated with the indicated key. The key codes are shown in figure 3−1 (see KEYCDE). Refer to ON KEY # for additional information.

```
10 OFF KEY # 32
```

ON END

ON END # filenumber CALL subprogram
or
ON END # filenumber GOSUB linenumber
or
ON END # filenumber GOTO linenumber

The ON END statement causes a transfer of control to the specified statement or subprogram when an end of file condition is detected. Note that the transfer will occur for the specified file only. The ON END statement must be executed prior to the end of file condition in order to be effective. Subsequent ON END statements can change the destination subprogram or linenumber. (Refer to the ON ERROR statement for additional information.)

The ON END # statement has effect only within the current program unit. It will not affect operation of a subsequently called subprogram. A new ON END # statement must be executed in each subprogram.

If an end of file condition is reached and an ON END statement has not been previously executed for the file, the program will be terminated with an error.

```
10 ON END # 1 CALL Help
20 ON END # 2 GOSUB 300
30 ON END # 3 GOTO  500
 .
110 PRINT #1;X,Y
120 READ #2;A$
130 PRINT #3;A$,X,Y
 .
200 SUB Help
210 PRINT "NEED BIGGER FILE";
220 SUBEND
```

ON ... GOSUB

ON expression GOSUB linelist

Causes the program to execute a subroutine at one of several line numbers. The expression is evaluated and rounded to an integer. This integer selects the first, second, or nth line number from the linelist. If the integer less than 1 or greater than the number of lines in the linelist, the statement following the ON ... GOSUB statement is executed instead. If the integer is less than 0 or greater than 255, an error will result.

Upon return from the selected subroutine the program will continue execution with the statement following the ON...GOSUB statement.

```
10 ON N+M GOSUB 300,400,500,600,700
```

ON ... GOTO

ON expression GOTO linelist

Causes a branch to one of several statements. The expression is evaluated and rounded to an integer. The integer is then used to select a statement from the line list in the same manner as the ON...GOSUB statement. The integer must be between 0 and 255 otherwise an error will result. If the integer is 0 or exceeds the number of line numbers in the line list, the statement following the ON...GOTO statement will be executed.

```
400 ON N+2 GOTO 100,200,300
```

ON ERROR

ON ERROR GOTO linenumber

When an error is detected by the Interpreter, a branch to the indicated line number will occur. The type of error can then be examined and the appropriate action taken.

Once an ON ERROR statement has been executed, all errors will cause a branch to the indicated error handling routine. If the line number given is not valid, a special error message will be generated.

```
10 ON ERROR GOTO 1000
```

If you specify a line number of 0, the ON ERROR branch is disabled and subsequent errors will cause the program to halt and an error message to be printed. If the ON ERROR GOTO 0 statement is contained in the error handling routine itself, it will cause the program to stop and print the error message. You should use ON ERROR GOTO 0 in error handling routines when there is no recovery procedure for a particular error.

If an error occurs within the error handling routine, the program will halt and an error message for the error occuring in the error handling routine will be printed. You cannot trap errors that occur within the error handling routine.

The ON ERROR GOTO statement must be executed before the error occurs. You can test for the type and location of the error using the error variables ERRL and ERRN. ERRL contains the line number in which the error occurred. If there is no error, ERRL will be 0. In Direct Computation mode ERRL will always be 0. ERRN contains the code number of the error. If there is no error ERRN will be 0.

ERRL and ERRN cannot be used on the left side of the "=" sign in a LET or assignment statement.

The following example shows how a simple error handling routine would work.

```
200 ON ERROR GOTO 400
210 INPUT "WHAT ARE THE VALUES TO DIVIDE?";X,Y
220 Z=X/Y
230 PRINT "THE QUOTIENT IS ";Z
240 GOTO 210
400 IF ERRN=1034 AND ERRL=220 THEN 420
410 ON ERROR GOTO 0
420 PRINT "DIVIDE BY ZERO ERROR"
430 RESUME 210
```

ON KEY

ON KEY # keycode CALL subprogram
or
ON KEY # keycode GOSUB linenumber
or
ON KEY # keycode GOTO linenumber

The ON KEY # statement provides a program interrupt when the indicated key is pressed. The default keycodes are listed in table 3–1 (see KEYCDE). When the selected key is pressed, the program will perform the specified transfer as soon as the current statement has completed execution. The RETURN from the GOSUB will return the program to the statement following the point where the key interrupt occurred. Interrupt keys are not detected if they are entered as part of the response to an input or linput statement.

```
10 ON KEY # 27 CALL Gotcha
20 ON KEY # 32 GOSUB 400
30 ON KEY # 12 GOTO 100
```

PRINT

PRINT printlist

Prints the values of a list of expressions on the display device. Items in the list may be separated by commas or semicolons. Commas space output in 15–character fields. Semicolons cause output to be printed without extra spaces. A comma or semicolon at end of print list suppresses the final carriage return and linefeed. The print list may also include print functions to control output format.

```
480 PRINT A,5,8/E;B[4]&C$&"Q-9*",
490 PRINT "NAME",SPA(3);"ADDRESS"
```

PRINT

PRINT # filenumber [BYTE count]; printlist
or
PRINT # filenumber [BYTE count] USING format; printlist

PRINT # writes values of expressions in the printlist serially to the specified file. A record is sent to the file each time a line feed character is encountered' in the printlist (unless the BYTE option is used) and at the end of the printlist. If the printlist is ended with a comma or semicolon the data is sent to the specified file without carriage return and line feed characters at the end of the record.

The filenumber parameter is normally an integer between 1 and 127. If a filenumber of 0 is used the operation will be interpreted as an AGL operation. Refer to the AGL (A Graphics Language) description later in this manual for a description of PRINT #0.

Note that if you attempt to print 256 or more consecutive characters to a.given file, a buffer overflow error will result.

If the optional byte count is used, BASIC will transfer the specified number of bytes to the file. This allows the transfer of 8—bit data and suppresses transmission of records upon encountering the line feed character. If a byte count of 0 is given, the actual byte count will be determined from the print list items.

The PRINT # USING statement allows you to format data as it is written to the file. Refer to the PRINT USING statement.

```
10 PRINT #3; A$,R
30 PRINT #N; "HELLO"
40 PRINT #1 BYTE 24; A$,B$
50 PRINT #1 BYTE 80 USING 200; A,B,C$
```

Computer
Museum

PRINT USING

PRINT USING format ; printlist
or
PRINT USING image line ; printlist

The PRINT USING statement prints the values of the items in the printlist according to the specification given in the formatstring or in the referenced IMAGE statement. If the end of the format string is reached before all of the items in the printlist have been output, the format string is repeated.

The examples below show three ways of referencing format strings. The first (530) contains the format string in the PRINT USING statement itself. The second (540) uses a string variable to reference the format string. The third refers to an IMAGE statement containing the format string.

```
530 PRINT USING "3A,3A";A$,"ABC"
540 PRINT USING F$;N,M,C$
550 PRINT USING 200;A,B,D
```

Format Symbols

The format string used in PRINT USING and IMAGE statements is made up of the following format control characters:

A — ASCII character
X — Blank
S — Numeric sign character (+ or −)
M — Minus sign
D — Numeric digit (0—9) with blank fill
. — Decimal point
R — Decimal point using a "," character
C — Comma
P — Period
K — Compressed format
/ — Separates specifications and gives CR/LF
() — Delimits a group specification
, — Separates specifications
@ — Separates specifications and causes a top —of —form
"" — Encloses literal specifications

READ

READ variablelist

The READ statement assigns numbers and strings from one or more DATA statements to the variables specified in the variable list.

```
570 READ X,A$,B$,Y
575 READ A(N,M),B(N,M)
```

READ

READ # filenumber; variablelist

The READ # statement assigns numbers and strings from the specified file to the variables in the variable list. Reads the file serially beginning at the current position of the file pointer. The file is read until the variable list is exhausted. The read may cross record boundaries. Entries in the file must be separated by commas.

```
580 READ #5; A, B$
595 READ #1; A(N,M)
```

REMARK

REM comments

Allows you to place remarks within program listing. Any characters may be used. Note that the "!" character can be used following a statement in a line to indicate that the remaining characters on the line are comments.

```
640 REM ---ANY TEXT---
```

RESTORE

RESTORE [line number]

The RESTORE statement resets data pointer to the specified DATA statement. If the line number is omitted, the pointer is reset to the lowest numbered DATA statement in the current program unit.

```
650 RESTORE 60
```

RESTORE

RESTORE # filenumber

Repositions the file pointer to the beginning of the specified file.

```
660 RESTORE #2
```

RESUME

RESUME [linenumber]
or
RESUME NEXT

Causes the program to continue execution after an error recovery routine (ON ERROR) has been executed. You can resume execution at one of the following points:

- the statement that caused the error (no linenumber)
- the statement following the error (NEXT)
- some specified line number

To continue execution at the statement causing the error use the following statement:

```
40 RESUME
```

To continue execution at the statement following the error use the following statement:

```
40 RESUME NEXT
```

To continue execution at a specified line number use the following statement:

```
40 RESUME 500
```

Note that the linenumber parameter must be a valid line number (>0). Refer to the ON ERROR statement for additional information.

RETURN

RETURN

Returns control from a subroutine to the statement immediately following the most recently executed GOSUB statement.

Example:

```
100 GOSUB 200
110 REM CONTINUE PROGRAM
  .
  .
  .
200 REM Subroutine A
  .
  .
  .
220 RETURN
```

SHORT

SHORT itemlist (arraysize)

Declares the variables in the itemlist to be type SHORT. If arraysize is specified, the maximum size for the array is also set.

```
730 SHORT A,B(4,4)
```

SLEEP

SLEEP

The SLEEP statement allows you to place the Interpreter into a dormant state. Execution of the BASIC program is suspended and the terminal returns to normal operation. The terminal remains in the normal mode of operation until either the "break" key is entered (normally a CONTROL—A) or a key is pressed for which an ON KEY statement was executed in the suspended program. If the "break" key is used to return to the BASIC Interpreter, the suspended program will resume at the statement following the SLEEP statement.

If a keycode interrupt is used to return to BASIC, the suspended program will execute the statement(s) specified in the ON KEY statement. If the ON KEY ... GOSUB or the ON KEY ... CALL forms were used, the indicated subroutine or subprogram is executed and then the program will return to the SLEEP state. If the ON KEY ... GOTO form of statement was used, the program will not return to the SLEEP state unless a SLEEP statement is again executed.

The WAKEUP statement can also be used in an ON KEY subroutine or subprogram to cancel the current SLEEP statement.

Examples:

```
20 ON KEY # 160 GOSUB 200
30 ON KEY # 170 CALL Subproga
40 ON KEY # 180 GOTO 300
50 SLEEP
  .
200 PRINT "TEST1"
210 RETURN
300 STOP
  .
400 SUB Subproga
410 PRINT "TEST2"
420 WAKEUP
430 SUBEND
```

STOP

STOP

The STOP statement terminates program execution. The program can be resumed with the GO command.

```
740 STOP
```

SUB

SUB subprogramname [(parameters)]

The SUB statement must be the first statement in a subprogram. It provides the name of the subprogram and marks the entry point for the main or calling program. The SUB statement also lists any parameters that are to be passed between the calling program and the subprogram. The last statement in the subprogram must be a SUBEND statement.

```
300 READ #1 ;A$
400 C=NUM(A$[1,1])
410 IF C>48 AND C<57 THEN 440
420 CALL Alpha (C)
430 GOTO 300
440 CALL Numeric (C)
  .
  .
  .
500 SUB Alpha (N)
502 PRINT CHR$(N);
504 SUBEND
600 SUB Numeric (N)
602 K=K*10+N
604 SUBEND
```

SUBEND

SUBEND

The SUBEND statement is used to indicate the end of a subprogram. Refer to the SUB statement for additional information.

The SUBEND statement must be the only statement on the line. It cannot be used in multistatement lines or with the "!" form of comments or with the REMARK statement. The only statement that can follow the SUBEND statement is a SUB statement beginning another subprogram. If any other statement type is used, the message MISSING SUB will be displayed and the program will be halted.

```
10  SUB Test (N,K,A$)
  .
  .
  .
50  SUBEND
```

WAKEUP

WAKEUP

The WAKEUP statement cancels the effect of the last executed SLEEP statement.

```
400 WAKEUP
```

BUILT–IN FUNCTIONS

The following paragraphs describe the built–in functions available in BASIC. The functions are divided into the following categories:

- Numeric
- String
- Array
- Print
- Input
- Other

Within each category, the functions are listed in alphabetic order. The letter which appears after most function names specifies the type of result that is returned by the function. These letters are interpreted as follows:

S – short
I – integer
L – long
ST– string
T – same type as X

Numeric Functions

Function Type Description

ABS(X) T Absolute value of X.

```
10 A = ABS(2)! A = 2
20 B = ABS(-2)! B = 2
```

CSENA(R,C) I Returns the absolute row and column position of the cursor in R and C. C is 1 to 80 and R is 1 to 528. The function value is set to the number of the current user window (1-4).

```
10 A = CSENA(P,Q)
```

CSENS(R,C) I Returns the screen relative row and column position of the cursor in R and C. C is 1 to 80 and R is 1 to 24. The function value is set to the number of the current user window (1–4).

```
10 B = CSENS(A(1),B(1))
```

EXP(X) S The natural log e raised to the X power (E↑X).

```
70 G=EXP(2) ! G = E^2 = 7.39
```

INT(X) I Largest integer <= X.

```
80 H=INT(6.999)! H=6
```

LOG(X) S Natural logarithm ; X>0

```
100 J=LOG(2)
```

LONG(X) L Returns the LONG value of X.

```
120 R=LONG(Y)
```

RND S The next pseudo—random number in a standard sequence of numbers >= 0 and <1.

```
130 M=RND
```

SGN(X) I The sign of X; −1 if X<0, 0 if X=0, and +1 if X>0.

```
140 N=SGN(J)
```

SHORT(X) S X converted to SHORT representation

```
145 M=SHORT(N)
```

SQR(X) S The positive square root of X.

```
160 Q = SQR(169)
```

Trigonometric Functions

ATN(X) S Returns the arctangent of X in radians.

```
200 E = ATN(Y)
```

COS(X) S Returns the cosine of X where X is in radians.

```
300 X = R*COS(A)
```

SIN(X) S Returns the sine of X where X is in radians.

```
400 Y = R*SIN(A)
```

TAN(X) S Returns the tangent of X where X is in radians.

```
500 T = TAN(A)
```

String Functions

CHR$(X) ST Returns a string character with the value of the numeric expression X. The value of X must be between 0 and 255.

```
10 LET T$=CHR$(32) ! T$ = " " (blank)
```

LEN(S$) I Returns the number of ASCII characters in the string S$.

```
10 DIM A$[20]
20 LET A$ = "1234"
30 LET B  = LEN(A$)!  B = 4
```

NUM(S$) I Returns a numeric value between 0 and 255 corresponding to the first character of the string S$. S$ cannot be the null string "".

```
10  LET  T$  =  "Payroll"
20  LET  B  =  NUM(T$[2;1])!   B  =  97
```

POS(S1$,S2$) I Searches the string S1$ for the first occurrence of the string S2$. Returns the starting index if found, otherwise returns 0.

```
10 LET T$="Television"
20 LET V$ = "vision"
30 LET C = POS(T$,V$)  !  C = 5
```

VAL(S$) * Returns the numeric equivalent of the string S$. * — The data type returned may be integer, short, or long as required.

```
200 N = VAL(N$)+I
```

VAL$(X) ST Returns the value of the numeric expression X expressed as a string of ASCII digits.

```
10 LET T$ = VAL$(128+34) ! T$ = "162"
```

RPT$(S$,X) ST Repeats the string in S$ X times.

```
10 LET T$ = "Ring!"
20 LET E$ = RPT$(T$,3)
E$ = "Ring!Ring!Ring"
```

TRIM$(S$) ST Returns a string which is equal to S$ with all leading and trailing blanks stripped off.

```
10 LET T$ = "  Pay roll   "
20 LET F$ = TRIM$(T$)
F$ = "Pay roll"
```

UPC$(S$) ST Returns a string made up of the uppercase equivalents of each of the characters in S$.

PROC$(X) ST Returns the string representation of the key with a code value of X. If the key is a function key, the appropriate escape sequence is returned. PROC$ cannot process all of the terminal keycodes. Refer to the BASIC reference manual for additional information.

Print Functions

The following functions can only be used in a PRINT statement. If they are used in a PRINT USING statement they will cause your program to end with an error. The MOVC functions will cause an error if used in PRINT # statements. They do not return a function value.

LIN(N) Skips N lines in the alphanumeric display (current window).

SPA(N) Prints N blanks in the alphanumeric display (currrent window).

TAB(N) Prints blanks up to column N in the alphanumeric display (current window).

Example: Print "TEST" skip 3 lines, space 5 columns to the right, print "BEGIN", then tab to column 20 and print "# 1".

```
10 PRINT"TEST",LIN(3),SPA(5),"BEGIN",
TAB(20),"#1"
```

MOVCS(R,C) Moves the cursor to row R, column C. R and C are screen relative coordinates (1,1 to 24,80).

MOVCA(R,C) Moves the cursor to row R, column C. R and C are absolute display memory addresses (1,1 to 255,80).

MOVCR(R,C) Moves the cursor to the current row plus R and the current column plus C. R and C can be positive or negative.

Example: Position the cursor at absolute row 10 and column 20. Print the letter "A" and then move the cursor down 5 rows and to the right 10 columns.

```
10 PRINT MOVCA(10,20),"A",MOVCR(5,10);
20 INPUT B
```

Note that a semicolon (";") must be used at the end of the print list to prevent an automatic carriage return.

Input/Output Functions

GETKBD(X) I Returns in X the keycode associated with the next key struck. The function value is 1 if a keycode is returned and 0 if no keycode was found. This function can only be executed if the GETKBD flag is set (see GETKBD ON).

DSPIN$(L,X) ST Returns a string from the display, starting at the current cursor position. The absolute value of L is the length of the returned string. If format mode is on, the text returned will be limited to the current unprotected field. If L is negative, character values in the range 128—255 are ignored. If L is positive, characters in the 128—255 range are expanded to their representative escape sequences. In the terminal, characters with values greater than 127 are used to indicate display enhancements, field types, etc. Additional information on these character values is given in the BASIC for Terminals reference manual.

X must be a numeric variable and will be set to 0 if the returned string is not the last data in the display workspace. X will be set to a non—zero value if the end of the workspace is encountered.

GETDCM(S$) I Returns one character from the datacomm without wait. The character is stored in S$. The value of the function is 1 if a character was returned and 0 if there was no character. This function can only be executed if the GETDCM flag is set (see GETDCM ON).

PUTDCM(S$) I Sends the first character in S$ to the datacomm. The function value is 1 if the transfer was successful, 0 if the transfer was unsuccessful, and −1 if the datacomm was busy.

Other

ERRL I Returns the line number in which the last error occurred. See ON ERROR.

ERRN I Returns the error code of the last error. See ON ERROR.

FRE(expression) I Returns the amount of available memory space. If the expression is numeric, the returned value is the amount of remaining program and variable storage. If the expression is a string, the returned value is the amount of remaining string storage space.

A GRAPHICS LANGUAGE (AGL)

INTRODUCTION

What is "A Graphics Language" (AGL)?

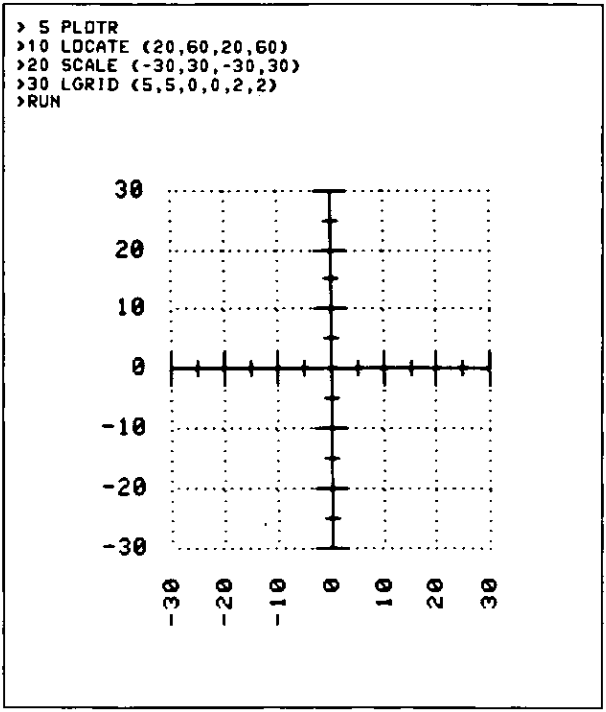
AGL is an extension to BASIC that provides easy to use graphics commands. If you have an HP 2647A Graphics Terminal, you can perform these additional graphics operations. These graphics operations are a subset of the AGL functions available on other Hewlett—Packard graphics systems.

AGL consists of powerful graphics functions that allow you to perform graphics operations with a minimum of programming.

For example, to draw a labeled grid it would take the following individual escape sequences:

```
10 REM DRAW AND LABEL GRID
 20 PRINT "ENTER XMIN, XMAX, AND XINTERVAL"
 30 INPUT X1,X2,X3
 40 PRINT "ENTER YMIN; YMAX, AND YINTERVAL"
 50 INPUT Y1,Y2,Y3
 60 REM DRAW X/Y LABELS
 70 PRINT CHR$(27)&"*m";X1;Y1;"J"
 80 FOR I=X1 TO X2 STEP X3
 90 I$=VAL$(I)
100 PRINT CHR$(27)&"*l"&I$
110 NEXT I
120 REM SAME FOR Y LABELS
  .
  .
  .
130 REM DRAW GRID
140 REM SELECT DOTTED LINE
150 PRINT CHR$(27)&"*m7B"
160 FOR I=X1 TO X2 STEP X3
170 PRINT CHR$(27)&"*pa";I;Y1;"b";I+5;Y2;"A"
180 NEXT I
190 REM SAME FOR Y GRID
  .
  .
  .
200 REM RESTORE LINE TYPE
210 PRINT CHR$(27)&"*m1B"
220 REM DRAW AXIS
  .
  .
  .
240 REM DRAW MAJOR AND MINOR TICS
  .
  .
  .
250 END
```

In AGL the same operation could be performed as follows:



```
> 5 PLOTR
>10 LOCATE (20,60,20,60)
>20 SCALE (-30,30,-30,30)
>30 LGRID (5,5,0,0,2,2)
>RUN
```

AGL requires fewer statements, no control characters, and English—like BASIC commands. In other words, its easier to use for high level graphics operations.

The rest of this section describes the AGL functions available in the HP 2647A Graphics Terminal and gives examples of their use.

AGL TERMINOLOGY

Regions

There are several types of graphics regions used by AGL (see figure 5−1). These regions include the following:

- Logical Address Space (A1,A2)
- Mechanical Space (Limits) (M1,M2)
- Graph Limits (P1,P2)
- Graphic Display Space (GDU) (G1,G2)
- Region of Interest (Viewport) (V1,V2)

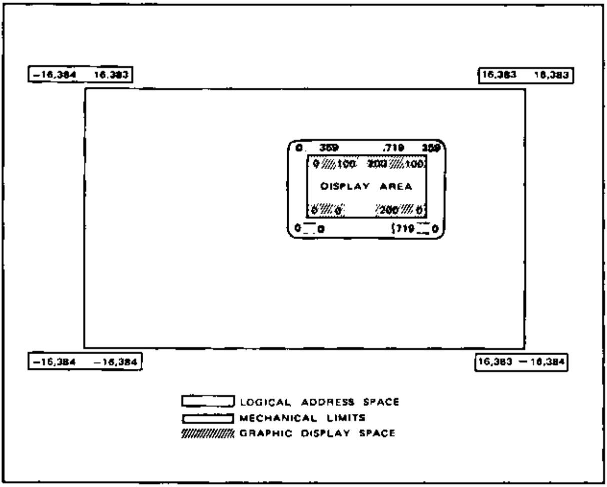


Figure 5−1. AGL Regions

The following paragraphs describe how these regions apply to the HP 2647A Graphics Terminal.

AGL can be used with several plotter devices. Refer to the appropriate plotter documentation for AGL operation.

Logical Address Space (A1,A2)

The Logical Address Space is the range of data values which may be referenced by the terminal or plotter. This is the "logical" plotting area and it may be larger than the mechanical limits described below (see figure 5−1). The logical limits for X and Y values on the HP 2647A are $-16{,}384 < X,Y < +16{,}383$.

Mechanical Limits (M1,M2)

The mechanical limits define the physical display area (device limits) of the terminal or plotter. The mechanical limits of the terminal correspond to points in the range 0,0 to 719,359 (see figure 5−1).

Graph Limits (P1,P2)

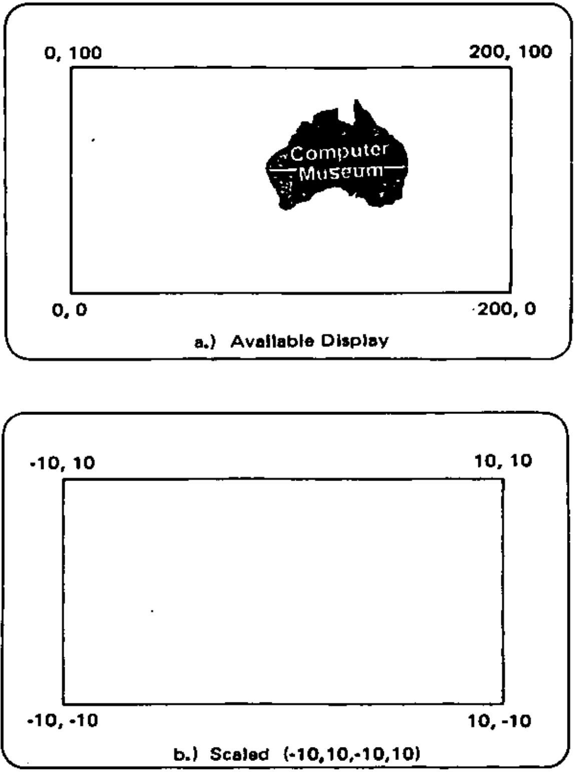
The graph limits define the desired display area. This area is within the terminal's mechanical limits. The graph limits (P1,P2) are set to default values by the GPON command or to user values by the LIMIT command. Note that these values can also be set with the front panel controls on a plotter (refer to the appropriate plotter manual). Whenever these points are redefined or read by AGL, the following regions are set to the graph limits (P1,P2):

- Graphic Display Space (GDU) (G1,G2)
- Region of Interest (V1,V2)
- Hard Clip Limits (H1,H2)
- Soft Clip Limits (S1,S2)

These points may be set at the beginning of a program to specify an area on the display where all plotting will be done.

Graphic Display Space (GDU) (G1,G2)

The available display in the terminal consists of an area 200 x 100 graphic display units (GDU's) in size. These units are used to refer to a logical display space (see figure 5—2).



Computer Museum

a.) Available Display

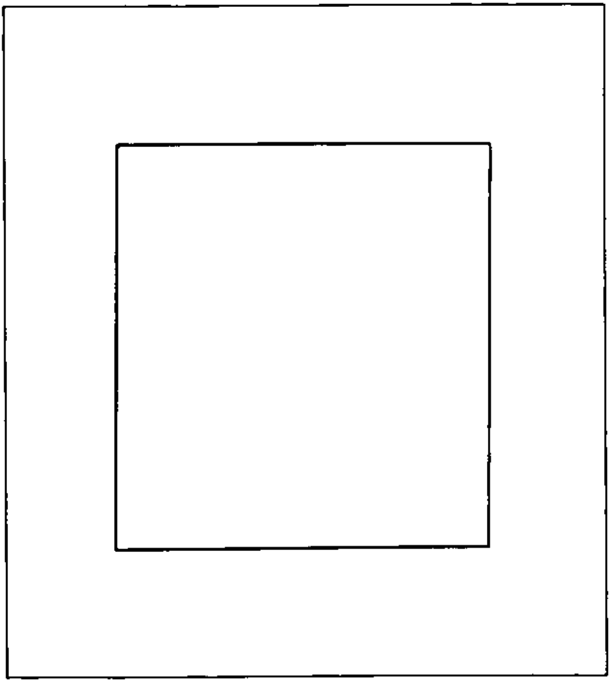
b.) Scaled (-10,10,-10,10)

Figure 5—2. Graphic Display Space

GDU's are used in the LOCATE command to describe the display, independent of the data values displayed.

Example:

```
> 5 PLOTR
>10 LOCATE (10,70,20,70)
>20 SCALE (-10,10,-10,10)
>30 FRAME
>RUN
```



The values of H1 and H2 define the limits of GDU space. H1 and H2 specify the hard clip region to which all plotting is confined. This space is defined such that the point 0,0 in GDU's is mapped to H1. This point is referred to as G1. Thus, each time H1 and H2 are changed, GDU space is redefined and the new G1,G2 points (limits) are set equal to H1 and H2.

AGL maintains the points G1,G2 and the scaling necessary to map GDU's to the display automatically. When using AGL, you will always plot using GDU's or UDU's (user defined units).

Region of Interest (Viewport) (V1,V2)

This is a rectangular area into which your data is plotted. You can define this region of interest with the LOCATE or MARGIN commands. Whenever points V1,V2 are changed, the soft clipping limits are also set (S1=V1 and S2=V2). See "Soft Clip Limits (S1,S2)."

Once this region has been defined, you can map data to this rectangle with the SCALE or SHOW commands. This region is useful when you wish to plot data in your own units and confine the plot to a specific region in the display. In this case, the LOCATE command is used in conjunction with SCALE or SHOW to allow plotting in user units. AGL automatically maps user units to plotter units so you need only concern yourself with units meaningful to you. Note whenever this region of interest is changed, AGL updates the soft clip limits to the new values of V1 and V2.

Clipping

Clipping eliminates that portion of the plotted data that lies outside of the specified graphic display area. There are two types of clipping limits: (1) Hard Clip Limits (H1,H2) and (2) Soft Clip Limits (S1,S2). Once these limits are set, only data that falls inside these limits is plotted.

Hard Clip Limits (H1,H2)

The Hard Clip Limits define the rectangular area within the graph limits (P1,P2) in which plotting can be done. No marks can be made outside this region. AGL automatically stops plotting at the current hard clip limits.

The hard clip limits can be changed with LIMIT and SETAR commands. The LIMIT command sets the graph limits (P1,P2) and then sets the hard clip limits (H1,H2) equal to P1,P2. The SETAR command reads the graph limits, calculates the proper aspect ratio, and then sets the resulting hard clip limits.

Hard clip limits are intended to clip both vectors and labels. In plotter devices, AGL maps points G1,G2 in GDU's to points H1,H2 which are in plotter units. This results in scaling factors used to map GDU's to plotter units. Not all plotter devices can change their hard clip limits. This results in some labels being made outside the designated display area.

Soft Clip Limits (S1,S2)

Soft clipping allows you to redefine the clipping area within the hard clip region. Only vectors (not labels) are affected by the soft clip limits. This allows plotted data to be clipped within the soft clip boundaries, while allowing labels and titles to be outside these boundaries. The soft clip limits can be changed and redefined at many points throughout an application program. S1 and S2 must lie within (or coincide with) the hard clip limits H1,H2. If you try to place S1 or S2 outside the hard clip region, the intersection of these regions is stored by AGL as the soft clip region. If the hard and soft clip regions do not intersect or these regions intersect as a line, an error message will be printed and no plotting will appear on the graphic device.

The Effect of AGL Commands On Graphic Regions

Table 5−1 contains a list of AGL commands and their effect on the various regions described previously. Each command affects the regions in a different way.

Units

AGL has the capability to use three unit systems:

- User Defined Units (UDU's)
- Graphic Display Units (GDU's)
- Metric Units

User Defined Units (UDU's)

This system defines the units used in your application. These units are automatically scaled and translated to machine units by AGL. You can switch between unit systems at different points in your application program. This allows you to plot in units that you understand.

Table 5–1. The Effect of AGL Commands On Graphic Regions

COMMANDS REGIONS	GPON(1)	LIMIT	GPON(2)	SETAR	GPON(3)	LOCATE MARGIN	CLIP
MECHANI-CAL (M1,M2)							
GRAPH (P1,P2)	SETS DEFAULT	SETS USER SPEC	READS FROM DEVICE				
HARDCLIP (H1,H2) (NOTE 1)	SETS H1=P1 H2=P2			COMPUTE H1,H2 USING P1,P2			
GRAPHIC DISPLAY (G1,G2) (NOTE 2)	SETS G1=H1 G2=H2						
REGION of INTEREST (V1,V2)	SETS V1=G1 V2=G2					SETS V1,V2 TO USER SPEC	
SOFT CLIP (S1,S2) (NOTE 3)	SETS S1=V1 S2=V2 SOFT CLIP ON					SETS S1=V1 S2=V2	S1,S2 TO USER SPEC SOFT CLIP ON

NOTES:

1. The device driver tries to set the device hard clip limits to H1,H2.
2. GDU space is mapped to G1, G2 by computing appropriate scale factors to be used by AGL in GDU plotting. These scale factors transform GDU's to device dependent machine units.
3. Soft clipping occurs only when soft clipping has been turned on.

Graphic Display Units (GDU's)

Graphic Display Units (GDU's) are defined as being one percent of the length of the shortest side of the space bounded by the hard clip limits (H1,H2). Thus the short side is 100 GDU's in length, and the long side is

```
= (100 GDU's)*(Long side /short side).
```

Once the hard clip limits have been established with the LIMIT or SETAR commands, the GDU system is automatically scaled to the hard clip boundaries. Figure 5−3 shows the drawing area defined in the GDU system.

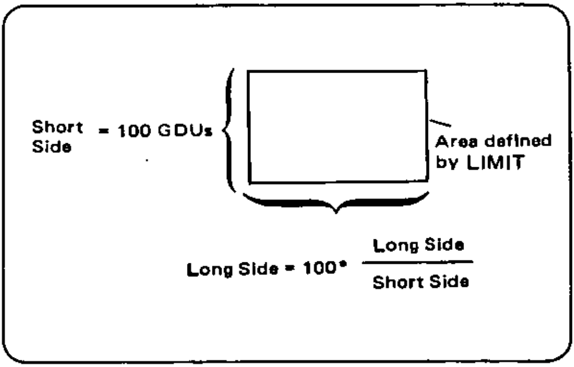


Figure 5−3. Graphic Display Units (GDU's)

Metric Units

The basic unit of distance in the METRIC mode is the millimeter. This mode defines user units so that functions plotted are scaled to millimeters. This mode is useful in drafting applications to plot physically scaled drawings. Metric units are turned on by the MSCALE command. When plotting to the terminal display, drawings may be distorted slightly due to round off errors.

Other Terms

Additional terms are defined where used in the command descriptions that follow.

FUNCTION GROUPS

The AGL functions can be separated into four groups:

- Set—Up Functions
- Plotting Functions
- Axis and Labeling Functions
- Interactive Functions

Table 5—2 contains a list of the AGL functions. The remaining paragraphs in this section provide detailed descriptions of the functions.

Note: The AGL functions are described for the HP 2647A Graphics Terminal. Functions whose operations vary on other graphic devices are also noted.

FUNCTION SYNTAX

The general form for all functions is:

```
keyword(P1,P2,P3,...Pn)

Where keyword = Function name

Pn = Numeric expressions or variables
```

Parameters are read left to right. Missing parameters are assigned default values where possible. Refer to the function descriptions for the default values for the various functions. Arrays and strings cannot be used as parameters.

Examples:

```
>10 X = 3
>20 Y = -10
>30 Npen = 2
>40 PLOT (X,Y,Npen)
```

This would read the X coordinate as 3, the Y coordinate as −10, and the pen parameter as 2 (lift after the plot).

```
>10 PLOT (5,10)
```

This would read X coordinate as 5, Y coordinate as 10, and assign the default value for the pen position. The pen default for the PLOT function is 1 (move and then put pen down). Therefore PLOT(5,10)=PLOT(5,10,1).

Table 5–2. AGL FUNCTIONS

FUNCTION	DESCRIPTION	
SETUP		
PLOTR	Select and Initialize	
GPON	Power On Reset Plotter	
SETAR	Set Aspect Ratio	
LIMIT	Set Hard Clip Limit	
GCLR	Clear Display	
LOCATE	Define Plotting Area	
MARGIN	Define Plotting Area	
SCALE	Define User Units	
SHOW	Define User Units	
MSCALE	Set Up Metric Scaling	
CLIP	Move Soft Clip Limit	
CLIPOFF	Suspend Soft Clip	
CLIPON	Restore Soft Clip	
SETGU/SETUU	Select GDU's/User Units	
AXIS/LABELING		
XAXIS	Draw Linear X Axis	
YAXIS	Draw Linear Y Axis	
LXAXIS	Draw Labeled X Axis	
LYAXIS	Draw Labeled Y Axis	
AXES	Draw Linear Axes	
LAXES	Draw and Label Axes	
GRID	Draw Linear Grid	
LGRID	Draw and Label Grid	
FRAME	Outline Soft Clip Area	
FXD	Set Label Format	
LORG	Set Label Origin Mode	
LDIR	Set Label Direction	
CSIZE	Set Character Size	
PLOTTING		
PENUP/PENDN	Lift/Drop "Pen"	
PEN	Select a "Pen"	
LINE	Select Dash Pattern	
PLOT	Absolute Plotting	
MOVE	Absolute Move	
DRAW	Absolute Draw	
RPLOT	Relative Plotting	
IPLOT	Incremental Plot	
PDIR	Plot Direction (for RPLOT and IPLOT)	
PORG	Set Relocatable Origin	
INTERACTIVE		
WHERE	Read Pen Position	
POINT	Set Cursor Position	
CURSOR	Read Cursor Position	
DIGITIZE	Read Cursor with Wait	
GPMM	mm to GDU Conversion	
DSIZE	Return Size to Data	
DSTAT	Display Status	
GSTAT	Graphics Package Status	

SET-UP FUNCTIONS

PLOTR

PLOTR [(LU# [, action [, HPIB [, LOGLU]]])]

Where: LU# = the logical unit number of the plotting device. Default is 0 which is the terminal.

action = one of five possible device operations described below. Default is 1 which selects and initializes the new plotting device.

HPIB = the address of the output (plotting) device. Default is equalled to the assigned LU#. If LU# is 0, then HPIB is ignored.

LOGLU = the logical unit number of the Log device. Default is 0 which ignores the Log device assignment. IF the LU# is 0 then the LOGLU# is ignored.

The PLOTR statement must be used to initialize AGL. PLOTR selects and initializes the plotting or display device to be used. The default units are GDU's. This can be changed using the SETUU, SCALE, SHOW, or MSCALE statements. You can select one of five device operations by specifying one of the following actions:

0: Terminates use of device.
1: Default. Selects new device and initializes (level 2A of GPON). A clear operation (GCLR) is not performed. No buffering.
2: Resumes use of device (no initialize).
3: Suspends device.
4: Same as action 1 above. On other plotting devices this may select a device with data buffering.

The HPIB parameter is the HP—IB address of the output device. This parameter is ignored if the LU# is 0. If an LU# has not previously been assigned, the HPIB parameter will be assigned to the logical unit number.

The LOGLU parameter selects the Log (write only) device. This parameter is only used when the LU# is not 0.

Example:

```
>10 PLOTR (1,0)
>20 PLOTR (A,L)
>30 PLOTR (A+B,C-1)
```

GPON

GPON [(<level>)]

The GPON function is defined as the "Power-On" reset level of the plotting device, where level is one of the following:

Level 1: Sets Graph Limits P1,P2 to the (device-dependent) default values and performs all Level 2 and 3 operations.

Level 2: Clears display (paper-moving devices do not advance paper). If the level is not specified, GPON(2) is used.

Level 2A: Reads the hardware Graph Limits (P1,P2) in MU's. This level is used by the PLOTR and SETAR statements when no parameters are given.

Level 2B: Sets the hard clip limits H1,H2 equal to P1,P2. This level is used by the LIMIT statement.

Level 2C: Sets the hard clip limits H1,H2 independent of P1,P2. Calculates GDU to machine unit scale factors by mapping G1,G2 to H1,H2. Also, sets the hard clip limits of the device to H1,H2. P1 and P2 are not affected. This level is used by SETAR when parameters are given.
Selects Pen 1
Puts the pen up at the HOME (0,0 GDU) position.
Updating of the relocatable origin is enabled (see PORG).
Sets relocatable origin to (0,0) GDU's.
Performs all Level 3 operations.

Level 3: Gets address space information needed by plotting package.
Sets S1= V1= (0,0) GDU's, and S2= V2= "G2" in GDU's.
Sets user units = GDU's (i.e., U1= V1; U2= V2).
Puts the "pen" up, without moving it.
Selects solid lines. This does not affect the device line flag.
Selects standard character set.
Selects LORG (1) (left-justified labeling).
Sets labeling direction to left-to-right (LDIR (0)).
Clears any error conditions.
CSIZE is set to the device default value.
FXD(0,0) is executed.
Current units are set to GDU's.
Sets PDIR argument to zero degrees.
On devices with only one pen, the "linetype called" flag is reset.
The "soft clipping" flag is set (turns clipping on).

Note: The levels 2A, 2B, and 2C above are not attainable by a parameter in the GPON statement. They are used by the PLOTR, SETAR, and LIMIT commands.

Examples:

```
10 GPON
20 GPON (3)
30 GPON (N)
```

SETAR

SETAR [(<aspect ratio>)]

The set aspect ratio (SETAR) function maintains the height and width ratio of the plotted data from one device to another. The hard clip limits H1,H2 and GDU limits are reset so that GDU space will have the desired aspect ratio. This redefined GDU space is made as large as possible within the original Graph Limits P1,P2.

The Graph Limits are read, the hard clip limits H1,H2 are calculated, and the GDU to MU scale factors are defined.

When a ratio is given, the hard clip limits H1,H2 are calculated and a call to GPON (2C) is made. If no ratio is given a call to GPON (2A) is made (see GPON).

Examples:

```
10 SETAR
20 SETAR (2)
30 SETAR (Y/X)
```

LIMIT

LIMIT [(<x1>,<x2>, <y1>,<y2>)]

Where X1,Y1 defines coordinate G1 and X2,Y2 defines coordinate G2. Both coordinates are in millimeter units and their origin begins at the lower-left corner (mechanical limits).

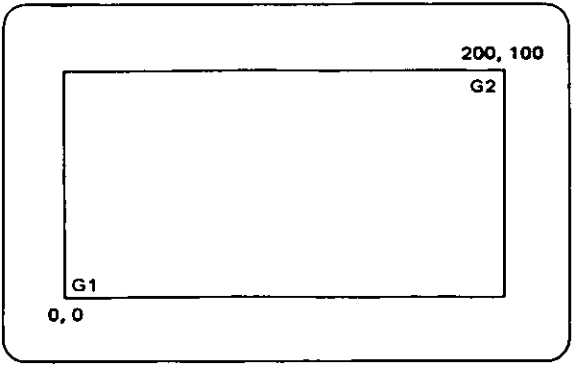
The LIMIT function specifies the display space available for plotting data on a terminal or plotting device.

If no coordinates are supplied, AGL will execute the DIGITIZE twice to allow you to input the values with the graphics cursor. A message will appear in the message window giving the cursor location in MM's (even though UDU's may be indicated in the message). (See the DIGITIZE statement.) The points may be entered in any order. The lower left point will define G1, and the upper right G2.

The LIMIT statement sets P1,P2 to the user specification. It then calls GPON (2B) to reset the Hard Clip limits, redefines GDU space, and restores default conditions. The units for LIMIT are millimeters.

Example:

```
> 5 PLOTR
>10 LIMIT (0,200,0,100)
>20 FRAME
>RUN
```



GCLR

GCLR [(paperfeed)]

The GCLR function clears the plotted display on the terminal or plotting device and positions the pen at 0,0 (in GDU's).

The optional paperfeed parameter is interpreted as follows:

>0 - form feed (default = 1).
=0 - no action.
<0 - previous page.

GPON should normally be used between plots, possibly with GCLR, since it restores the default conditions.

Note: The terminal and most plotting devices cannot advance paper. If the optional distance parameter is used with these devices, it will be ignored.

Examples:

```
10 GCLR
20 GCLR(-1)
30 GCLR(N)
```

LOCATE

LOCATE [(<x1>,<x2>, <y1>,<y2>)]

Where X1,Y1 defines coordinate V1 and X2,Y2 defines coordinate V2. All parameters are type REAL, in GDU's.

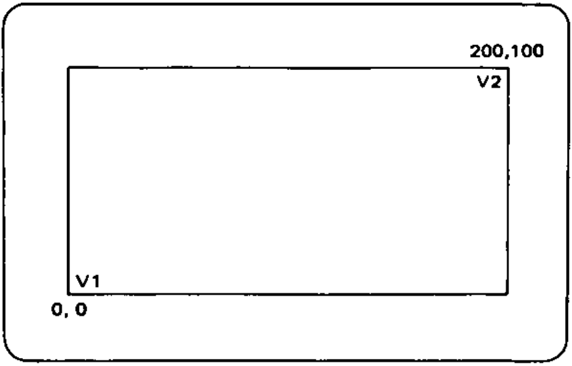
The LOCATE function defines the rectangle on the plotting device onto which SCALE will map, or SHOW will fill, and also (re)sets the default clipping boundary.

LOCATE sets the values of V1 and V2, and also the default "soft" clipping limits S1 and S2. V1 is the lower left corner, and V2 is the upper right corner of the rectangle. LOCATE thus specifies a rectangle which will contain the data transformed from User Units. These limits can be overridden by a call to CLIP (or CLIPOFF). CLIP may be used to separate the clipping rectangle from the mapping rectangle. LOCATE turns clipping on.

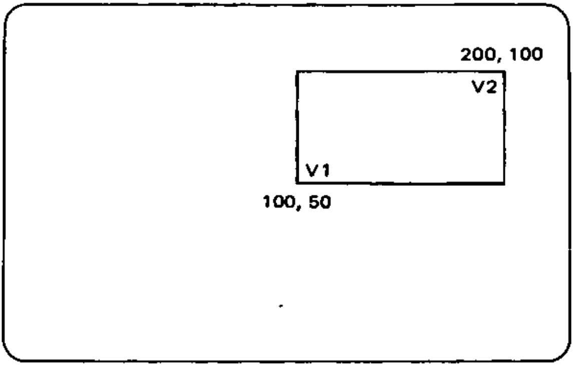
If LOCATE is to be used, it must be called before SCALE or SHOW. If no coordinates are given, the DIGITIZE function will be used to allow you to input values using the graphics cursor.

Examples:

```
> 5 PLOTR
>10 LOCATE (0,200,0,100)
>20 FRAME
>RUN
```



```
> 5 PLOTR
>10 LOCATE (100,200,50,100)
>20 FRAME
>RUN
```



MARGIN

MARGIN (left, right, bottom, top [, units])

MARGIN specifies the LOCATE rectangle relative to the hard--clip boundary (in characters or GDU's). This allows you to set the number of characters that can be between the hard and soft clip limits. The character size in effect at the time of the MARGIN call is used. All parameters are type REAL.

The <units> parameter causes the spacing parameters to be interpreted as either character cell or GDU distances inward from the hard clip boundries. If the <units> parameter is =0 (the default), the spacing parameters are read as character spacings. If the <units> parameter is <>0 the spacing parameters are read as GDU's.

When character cell spacing has been selected (units=0), the spacing parameters, LEFT, RIGHT, TOP, and BOTTOM give the number of 7 wide by 10 high characters that will fit in the margin. If the LEFT and RIGHT values are positive, the character cell width is used. Negative values cause the character cell height to be used.

If the TOP or BOTTOM values are positive, the character cell height (line feed) is used. Negative values cause the character cell width to be used.

Example:

```
30 MARGIN (10,10,5,5)
```

SCALE

SCALE (<x1>,<x2>, <y1>,<y2>)

Where X1,Y1 defines coordinate U1 and X2,Y2 defines coordinate U2.

SCALE specifies a rectangle of user space which is to be mapped exactly onto the plotter space defined in tha LOCATE statement. Typically the units and scale factors will be different for X and Y. The values used for X1,Y1 and X2,Y2 are in user units.

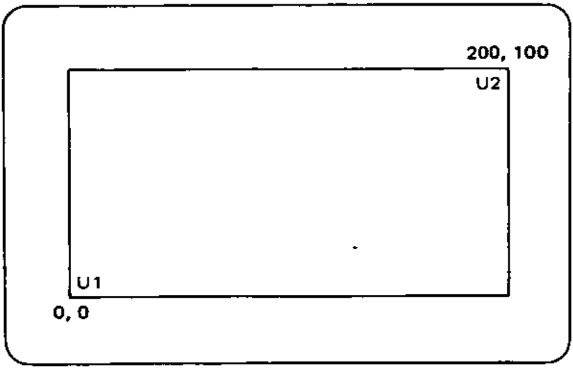
The scaling equation is computed using the current values of V1 and V2 such that U1 maps onto V1 and U2 maps onto V2; then U1 and U2 are discarded.

SCALE sets current units to User Units (see SETUU).

If LOCATE is to be used, it must be called prior to SCALE.

Example:

```
> 5 PLOTR
>10 SCALE (0,200,0,100)
>20 FRAME
>RUN
```



SHOW

SHOW (<x1>,<x2>, <y1>,<y2>)

Where X1,Y1 defines the coordinate U3 and X2,Y2 defines coordinate U4. The units are always in user units.

SHOW specifies a rectangle of user space which is to be mapped into the plotter-space rectangle defined by LOCATE, in such a way that all of the rectangle U3,U4 is shown centered within the rectangle V1,V2, as large as possible, and with no distortion.

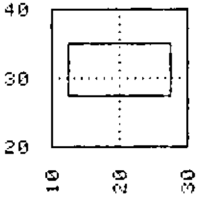
The scaling equation is computed using the current values of V1 and V2; then U3 and U4 are discarded. SHOW sets current units to User units (calls SETUU). The same user units normally apply in both X and Y (as is the case in drafting or other geometric applications). SHOW ensures that X and Y scaling factors are matched, thus eliminating distortions.

In practice, SHOW will cause a larger user-space rectangle U1,U2 to be mapped onto V1, V2. CLIP may be used to restrict the visible data to the same values used by SHOW, if desired.

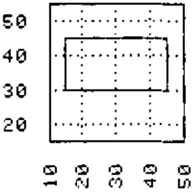
If LOCATE is to be used,it must be called prior to SHOW.

Example:

```
>10 PLOTR
>20 LOCATE(10,30,20,40)\FRAME
>30 SHOW(10,50,20,50)
>40 MOVE(15,30)\DRAW(45,30)\DRAW(45,45)
>50 DRAW(15,45)\DRAW(15,30)
```



GDU's



UDU's

MSCALE

MSCALE (< x reference>, <y reference>)

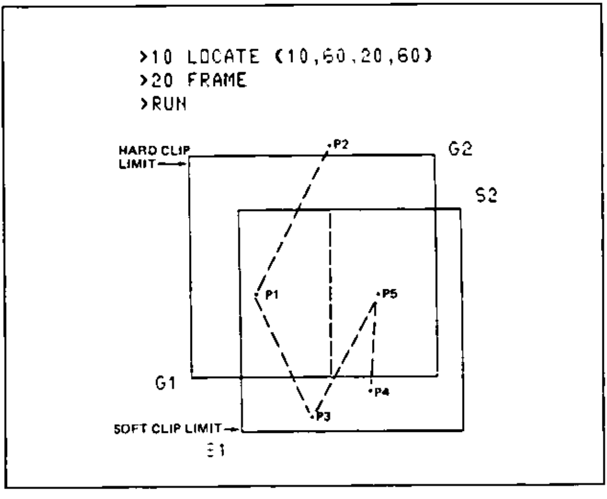
The MSCALE function causes AGL to accept X,Y values in millimeters. The origin (0,0) is offset from the hard clip corner G1 by the x and y reference values. The reference point need not be inside the G1,G2 rectangle. The X and Y parameter values must be in millimeters.

CLIP

CLIP (X1,X2,Y1,Y2)

CLIP redefines the soft clip limits. X1,Y1 defines point S1 and X2,Y2 defines point S2 in current units. The CLIP statement also turns on the soft clipping operation (see CLIPON).

Example: A sequence of pen down plots to points P1, P2,...P5 would leave the plot as shown below:



```
>10 LOCATE (10,60,20,60)
>20 FRAME
>RUN
```

If part of the soft–clip region falls outside the hard–clip limit, the soft–clip limits will be redefined to correspond to the intersection of the regions. If there is no intersection between the regions, an error will occur.

CLIPOFF/CLIPON

CLIPOFF
or
CLIPON

CLIPOFF turns off soft—clipping, but retains the limit values. This allows positioning the "pen" anywhere in the display space defined by G1 and G2 while in user—defined units.

CLIPON restores soft clipping.

Clipping is also restored by executingℓPLOTR, LIMIT, LOCATE, GPON(3), or CLIP. SCALE and SHOW do not affect clipping.

SETGU/SETUU

SETGU
or
SETUU

SETGU selects graphic display units (GDU's) and SETUU selects user—defined units (UDU's). Several graphic functions interpret their parameters based on the units currently selected. These two functions provide the mechanism for setting the current mode.

Graphic Display Units (GDU's) are primarily intended as a device—independent coordinate system used for positioning items on the display. Simple programs can leave the mode set to "user".

AXIS AND LABELING FUNCTIONS

The axes and labeling functions use the clipping boundries to define the end points of the axes. The soft clipping limit values are used for X1 and X2 (end points of x axis). This allows the Y limits to be placed to position the X axis conveniently. The X axis can even be placed outside of the current clipping area. This allows the placement of multiple X axes outside of the region of interest.

The Y tic marks are placed symmetrically about the x-axis and are not clipped. Grid lines extend from one plotting boundry to the other. (Y1, Y2 are soft clip values when in UDU's or hard clip limits when in GDU's.

XAXIS

XAXIS [(<x tic-spacing>[,<x origin>,<y origin>
[,<x major count>[,<tic size>]]])]

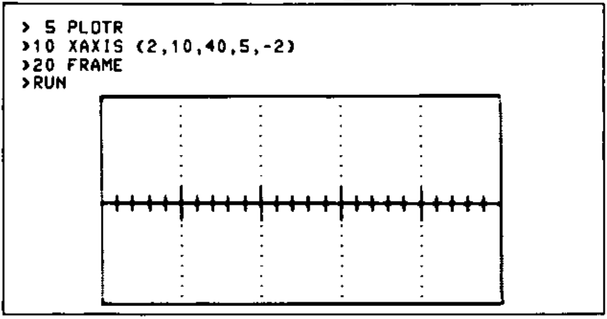
Where "x tic-spacing" is interpreted in the current units mode. The sign is ignored. Default of 0=> no tics.

The "origins" are in current units. This allows proper placement of axis and tics. The "y origin" specifies the placement of the X axis and the major Y tics while the "x origin" specifies the placement of major X tics and grid lines. The default origin is 0,0.

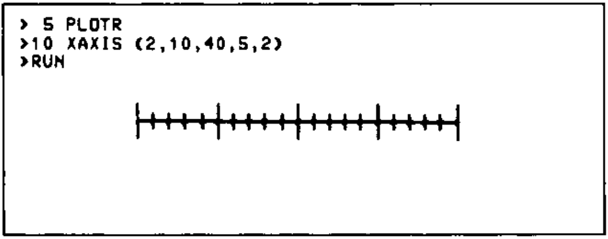
The "x major count" is a unitless integer value which specifies the intervals between "major" tic marks as numbers of "minor" tics. The signs are ignored. The default is 1 => all major tics. 0 => all minor tics.

The "tic size" specifies the length of the minor tics (end to end; tics are symmetric about the axis). Signs are ignored for minor tics. If the sign is positive the major tics are twice as long as minor tics. If the parameter is negative, the major tics become grid lines which extend from one LOCATE boundary to the other. The default tic size is +2 GDU's. The parameter is type REAL interpreted in GDU's. XAXIS generates an X axis at y = y origin. Tic marks are positioned along the axis such that a major tic mark falls on the origin (whether visible or not); the origin may lie outside the current soft clipping region. Tic marks may or may not coincide with the edge of the clipping boundary.

Examples:



```
> 5 PLOTR
>10 XAXIS (2,10,40,5,-2)
>20 FRAME
>RUN
```



```
> 5 PLOTR
>10 XAXIS (2,10,40,5,2)
>RUN
```

YAXIS

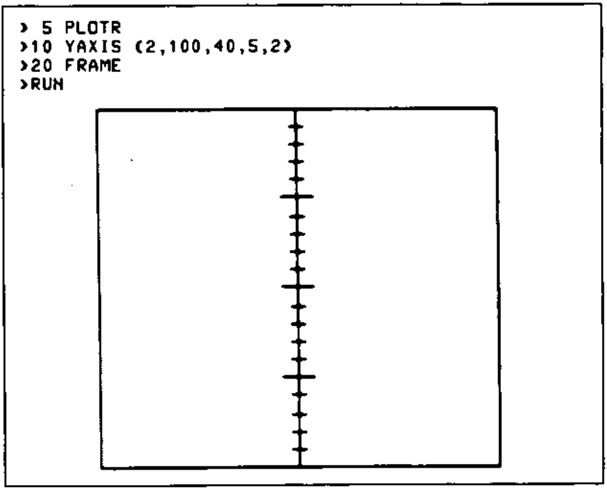
**YAXIS [(<y tic spacing>[,<x origin>,<y origin>
[,<y major count>[,<tic size>]]])]**

Where all YAXIS parameters are defined as in XAXIS, except in the Y (vertical) direction.

YAXIS generates a Y axis at x = x origin. All parameters are interpreted as in XAXIS except that now all concern the drawing of a Y axis.

It is useful to note that a call to XAXIS followed by a call to YAXIS will generate a pair of perpendicular axes which specify a Cartesian Coordinate System in the current units mode.

Example:



```
> 5 PLOTR
>10 YAXIS (2,100,40,5,2)
>20 FRAME
>RUN
```

LXAXIS

**LXAXIS [(<x tic spacing>[,<x origin>,<y origin>
[,<x major count>[,<tic size>[,<label loc>]]]])]**

Where the "tic spacing" is used to determine the orientation of labels: + for perpendicular to the corresponding axis; - for parallel. Otherwise, the tic spacing, origins, major count, and the tic size are all interpreted as in XAXIS. The "label loc" parameter is used to place the label relative to the axis. Use one of four label locations:

0: label below and outside the current vector clipping limits.

1: label immediately below the axis about two GDU'S from the corresponding tic.

2: label immediately above the axis about two GDU's from the corresponding tic.

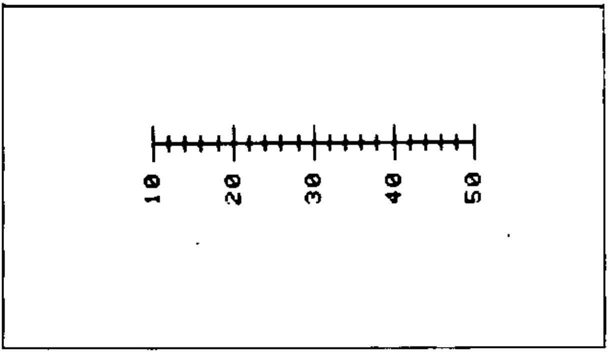
3: label above and outside the current vector clipping limits.

LXAXIS draws and labels the X axis in current units mode.

Example:

Computer Museum

```
>10 PLOTR
>20 SETUU
>30 LOCATE (10,50,0,100)
>40 LXAXIS (2,10,40,5,2,1)
>RUN
```



LYAXIS

LYAXIS [(<y tic spacing>[,<x origin>,<y origin>
[,<y major count>[,<tic size>[,<label loc>]]]])]

Where LYAXIS parameters are the same as LXAXIS, except in the Y (vertical) direction and the label locations. The label locations are as follows:

0: label on the left side, just outside of the current vector clipping limits.

1: label to the immediate left of the axis, about two GDU's from the corresponding tic.

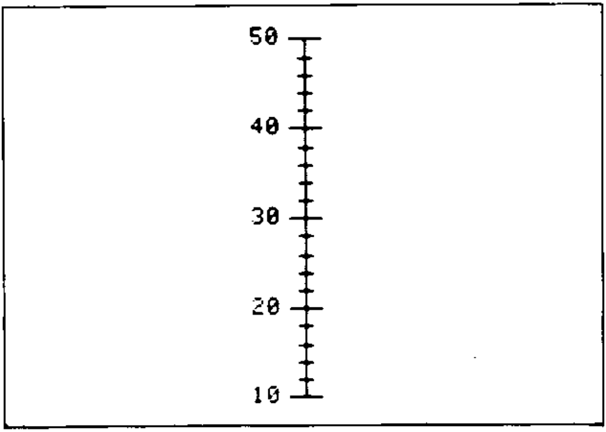
2: label to the immediate right of the axis, about two GDU's from the corresponding tic.

3: label on the right side, just outside of the current vector clipping limits.

LYAXIS draws and labels the Y-axis.

Example:

```
>10 PLOTR
>20 SETUU
>30 LOCATE (0,200,10,50)
>40 LYAXIS (2,100,40,5,2,1)
>RUN
```



AXES

```
AXES [ ( <x tic-spacing>, <y tic-spacing>
        [, <x origin>, <y origin>
         [, <x major count>, <y major count>
          [, <minor-tic size> ] ] ] ) ]
```

Where "tic-spacings" are REAL values in current units. The signs are ignored. Default or 0 => no tics.

The origins are also REAL values in current units. Default is (0,0).

The X and Y "major counts" are unitless integer values which specify the intervals between "major" tic-marks as numbers of "minor" tics. The signs are ignored. Default is 1 (i.e., all "major tics").

The "tic size" specifies the length of the minor tics (end-—to—end; tics are symmetric about the axes). Major tics are always twice as long as minor tics. The size is type REAL, in GDU's. Default size is approximately 2 GDU's.

AXES generates an X axis at y=y origin, and a Y—axis at x= x origin, assuming that these values are within (or on) the current user—units clipping limit. The axes will extend across that same clipping region. Tic—marks will be clipped at the clipping limit.

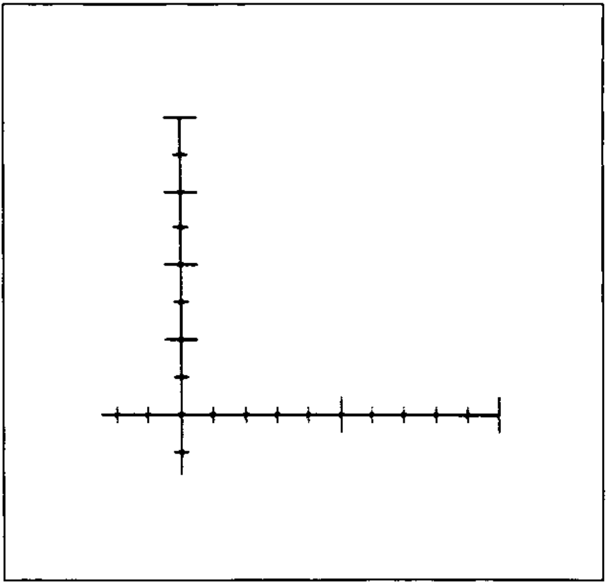
Tic marks are positioned along each axis such that a major tic—mark falls on the origin (whether visible or not). Tic marks may or may not coincide with the edge of the clipping boundary.

Axes and tic marks will be drawn using solid lines and the current "pen" (however, note that selecting solid lines on some displays is equivalent to selecting pen 1).

Example:

The following two calls would generate the axes shown within the plotting region specified by LOCATE (or the default plotting region). Labels in parentheses are for reference only.

```
>PLOTR
>SCALE (-5., 20., .8, 1.5)
>AXES (2.0, .1, 0.0, 1.0, 5, 2)
        \  /     \  /     \ /
        tics    origin  major interval
```



LAXES

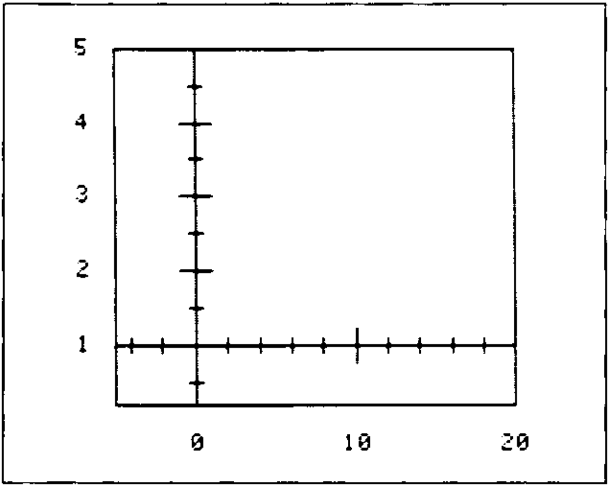
```
LAXES  [ ( <x tic-spacing>, <y tic-spacing>
          [, <x origin>, <y origin>
            [, <x major count>, <y major count>
              [, <minor tic size> ] ] ] )]
```

The LAXES parameters are the same as the AXES functions, except the signs of the tic-spacings determine the orientation of the labels: + for perpendicular to the corresponding axis, and - for parallel.

Each major tic is labeled. All tics are considered major tics if no "major count" is specified. Labels perpendicular to the X-axis will be lettered bottom-to-top (right-justified along the bottom border). Labels are placed outside the clipping limit, and are limited to a maximum of seven visible characters.

Example: The following three calls would generate the axes shown within the plotting region specified by LOCATE (Not the default plotting region). Labels are outside this plotting region.

```
>10 PLOTR
>20 LOCATE (20,80,20,80)
>30 SCALE (-5, 20, .2, 5)
>40 FRAME
>50 LAXES (2, .5, 0, 1, 5, 2)
>RUN
```



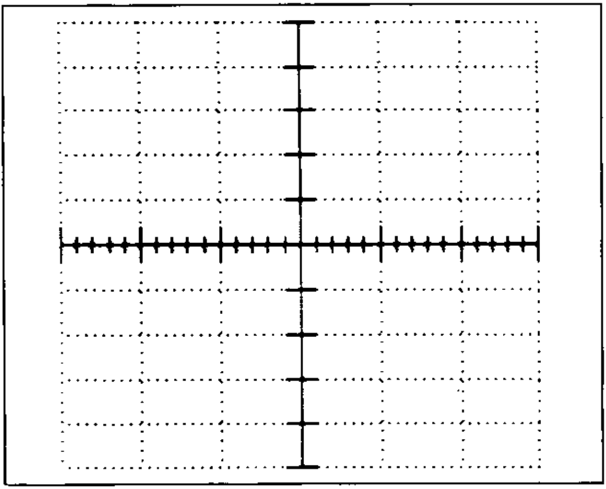
GRID

GRID [(<x tic-spacing>, <y tic-spacing>
[, <x origin>, <y origin>
[, <x major count>, <y major count>
[, <minor tic size>]]])]

GRID may be used as an alternative to AXES when a full grid is desired. Heavy lines (LINE(0)) are used for the axes. Lighter lines are used for minor divisions. All the lines extend throughout the current soft clipping region. GRID function parameters are the same as AXES, except that light cross-lines replace the tic-length indications.

Example:

```
>10 PLOTR
>20 LOCATE (0,60,10,60)
>30 GRID (2,10,100,50,5,1,2)
>RUN
```



LGRID

LGRID [(<x tic-spacing>, <y tic-spacing>
[, <x origin>, <y origin>
[, <x major count>, <y major count>
[,minor tic size]]]))]

LGRID draws a labeled grid and its characteristics are the same as for GRID, except that as in LAXES the signs of X and Y tic-spacings are used to specify the label orientations. LGRID uses units in the number range set up by the SCALE statement.

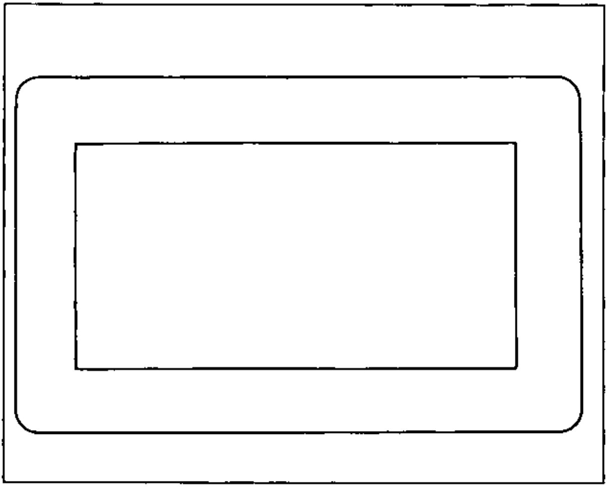
FRAME

FRAME

The FRAME function draws a "box" around the current vector-clipping region using the current pen and linetype characteristics.

Example:

```
>PLOTR
>FRAME
```



FXD

FXD [(X digits [,[Y digits]])]

Where "digits" is a positive integer. FXD selects the Axis-labeling format. Using FXD(n) causes the labels generated by LAXES and LGRID to be printed in F7.n format. Leading zeros are suppressed. If the number will not fit in the space to the left of the decimal point, the decimal will be moved to the right. When that is insufficient or the number underflows, F7.0 format will be used.

Note that the F7.0 format does not allow for more than one significant digit if a number overflows the specification. If you used 1.E+07 for example, only one digit after the decimal point would be displayed.

The default value for X digits is 0, the default value for Y digits is the X digits value. The maximum number of digits that can be used is 5.

LORG

LORG (mode)

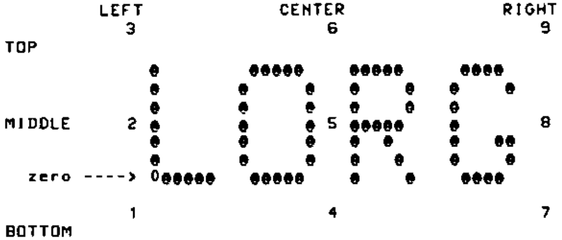
Where mode is a number in the range 0—9. If the number is not selected, LORG defaults to 1.

LORG selects the label—origin position and determines how subsequent labels will be placed relative to where the pen is when each label is received. Labels can be automatically right or left justified, or centered about a specified point. LORG 0—9 indicates the origin (justification and base line) for characters with respect to the current pen position. Labels are output using the PRINT #0 statement.

A carriage return (CR) causes the pen to be returned to its original position, and a linefeed (LF) causes the pen position to move in the expected direction.

If a label is to be left justified, the current pen position is the left margin. Center causes the label to be centered on the pen position. Right justify selects the pen position as the right margin. Bottom, middle, and top select the base line for the labeling string.

In the following illustration, each number shows the initial position of the pen relative to the label when LORG selects a number from 0 through 9. For example, if the label was to be right justified and set with a base line on top of the normal character position, the number "9" would be used.



Label positioning is done by the plotting device. When centering or right justification is used, the labeling string is buffered (stored) until all of the characters in the string have been received. The string end is detected and displayed by a CR or LF. The maximum length of a string when centering or right justifying is 80 characters (blanks are included but CR or LF is not). In all cases, data written beyond the edge of the screen is lost. There is no automatic RETURN when the screen boundary is reached.

The positioning of labels in the various modes should be such that the following two sequences would receive the same result:

```
LORG (5)
MOVE (A,B)
PRINT #0; "ABCD"
```

```
LORG (8)
MOVE (A,B)
PRINT #0; "AB"
MOVE (A,B)
LORG (2)
PRINT #0; "CD"
```

LDIR

LDIR (<angle in radians >)
or
LDIR (<run>, <rise>)

LDIR sets the lettering direction of labels. In general each device will use the available angle which is closest to the requested angle. Angles are measured in radians (1 radian = 57 degrees).

Since many devices cannot letter at angles other than multiples of 90 degrees (pi/2 radians), programs which use other angles may not run acceptably on all devices.

For angles in the range −90 degrees (−pi/2 radians) to +90 degrees (pi/2 radians), (<rise>/<run>) = TAN (<angle>).

Parameter interpretation depends on number of parameters. If only one parameter is supplied, it is assumed to be the counter-clockwise angle from the normal lettering direction (0 => towards 3 o'clock, 90 degrees => towards 12 o'clock, etc).

If two parameters are supplied, the angle used is that of an incremental vector plotted with arguments X increment = run, and Y increment = rise with X and Y in current units.

Example:

```
40 LDIR (5,8)
50 LDIR (90)
```

CSIZE

CSIZE (<height> [,<ratio> [,<slant in radians>]])

The character size (CSIZE) function specifies the size and aspect ratio of the alphanumerics or symbols to be drawn for labels. In general, if the requested size is not possible, the next smaller size (if any) should be used.

"Height" is the character cell height in GDU's. Default height is device-dependent.

"Ratio" is the aspect ratio of the character cell, defined as width/height. Default varies with device.

"Slant" specifies the clockwise rotation (in radians) of the character relative to its normal position. Horizontal lines are not affected. The character cell becomes a parallelogram (i.e., 20 degrees of slant gives an italic like character). Units are current angle-specifying units for the system. Default slant is 0 degrees.

Note that many devices are incapable of drawing characters with arbitrary sizes and aspect ratios. The minimum character height may be of the order of 4 or 5 GDU's, aspect ratio may be fixed (e.g., 5x7 or 7x9 dot matrix), and slanting may not be possible at all. You may request size, etc, with this command, but you should then use the CSIZE call to determine the actual sizes for purposes such as computing margins.

Width of character = <height> * <ratio>

Example:

```
400 CSIZE (10,2,45)
```

PLOTTING FUNCTIONS

PEN

PEN (<pen number>)

Where "pen number" is an integer in the range −2 to 4.

The PEN function selects any one of the pens found on the plotting device. This provides you with a convenient way of asking for one of four line types without knowing details of the device used.

PEN "0" selects a "blank" pen (returns all physical pens to holder, or selects a "NOP" mode in a CRT).

Pens 1−4 may be physical pens (presumably of several colors) or may be simulated by using line−types 0,2,3, or 4 (see "LINE") on a device without actual pens. If LINE has been called since the last call to GPON(2), the PEN statement is ignored. If a pen number greater than 4 is used, it will be interpreted modulo 4 (ie. 5=1, 6=2, etc.).

A device with "m" pens will interpret the <pen number> modulo m.

Negative pen numbers are used for certain device dependent functions as on the terminal:

- PEN (0) is a blank pen.
- PEN (−1) is an "eraser" or clear mode.
- PEN (−2) complements the image along its path, such that a second complementing would restore the original display.

PEN(1) = LINE(0)
PEN(2) = LINE(2)
PEN(3) = LINE(3)
PEN(4) = LINE(4)

When a negative pen number is used on a plotter, this is interpreted as a request for a blank pen (PEN(0)).

Example:

```
30 PEN(3)
```

PENUP/PENDN

PENUP
or
PENDN

Where there are no parameters.

PENUP lifts the "pen" and PENDN lowers the "pen." PENUP declares that the following set of PLOT commands describe an object not to be connected to what has been drawn before. Typical use is to do a PENUP before a loop containing PLOT commands. A call to PENDN followed by a PENUP leaves a mark.

Examples:

```
30 PENUP
40 MOVE (100,200)
50 PENDN
```

LINE

LINE [(<linetype> [,<length>])]

LINE selects one of 8 (0-7) predefined line types. "Length" is a pattern repeat distance in GDU's. The default line type is solid (0). If a line type greater than 7 is used, it will be interpreted modulo 7 (ie. 8=1, 9=2, etc.). Default length may depend on device and selected pattern, values in the range of 4-10 GDU's are typical for plotters. On the terminal the length is 8 display dots.

- LINE 0 is always solid lines.
- LINE 1 is always "fainter" than type 0, for example, short or low duty-cycle dashes.
- LINE 0 or 2-4 are all distinct; additional distinct types may also be supplied by some devices.
- LINE types less than zero should not be used.

LINE patterns used by the terminal (3 cycles are shown):

```
         1 length
          <------>
0.   ------------------------  (solid)

1.   - - - - - - - - - - - -   (dim)

2.   ----    ----    ----      (short dash)

3.   ------  ------  ------    (long dash)

4.   ---  -  ---  -  ---  -    (centerline)

5.   .(starting point)      .  (end points)

6.   --- - - --- - - --- - -

7.   ---  ---  ---  --- ---
```

LINE 0 and the standard "fainter" LINE 1 can be used for drawing major and minor grid lines. LINE 0 and 2-4 are used to simulate the four "pens" on devices without color or physical pens.

Devices which support only types 0-6 should map larger type numbers onto standard types 0-6. Since not all devices can allow arbitrary lengths, the device should use the best available length. In some devices the pattern length will also vary line angle (e.g., by a factor of 1.414 for 45 degrees). Typically, pen plotters cannot support dim lines, so a request for LINE 1 results in use of LINE 0 or solid lines.

Note that GSTAT (9,1) will return the pattern repeat distance. If this value is 0, the device dependent default is being used.

PLOT

PLOT (<x coord> , <y coord> [,<pen cntl>])

Where x and y coordinates are type REAL and are interpreted according to the current "units" mode. The pen–control parameter is an integer, and defaults to 1 (moves, then drops). The pen control parameter is interpreted as follows:

```
EVEN:  lift pen (pen-up)
 ODD:  drop pen (pen-down)
   + :  pen change after motion
   - :  pen change before motion
```

Examples:

```
+1  move or draw, then drop pen if up (default)
 2  move or draw, then lift pen if down
 0  move or draw, then lift pen if down
-1  lower pen, then DRAW
-2  lift pen, then MOVE
```

The PLOT function provides absolute data plotting with pen control. PLOT is the preferred pen moving command for automatically generated data, because the pen is under direct program control.

Pen motions will be clipped as shown under CLIP. PLOT commands will be affected by the current vector clipping limits (see CLIP). Clipping allows plotted data to be clipped at the plot boundary ("soft" clip), but to allow labels to be positioned outside the boundary, typically in GDU's.

In the typical case, a PENUP command is executed at the start of a program segment, and the default pen control mode is used in a subsequent plotting loop. The first data point plotted will then be isolated from any prior drawing.

Examples:

```
30 PLOT(5,10)
40 PLOT (Dx,Dy,Spen)
```

MOVE

MOVE (<x coord>, <y coord>)

Where X and Y coordinates are interpreted according to the current units used. A MOVE (X,Y) is equivalent to a PLOT (X,Y,−2).

The MOVE function lifts the pen and moves it to the absolute X,Y coordinate. MOVE is allowed to define a logical pen position beyond the normal clipping limits. If the pen is off−page to the left and a label is started, the first few characters are invisible until the pen moves back within the hard−clipped region. Then the remaining characters are drawn normally.

Example:

```
50 MOVE (5,10)
```

DRAW

DRAW (<x coord>, <y coord>)

Where X and Y coordinates are interpreted according to the current units used. A DRAW (X,Y) is equivalent to a PLOT (X,Y,−1).

The DRAW function drops the pen and moves it (within the soft clip region) to the absolute X,Y coordinate. DRAW allows an easy way of drawing a line from the current pen location to a new location without regard to whether the pen is up or down.

Example:

```
60 DRAW (30,30)
```

RPLOT

RPLOT (<x coord>, <y coord> [,<pen cntl>])

RPLOT provides relative plotting capability with pen control from the last plotted point. RPLOT interprets its coordinate values as being relative to a relocatable origin, whose location is the last point addressed by an absolute PLOT, IPLOT, MOVE, or DRAW, except if PORG has been executed (see PORG). This relocatable coordinate system may also be rotated about this origin relative to the master coordinate system by means of a plot direction (PDIR).

Example:

```
200 RPLOT (10,20,2)
```

IPLOT

IPLOT (<x incr>, <y incr> [,<pen cntl>])

The IPLOT function provides incremental plotting capability with pen control. IPLOT plots incrementally from the current pen position. IPLOT causes the relocatable origin to be updated unless the PORG statement has been used. (See PORG.)

Example:

```
30 IPLOT (5,-5,3)
```

PRINT #0

PRINT #0; text

The PRINT #0 statement is used to perform graphics labeling. The current graphics character size, slant, and origin is used. Appending a ";" (semi-colon) will not suppress a carriage return linefeed in the text line.

A ";" should be used to terminate text to prevent a CR/LF from being sent to the terminal display. If the text contains a CR/LF, it will be sent to the terminal's alphanumeric display. If the text contains a null character, output will be terminated by the null.

Example:

```
60 PRINT #0; "This is the X-axis";
```

PDIR

PDIR (<angle in radians>)
or
PDIR (<x component> , <y component>)

The plotting direction (PDIR) function sets the angle of rotation for relative (RPLOT) and incremental (IPLOT) plotting. PDIR sets the orientation of the local coordinate system explicitly or by supplying a direction vector for the local X axis.

- If only one parameter is supplied, it is assumed to be the counter-clockwise angle in radians from the "right horizontal" direction (0 => towards 3 o'clock, 90 degrees => towards 12 o'clock, etc). Angles are measured in whatever units would be expected by the trigonometric functions of the system at the time of the call.
- If two parameters are supplied, the angle used is that of an incremental vector plotted with X equal to the run and Y equal to the rise in current units.

Example:

```
60 PDIR (5,4)
```

PORG

PORG [(<x coord>,<y coord>)]

PORG defines the local origin for relative plotting (RPLOT). The X and Y coordinates specify a point where the relocatable origin is to be placed. The values are real and are interpreted in the current units mode. Default values are zero for both. The coordinates given override any previously defined points.

When the PORG statement is executed, it prevents the relocatable origin from being updated by MOVE, DRAW, and PLOT statements. Executing a GPON(2) statement will enable the updating of the relocatable origin by a MOVE, DRAW, or PLOT statement..

Example:

```
50 PORG (50,100)
```

INTERACTIVE FUNCTIONS

WHERE

WHERE (<x var.>, <y var.> [,<pen var>])

The WHERE function returns the pen location of the last plotted point. WHERE also determines the logical pen location and whether it is up or down. If the pen is up, pen status will be a 0; if it is down, it will be a 1.

This function can be used to allow an "isolated" software module to determine the pen position. Tne coordinates are type REAL and are in the current units mode. The pen status is an integer with a value of 1 or 0.

Example:

```
70 WHERE (x1,y1,p1)
```

POINT

POINT (<x coord>, <y coord>)

The POINT function positions the cursor under program control at the specified absolute location and specifies the cursor type. The X and Y coordinates are type REAL and are in the current units mode.

Example:

```
90 POINT (90,75)
```

CURSOR

CURSOR (<x var.>, <y var.> [<"z" var.>])

The CURSOR function reads the cursor position without waiting for operator input. The X,Y coordinates are REAL values in current units. The X,Y coordinates are the same as for the POINT function. The Z coordinate is a binary valued integer: 1 = pendown and 0 = penup.

Example:

```
400 CURSOR (X3,Y3,Z)
```

DIGITIZE

DIGITIZE (<x var.>, <y var.> ,<z var.>)

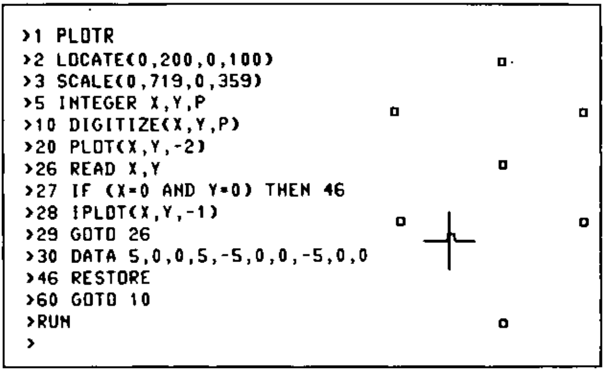
DIGITIZE waits for a user response and then reads the coordinates of the cursor. the X and Y coordinates as well as the pen state (plotters) or key pressed (terminals) are returned in the digitize variables. The coordinates are in the current units system. If a user prompt is desired it must be output using a PRINT statement.

The terminal message line will display the cursor coordinates. The coordinates are updated each time the cursor control keys are used. The display is initially in current units. Pressing Control-G.CURSOR will cause the units to be displayed in machine units. Pressing Control-G.CURSOR again will cause the message line to be cleared.

If a terminal is used as the input device, the graphics cursor will be turned on, indicating the current cursor position. You can then position the cursor using the graphics cursor keys.

Once the cursor is positioned, press any of the ASCII character keys. This will cause the cursor position to be read and returned in the X and Y variables. The ASCII code (0-127) for the key pressed will be returned in the Z variable.

If a plotter is used as the input device, the plotter's ENTER key is used to indicate that the pen has been positioned. The X and Y variables are set to the pen's coordinates and the Z variable is set to a "0" if the pen is up and a "1" if the pen is down.



```
>1 PLOTR
>2 LOCATE(0,200,0,100)
>3 SCALE(0,719,0,359)
>5 INTEGER X,Y,P
>10 DIGITIZE(X,Y,P)
>20 PLOT(X,Y,-2)
>26 READ X,Y
>27 IF (X=0 AND Y=0) THEN 46
>28 IPLOT(X,Y,-1)
>29 GOTO 26
>30 DATA 5,0,0,5,-5,0,0,-5,0,0
>46 RESTORE
>60 GOTO 10
>RUN
>
```

GPMM

<variable> = GPMM (<millimeter value>)

GPMM (GDU's per millimeter) converts millimeters to GDU's. GPMM is a function which returns the number of GDU's corresponding to the number of millimeters specified by the argument. For CRT—like devices, the conversion is typically only an approximation.

GPMM may be imbedded in functions such as CSIZE to allow specifications in millimeters, or may be called once with an argument of 1 to get a multiplier to be used in similar situations. In the latter form it may also be used as a divisor to convert GDU's to mm.

Example:

```
20 N = GPMM (220)
```

Computer Museum

DSIZE

DSIZE (<GDU X limit>,<GDU Y limit>
[,<cell height>,<cell aspect ratio>
[,<X resolution>, <Y resolution>]])

The DSIZE function returns the display size available for the plotting device.

The X,Y limits define G2 in GDU's and G1 is assumed set at 0,0. This is the available display space in GDU's. The character cell dimensions give the height in GDU's and the aspect ratio of the current character. The X,Y resolution parameters in GDU's represent the minimum available step size in each direction.

Example:

```
10 DSIZE (X0,A0,h,Ratio)
```

DSTAT

< string variable > = DSTAT

The DSTAT function returns a string containing the model number of the plotting device (i.e., "9872" for the HP 9872 plotter). If the device is not available, the DSTAT function returns a null string.

Example:

```
20 A$=DSTAT
```

GSTAT

< variable > = GSTAT [(< Index > [,< Subscript >])]

The GSTAT function returns the graphics display status of the plotting device. Index specifies a group from 0-14 and Subscript specifies an element within the group from 0-5. The default parameters are 0 for Index and 0 for Subscript. The table below shows the relationship between the Index and Subscript parameters.

Example:

```
40 N = GSTAT (I,S)
```

* Given that Xu and Yu are coordinate values in a user coordinate system, the equivalent machine coordinates Xm and Ym are:

```
Xm = A * Xu + B
Ym = C * Yu + D
```

The scale factor for X is A. The offset for X is B. The scale factor for Y is C. The offset for Y is D.

In the example below, Xu and Yu are 50,50:

```
A = 10, B = -10
C = 20, D =   0

Xm = 10 * (50) + (-10) =  490
Ym = 20 * (50) +   0   = 1000
```

INDEX	SUBSCRIPT	DESCRIPTION
0		Pen Position
	0	X Pen Position in MU's
	1	Y Pen Position in MU's
1		Pen
	0	Pen State (0=up; 1=down)
	1	Pen Number
2		GDU Space Limits
	0	GX1 (MU's) low value
	1	GX2 (MU's) high value
	2	GY1 (MU's) low value
	3	GY2 (MU's) high value
3		Locate Mapping Points
	0	VX1 (CU's)
	1	VX2 (CU's)
	2	VY1 (CU's)
	3	VY2 (CU's)
4		Soft Clip Limits
	0	SX1 (CU's)
	1	SX2 (CU's)
	2	SY1 (CU's)
	3	SY2 (CU's)
5		User to Machine Scale Factors*
	0	A
	1	B
	2	C
	3	D
6		GDU to Machine Scale Factors*
	0	A
	1	B
	2	C
	3	D
7		Millimeter to Machine Scale Factors*
	0	A
	1	C
8	0	Current Units Mode (0=GDU's; 1=UDU's)
9		Line Type Information
	0	Line Type Argument
	1	Pattern Repeat Distance (GDU's)
10	0	LORG Argument
11		LDIR Argument
	0	Run
	1	Rise
12		PDIR Argument
	0	Run
	1	Rise
13		FXD Argument
	0	X
	1	Y
14		Relocatable Origin
	0	X (CU's)
	1	Y (CU's)

TERMINAL FUNCTIONS

KEY	CODE	FUNCTION
ALPHANUMERIC DISPLAY CONTROL		
↑	ESC A	Cursor up
↓	ESC B	Cursor down
→	ESC C	Cursor right
←	ESC D	Cursor left
BACK SPACE	BS (H^{c})	Cursor left one space
CNTL ↖	ESC F	Cursor home down
— — —	ESC G	Cursor return
↖	ESC h	Cursor home (excluding transmit-only fields)
— — —	ESC H	Home cursor (including transmit-only fields)
RETURN	CR (M^{c})	Move cursor to left margin
— — —	LF (J^{c})	Move cursor down one line
TAB	HT (I^{c}) ESC I	Forward cursor to next tab position
CNTL TAB or CNTL BACK SPACE	ESC i	Back tab
SET TAB	ESC 1	Set tab at the current cursor column
CLEAR TAB	ESC 2	Clear the tab at the current cursor column
CNTL CLEAR TAB	ESC 3	Clear all tabs
CNTL ←	ESC 4	Set left margin

NOTE: Unless otherwise specified, all display control functions apply to the present display-workspace only (i.e., the other three display-workspaces are not affected.)

KEY	CODE	FUNCTION
ALPHANUMERIC DISPLAY CONTROL (Continued)		
CNTL →	ESC 5	Set right margin
CLEAR DSPLY	ESC J	Clear memory from cursor position to end of memory
CNTL CLEAR DSPLY	ESC K	Clear line from the cursor to end of line
ROLL UP	ESC S	Scroll the display up one line
ROLL DOWN	ESC T	Scroll the display down one line
NEXT PAGE	ESC U	Display the next 24 lines of memory
PREV PAGE	ESC V	Display the previous 24 lines of memory
CNTL PREV PAGE	———	Display the next display-workspace
CNTL f1	ESC & d	Turn on display enhance
CNTL f2	ESC [	Start an unprotected field
CNTL f3	ESC]	End an unprotected field or transmit-only field
CNTL f4	ESC W (on)	Turn format mode on. Only unprotected fields can be modified.
CNTL f5	ESC X (off)	
CNTL f6	ESC {	Start transmit-only field
———	ESC 6	Alphabetic only field
———	ESC 7	Numeric only field
———	ESC 8	Alphanumeric field

KEY	CODE	FUNCTION
EDITING		
INSERT LINE	ESC L	Insert a blank line
DELETE LINE	ESC M	Delete line containing cursor
DELETE CHAR	ESC P	Delete character at cursor
CNTL DELETE CHAR	ESC O	Delete character with wraparound from next line
INSERT CHAR & indicator	ESC Q (on) ESC R (off)	Insert succeeding inputs at cursor
CNTL INSERT CHAR	ESC N (on) ESC R (off)	Character Wraparound Mode. Insert succeeding inputs at cursor with wraparound to next line.
TERMINAL CONTROL GROUP		
ESC	␛	Leads off an ASCII escape sequence
CNTL	— — —	Used to generate ASCII control codes and alternate key functions
CAPS LOCK	ESC&k0C(off) ESC&k1C(on)	Upper-case alphabetical lock
MEMORY LOCK & indicator	ESC l (on) ESC m (off)	Memory overflow protect; display lock
AUTO LF	ESC&k0A(off) ESC&k1A(on)	Line Feed with each terminal carriage return
REMOTE	ESC&k0R(off) ESC&k1R(on)	Remote (on-line) operations; otherwise, off-line operation
BLOCK MODE	ESC&k0B(off) ESC&k1B(on)	Block Mode: data displayed but not transmitted until requested; otherwise, terminal is in Character Mode and data transmitted as typed
ENTER	ESC d	Enables block transfers
BREAK	— — —	Transmits BREAK signal to interrupt computer
TRANSMIT indicator	— — —	Data link exists

KEY	CODE	FUNCTION
TERMINAL CONTROL GROUP (Continued)		
DISPLAY FUNCTIONS & indicator	ESC Y (on) ESC Z (off)	Control functions disabled and displayed
CNTL DISPLAY FUNCTIONS	ESC y (on) ESC Z (off)	Monitor Mode: display all codes received from data comm lines
RESET TERMINAL	ESC g	(First press): frees the keyboard and clears I/O operations
RESET TERMINAL	ESC E	(Second press): sets the terminal to power-on state
TEST	ESC z	Terminal Self-Test (no tape test)
ADDITIONAL FUNCTIONS		
— — —	ENQ (E^c)	Enquiry from the computer
— — —	ACK (F^c)	Acknowledge — response to ENQ
— — —	BEL (G^c)	Bell
— — —	ESC)	Define alternate character set: @, A, B, C
— — —	SO (N^c)	Turn on alternate character set
— — —	SI (O^c)	Turn off alternate character set
— — —	DC1 (Q^c)	Block transfer trigger
— — —	DC2 (R^c)	Block transfer enable from terminal
— — —	RS ($\wedge^c$)	Record separator
— — —	US ($\_^c$)	Unit separator
— — —	ESC @	Delay one second
— — —	ESC `	Cursor sensing (screen relative)
— — —	ESC a	Cursor sensing (absolute)
— — —	ESC b	Keyboard enable
— — —	ESC c	Keyboard disable
— — —	ESC d	Block transfer enable from computer (See DC2)

KEY	CODE	FUNCTION
ADDITIONAL FUNCTIONS (Continued)		
---	ESC e	Fast binary read
---	ESC f	Modem hang-up
CNTL NEXT PAGE	ESC j (on) ESC k (off)	Display user-defined soft keys
---	ESC ^	Terminal status
---	ESC ~	Extended status request

TERMINAL CONTROL

KEY	CODE	FUNCTION
---	ESC & a <parameters>	Cursor addressing

Example: Cursor to 12th row 35th column (+, − for relative addressing)

```
ᴱc & a 12r 35C
```

KEY	CODE	FUNCTION
---	ESC & b <parameters> or ESC & c <parameters>	HP diagnostics ONLY
CNTL f1 , < @ through O >	ESC & d <enhancement>	Turn on display enhancement

where: enhancements = @ through O

Example: Select half-bright, blinking, and underline.

```
ᴱc & d M
```

	Enhancement Character															
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Half-Bright									x	x	x	x	x	x	x	x
Under-line					x	x	x	x					x	x	x	x
Inverse Video			x	x			x	x			x	x			x	x
Blinking		x		x		x		x		x		x		x		x
End Enhancement	x															

KEY	CODE	FUNCTION

TERMINAL CONTROL (Continued)

KEY	CODE	FUNCTION
— — —	ESC & f <parameters>	Define soft keys

where:

$$\text{<parameters>} = \{0\text{-}8\}k \left\{\begin{matrix}0\ (\text{normal})\\1\ (\text{local only})\\2\ (\text{transmit only})\end{matrix}\right\} a\ \{0\text{-}8\}d\ \{1\text{-}80\}L\ \text{<label string> <text string>}$$

Example: Assign the string "HELLO-MYACNT␍" to the f1 key. The key should function as normal keyboard input.

␛ & f 1 k 2 a 6d 13 L LOG-ON HELLO-MYACNT␍

KEY	CODE	FUNCTION
— — —	ESC &f <key #>E	Execute the soft key. Key # = 0-8

KEY	DEFAULT	PROGRAMMABLE
RETURN	CR	
f1	ESC p to computer	
f2	ESC q to computer	
f3	ESC r to computer	
f4	ESC s to computer	Up to 80-character sequence for each key (local, transmit or both)
f5	ESC t to computer	
f6	ESC u to computer	
f7	ESC v to computer	
f8	ESC w to computer	

KEY	CODE	FUNCTION

TERMINAL CONTROL (Continued)

———	ESC & g <parameters>	Simulate PA, PF keys (see *Reference Manual)*
———	ESC & k <parameters>	Define latching keys

where:

$$\langle\text{parameter}\rangle = \begin{matrix}0\text{ (up)}\\ 1\text{ (down)}\end{matrix}\left\{\begin{matrix}\text{a (Auto LF)}\\ \text{b (Block Mode)}\\ \text{c (Caps Lock)}\\ \text{r (Remote)}\end{matrix}\right.$$

Example: Block Mode up Remote up Auto LF down Caps Lock down

ᴱc & k 1 a 0 b 1 c 0 R

READ	———	In REMOTE, transfers data from source device to computer. In LOCAL, transfers one file from source device to DISPLAY.
RECORD	———	In REMOTE, transfers data from computer to destination device. In LOCAL, transfers one file from DISPLAY to destination device.
ENTER	———	In REMOTE, enables block transfers. In LOCAL, operates same as RECORD.
———	ESC & s <parameters>	Define strap settings.

KEY **CODE** **FUNCTION**

GRAPHICS CONTROL SEQUENCES

ESC • <control sequence>

b	= Raster dump
d	= Display control
l	= Graphics text label
m	= Mode control
p	= Plot control
r	= Raster dump
s	= Status
t	= Compatibility mode

DISPLAY CONTROL

ESC • d <parameters>

KEY	CODE	FUNCTION
SHIFT CLEAR	a	Clear graphics memory
— — —	b	Set graphics memory
SHIFT G DSP	c	Turn on graphics display
SHIFT G DSP	d	Turn off graphics display
SHIFT A DSP	e	Turn on alphanumeric display
SHIFT A DSP	f	Turn off alphanumeric display
ZOOM	g	Turn on zoom
ZOOM	h	Turn off zoom
ZOOM IN or ZOOM OUT	<size> i	Set zoom size (1-16)
— — —	<x,y> j	Set zoom position
G CURSOR	k	Turn on graphics cursor
G CURSOR	l	Turn off graphics cursor
SHIFT RB LN	m	Turn on rubber band line
SHIFT RB LN	n	Turn off rubber band line

KEY	CODE	FUNCTION
DISPLAY CONTROL (Continued)		
— — —	<x,y> o	Move graphics cursor absolute
◄ , ▲ , ► , ▼	<x,y> p	Move graphics cursor incremental
— — —	q	Turn on alphanumeric cursor
— — —	r	Turn off alphanumeric cursor
[illegible] TEXT	s	Turn on graphics text mode
STOP	t	Turn off graphics text mode
	z	NOP

Example: Clear the graphics display, position the cursor at x=100, y=100, turn the cursor on, and zoom to 4 times.

␛ * d a 100,100o k 4i G

GRAPHICS LABEL

ESC * l <text label><␍, ␍␊, ␊␍, or ␊>

Example: Send the text "X=TIME, Y=TEMP"

␛ * l X=TIME, Y=TEMP ␍␊

KEY	CODE	FUNCTION
	VECTOR DRAWING MODE ESC * m <parameters>	
— — —	<mode> a	Select drawing mode (0-4)*
— — —	<line type> b	Select line type (1-11)**
— — —	<pattern scale> c	Define line pattern (2 bytes)
— — —	<pattern> d	Define area shading pattern (8 bytes)
— — —	<x1,y1,x2,y2> e	Fill area, absolute
— — —	<x1,y1,x2,y2> f	Fill area, relocatable
— — —	<x,y> j	Set relocatable origin
— — —	k	Set relocatable origin to current pen position
— — —	l	Set relocatable origin to graphics cursor position
SHIFT T SIZE	<size> m	Set graphics text size (1-8)
SHIFT T ANG	<rotation> n	Set graphics text orientation (1-4)
SHIFT T ANG	o	Turn on text slant
SHIFT T ANG	p	Turn off text slant
— — —	<0-9> q	Set graphics text origin
— — —	r	Set graphics defaults
	z	NOP

* 0 (no effect), 1 (clear), 2 (set), 3 (complement), 4 (jam)

** 1 (solid line)
2 (user line pattern)
3 (user area pattern)
4 (line #1)
5 (line #2)
6 (line #3)
7 (line #4)
8 (line #5)
9 (line #6)
10 (line #7)
11 (point plot)

Example: Select the set drawing mode, a graphics text size of 2 and slanted. Set the text to be center justified. ␛ * m 1a 2m o 4Q

KEY	CODE	FUNCTION
	PLOTTING COMMANDS	
	ESC • p <parameters>	
— — —	a	Lift the pen
— — —	b	Lower the pen
— — —	c	Use graphics cursor as new point
— — —	d	Draw a point at the current pen position and lift the pen
— — —	e	Set relocatable origin to the current pen position
— — —	f	Data is ASCII absolute
— — —	g	Data is ASCII incremental
— — —	h	Data is ASCII relocatable
— — —	i	Data is absolute
— — —	j	Data is short incremental
— — —	k	Data is incremental
— — —	l	Data is relocatable
— — —	z	NOP

Example: Draw a box 25 units wide and 10 units high, beginning at x=100, y=50.

```
␛ • p a f 100 50 g 25,0 0,10 -25,0 0,-10Z
```

KEY	CODE	FUNCTION
	GRAPHICS STATUS	
	ESC • s <parameter> ^	
———	1	Read device I.D.
———	2	Read pen position
———	3	Read graphics cursor position
———	4	Read cursor position and wait for key
———	5	Read display size
———	6	Read graphics capabilities
———	7	Read graphics text status
———	8	Read zoom status
———	9	Read relocatable origin
———	10	Read reset status
———	11	Read area shading
———	12	Read dynamics

Example: Read text status.

ᴱc • s 7 ^ DC1

KEY	CODE	FUNCTION
	COMPATIBILITY MODE	
	ESC • t <parameter>	
———	<0/1/2> a	Set graphics input terminator (0=CR, 1=CR EOT, 2=none)
———	<0/1> b	Set Page Full Break strap (0=out, 1=in)
———	<0/1> c	Set Page Full Busy strap 0=out, 1=in)
	z	NOP

Keyboard Interface switches:

P open = Scaled compatibility mode.
Q open = Unscaled compatibility mode.

Example: Select a CR input terminator and set the Page Full Busy strap.

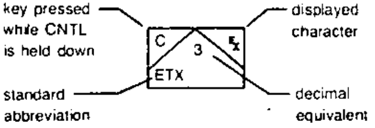
ᴱc • t 0a 1C

Computer Museum

BIT 765 4321	CONTROL (CNTL) CHARACTERS 000	 001	DISPLAYABLE CHARACTERS 010	 011	 100	 101	 110	 111
0000	@ 0 NU NUL	P 16 DL DLE	SP	0	@	P	`	p
0001	A 1 SH SOH	Q 17 D1 DC1	!	1	A	Q	a	q
0010	B 2 SX STX	R 18 D2 DC2	"	2	B	R	b	r
0011	C 3 EX ETX	S 19 D3 DC3	#	3	C	S	c	s
0100	D 4 ET EOT	T 20 D4 DC4	$	4	D	T	d	t
0101	E 5 EQ ENQ	U 21 NK NAK	%	5	E	U	e	u
0110	F 6 AK ACK	V 22 SY SYN	&	6	F	V	f	v
0111	G 7 🔔 BEL	W 23 EB ETB	'	7	G	W	g	w
1000	H 8 BS BS	X 24 CN CAN	(	8	H	X	h	x
1001	I 9 HT HT	Y 25 EM EM	)	9	I	Y	i	y
1010	J 10 LF LF	Z 26 SB SUB	*	:	J	Z	j	z
1011	K 11 VT VT	[27 EC ESC	+	;	K	[	k	{
1100	L 12 FF FF	\ 28 FS FS	,	<	L	\	l	¦
1101	M 13 CR CR	] 29 GS GS	-	=	M	]	m	}
1110	N 14 SO SO	30 RS RS	.	>	N	^	n	~
1111	O 15 SI SI	31 US US	/	?	O	_	o	■

AK — ACKNOWLEDGE
🔔 — BELL
BS — BACKSPACE
CN — CANCEL LINE
CR — CARRIAGE RETURN
D1 — DEVICE CONTROL 1
D2 — DEVICE CONTROL 2
D3 — DEVICE CONTROL 3
D4 — DEVICE CONTROL 4
■ — DELETE
DL — DATA LINK ESCAPE
EM — END OF MEDIUM
EQ — ENQUIRY
ET — END OF TRANSMISSION
EC — ESCAPE
EB — END OF TRANSMISSION BLOCK
EX — END OF TEXT
FF — FORM FEED
FS — FILE SEPARATOR
GS — GROUP SEPARATOR
HT — HORIZONTAL TAB
LF — LINE FEED
NK — NEGATIVE ACKNOWLEDGE
RS — RECORD SEPARATOR
SI — SHIFT IN
SO — SHIFT OUT
SP — SPACE
SH — START OF HEADING
SX — START OF TEXT
SB — SUBSTITUTE
SY — SYNCHRONOUS IDLE
US — UNIT SEPARATOR
VT — VERTICAL TAB

Control Character Legend:



	ESCAPE SENT FIRST					
BIT 7 6 5 4321	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0000	SP	0	@ DELAY 1 SEC	P DELETE CHAR	` CURSOR RELATIVE SENSE	p f_1
0001	!	1 SET TAB	A ↑ CURSOR UP	Q INSERT CHAR ON	a CURSOR ABSOLUTE SENSE	q f_2
0010	"	2 CLEAR TAB	B ↓ CURSOR DOWN	R INSERT CHAR OFF	b KEYBOARD ENABLE	r f_3
0011	#	3 CLEAR ALL TABS	C → CURSOR RIGHT	S ROLL UP	c KEYBOARD DISABLE	s f_4
0100	\$	4 SET LEFT MARGIN	D ← CURSOR LEFT	T ROLL DOWN	d ENTER	t f_5
0101	%	5 SET RIGHT MARGIN	E RESET TERMINAL	U NEXT PAGE	e BINARY READ	u f_6
0110	& PARAMETER SEQUENCE	6 START ALPHA FIELD	F CURSOR HOME DOWN	V PREV PAGE	f MODEM DIS CONNECT	v f_7
0111	'	7 START NUMERIC FIELD	G CURSOR RETURN	W FORMAT MODE ON	g SOFT RESET	w f_8
1000	(	8 START ALPHNUM FIELD	H HOME CURSOR (SEE NOTE 3)	X FORMAT MODE OFF	h HOME CURSOR (SEE NOTE 3)	x DATA COM SELF TEST
1001	) DEFINE CHAR SET	9	I HORIZONTAL TAB	Y DISPLAY FUNCTIONS ON	i BACK TAB	y MONITOR MODE ON
1010	* GRAPHICS SEQUENCE	:	J CLEAR DSPLY	Z DISPLAY FUNCTIONS OFF	j SOFT KEY DISPLAY ON	z TERMINAL SELF TEST
1011	+	;	K ERASE TO END OF LINE	[START UNPROTECT FIELD	k SOFT KEY DISPLAY OFF	{ START XMIT ONLY FIELD
1100	, COMMAND SEQUENCE	<	L INSERT LINE	\	l MEMORY LOCK ON	¦
1101	–	=	M DELETE LINE	] END UNPROTECT FIELD	m MEMORY LOCK OFF	}
1110	.	>	N INSERT CHAR W/WRAP ON	^ TERM PRIMARY STATUS	n	~ SEND SECONDARY STATUS
1111	/	?	O DELETE CHAR W/WRAP	_ INSERT NON-DISP TERMINATR	o	DEL

1. Lower case letter, lower case symbol, and control character codes are generated by standard terminal, but associated characters are not displayed on the screen. Press TAPE TEST key for displayable character set.

2. Single character escape sequences and control codes not listed with a function are neither acted upon nor displayed.

3. ESC H homes cursor including transmit-only fields. ESC h homes cursor excluding transmit-only fields.

DEVICE CONTROL

NOTE 1: Uppercase characters in the command syntax are acceptable abbreviations (see "Abbreviated Command Sequence"). Characters in brackets "[ ]" are optional.

NOTE 2: Shaded escape sequences are compatible with HP 2641A, HP 2645A and HP 2648A terminals; however, it is recommended that the ESC,c command sequences be used.

NOTE 3: When using the ESC & p format, the <x> and <y> values refer to source device and destination device numbers:

1 = Left tape
2 = Right tape
3 = Display
4 = Printer

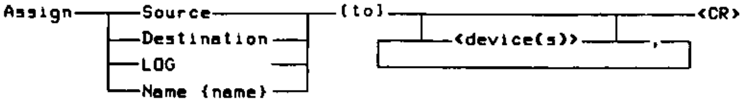
LEGEND

<device> defined as:
Left tape
Right tape
EXternal printer
SHared printer#<n> (where n>4 i.e., SH#5)
DIsplay
TErminal#<p>
Hp-ib#<p>[#<s>[#<m>]]
where p = primary address
s = secondary address
m = module address
DAtacomm
Null
Graphics

<name> defined as:
Source (only one may be selected)
Destination
LOG
user specified name

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

DEVICE CONTROL (Continued)



Example: Assign "Plotter" to Hpib device number 4.

`Assign Name PLOTTER to Hpib#4 <CR>`

Assign Source {to} Left tape ␍	ESC,cA S L ␍ or ESC &p 1S	Assigns left tape as source device.
Assign Source {to} Right tape ␍	ESC,cA S R ␍ or ESC,&p 2S	Assigns right tape as Source device.
Assign Source {to} Display ␍	ESC,cA S DI ␍ or ESC &p 3S	Assigns display as source device.
Assign Source {to} Hp-ib#<p> ␍	ESC,cA S H#<p> ␍	Assigns an HP-IB device as source device.
Assign Source {to} Graphics ␍	ESC,cA S G ␍	Assigns graphics memory as source device.

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION
	DEVICE CONTROL (Continued)	
Assign Destination {to} Left tape ␍	ESC,cA D L ␍ or ESC,&p 1D	Assigns left tape as Destination device.
Assign Destination {to} Right tape ␍	ESC,cA D R ␍ or ESC &p 2D	Assigns right tape as Destination device.
Assign Destination {to} DIsplay ␍	ESC,cA D DI ␍ or ESC &p 3D	Assigns display as Destination device.
Assign Destination {to} EXternal printer ␍	ESC,cA D EX or ESC &p 4D	Assigns local printer as Destination device.
Assign Destination {to} SHared printer#5 ␍	ESC,cA D SH#5␍	Assigns shared printer as Destination device.
Assign Destination {to} Hp-ib#<n>#<n>#<n> ␍	ESC,cA D H#<n> ␍	Assigns an HP-IB device(s) as Destination device(s).
Assign Destination {to} Graphics ␍	ESC,cA D G ␍	Assigns graphics memory as Destination device.
Assign LOG {to} <device> ␍	ESC,cA LOG <device> ␍	Assigns any device as LOG device.
Assign Name <user-defined name> {to} <device> ␍	ESC,cA N <name> <device> ␍	Assigns a user-specified name to any device.

```
BYE────────<CR>
```

BYE ↵	ESC,cBYE ↵	Defaults user.group name to "user.group" on shared printer listings.

```
CLOse──────Window#<n>────────<CR>
```

CLOse Window#5 ↵	ESC,cCLO W#5 ↵	Removes the Message Line on the display.
CLOse Window#6 ↵	ESC,cCLO W#6 ↵	Removes the Command Line on the display.
CLOse Window#7 ↵	ESC,cCLO W#7 ↵	Removes the Soft Key Label Line on the display.

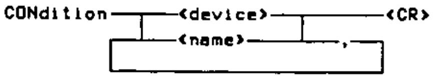
```
COmpare──┬──All──┬──[of]──┬──<device>──┬──[to]──┬──<device>──┬──<CR>
         ├──File─┤        ├──<name>────┘        ├──<name>────┘
         ├──Line─┘        └──<CR>               └──<CR>
         └──<CR>
```

Example: Compare a file on the display with a file on a device with a user name of "SALES".

```
COmpare File of Display to SALES <CR>
```

COmpare All {of} <device> {to} <device> ↵	ESC,cCO A <device><device> ↵ or ESC &p xs yd 1M	Compare data between devices. (All)
COmpare File {of} <device> {to} <device> ↵	ESC,cCO F <device> <device> ↵ or ESC &p xs yd 1F	Compare data between devices. (File)

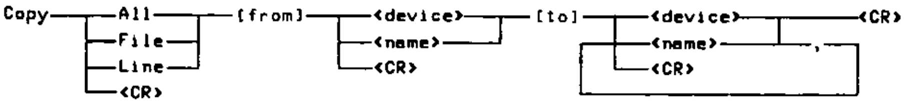
COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION
	DEVICE CONTROL (Continued)	
COmpare Line {of} <device> {to} <device> ␍	ESC,cCO L <device> <device> ␍ or ESC &p xs yd 1B	Compare data between devices. (Line)
COmpare All ␍	ESC,cCO A ␍ or ESC &p 1M	Compare data between previously-assigned Source and Destination devices. (All)
COmpare ␍	ESC,cCO F ␍ or ESC &p 1F	Compare data between previously-assigned Source and Destination devices. (File)
COmpare Line ␍	ESC,cCO L ␍ or ESC &p 1B	Compare data between previously-assigned Source and Destination devices. (Line)
COmpare All or File or Line {of}<name> {to}<name> ␍	ESC,cCO <A or F or L> <name> <name> ␍	Compares data between user-specified device names.



Example: Assume the left tape is assigned the user name: "DATA".
Condition left tape with a user name of "DATA" and the right tape.

```
CONdition DATA, Right tape <CR>
```

CONdition Left tape or Right tape ¶	ESC,cCON L or R or ESC &p 1u or 2u 4C	Conditions left tape or right tape.



Example: Copy the entire contents of the left tape to the shared printer.

```
Copy·All from Left tape to SHared printer#5 <CR>
```

NOTE: If you want to copy graphics data (i.e. binary), the TRANSFER command sequence must be used.

Copy File {from} Source {to} Display ¶	ESC,c C F S DI	Transfers one file from Source device to display.
Copy File {from} Display {to} Destination ¶	ESC,c C F DI D	Transfers all data from display workspace to Destination device.

Computer Museum

DEVICE CONTROL (Continued)

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION
Copy All ␍	ESC,cC A ␍ or ESC &p M	All files (current position) from previously-assigned Source device are transferred to previously-assigned Destination device.
Copy ␍	ESC,cC ␍ or ESC &p xs yd F	One file (current position) from previously-assigned Source device is transferred to previously-assigned Destination device.
Copy Line ␍	ESC,cC L ␍ or ESC &p xs yd B	One line (current position) from previously-assigned Source device is transferred to previously-assigned Destination device.
Copy All {from} <device> {to} <device(s)> ␍	ESC,cC A <device> <device> ␍ or ESC &p xs yd M	All files (current position) from a specified device are transferred to one or more specified device(s).
Copy File {from} <device> {to} <device(s)> ␍	ESC,cC F <device> <device> ␍ or ESC &p xs yd F	One file (current position) from a specified device is transferred to one or more specified device(s).
Copy Line {from} <device> {to} <device(s)> ␍	ESC,c C L <device> <device> ␍ or ESC &p xs yd B	One line (current position) from a specified device is transferred to one or more specified device(s).
Copy All {from} <name> {to} <name> ␍	ESC,cC A <name> <name> ␍	All files (current position) from user-assigned device name are transferred to user-assigned device name(s).

Copy File {from} <name> {to} <name> ␍	ESC,cC F <name> <name> ␍	One file (current position) from user-assigned device name is transferred to user-assigned device name(s).
Copy Line {from} <name> {to} <name> ␍	ESC,cC L <name> <name> ␍	One line (current position) from user-assigned device name is transferred to user-assigned device name(s).

```
Display      Window#<n>      <CR>
```

Example: Display the message "Insert the tape into the left slot."

```
Display Window#5<CR>
Insert the tape into the left slot <CR>
```

Display Window#<n> ␍	ESC,cDI W#<n> ␍	Display workspace 1, 2, 3, or 4, Message Line (5), Command Line (6), or Soft Key Label Line (7).

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

DEVICE CONTROL (Continued)

```
Disable ─┬── Edit ── [mode] ───┬──<CR>
or       ├── Record [mode] ────┤
Enable   └── Verify [mode] ────┘
```

Enable/Disable Edit ␍	ESC,cE or D E ␍	Turns Edit Mode on/off.
Enable/Disable Verify ␍	ESC,cE or D Ve ␍ ESC &p 10C (on) ESC &p 9C (off)	Toggles Verify Mode. (Reads each record transferred and compares it with the original.
Enable/Disable Record ␍	ESC,cE or D R ␍	In REMOTE, transfers data from computer to destination device. In LOCAL, transfers one file from DISPLAY to destination device.

```
EXecute ─┬── <device> ──┬──<CR>
         └── <name> ────┘
```

Example: Execute the list of commands in the present file on the right tape.

```
EXecute Right tape <CR>
```

EXecute <device or name> ␍	ESC,cEX <device or name> ␍	Executes commands stored on the specified device.

```
EXIT———Command file———<CR>
       Application
```

EXIT Command File ␍	ESC,cEXIT C ␍	Terminates a command file.
EXIT Application ␍	ESC,cEXIT A ␍	Terminates a subsystem (e.g., BASIC).

```
Find—┬—File ————<[+|-] n>—┬——[on]—┬┬—<device>——┬———<CR>
     └—End {of}——————Data—┘       │└—<name>————┴—,—┐
                                  └────────────────┘
```

Example: Find the seventh file on the left tape.

```
Find File 7 on Left tape <CR>
```

Example: Position the right tape forward 3 files.

```
Find File +3 on Right tape <CR>
```

Example: Find end-of-data on tape with user name "MEMO".

```
Find End of Data on MEMO <CR>
```

Find File<+, −n> {on} Left Tape ␍	ESC,cF F <+, −n> L ␍ or ESC &p (+, −n)p 1u 2C	Positions left tape to a relative (+, −n) or absolute (n) file.
Find File <+, −n> {on} Right tape ␍	ESC,cFF <+,−n> R ␍ or ESC &p (+, −n)p 2u 2C	Positions right tape to a relative (+, −n) or absolute (n) file.

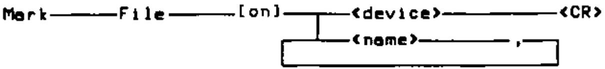
COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

DEVICE CONTROL (Continued)

Find End {of} Data {on} Left tape ␍3	ESC,cF E D L ␍ or ESC &p 1u 3C	Finds end-of-data on left tape.
Find End {of} Data {on} Right tape ␍	ESC,cF E D R ␍ or ESC &p 2u 3C	Finds end-of-data on right tape.

```
HELLO———<CR>
```

HELLO <user.group> ␍	ESC,cHELLO <user.group> ␍	Enters user.group name to be used on shared printer listing.



Mark File Header {on} Left tape ␍	ESC,cM F H L ␍	Write a file mark on left tape.

Mark File Header {on} Right tape ␍	ESC,cM F H R ␍	Writes a file mark on right tape.
Mark File Header {on} EXternal printer ␍	ESC,cM F H EX ␍	Causes a Form Feed on external printer.
Mark File Header {on} SHared printer#5 ␍	ESC,cM F H SH#5 ␍	Causes a Form Feed on shared printer.
Mark File Header {on} <name> ␍	ESC,cM F H <name> ␍	Writes a file mark or causes a Form Feed on device that has a user-assigned name.

```
REPort Status [of]-------Command-------<CR>
```

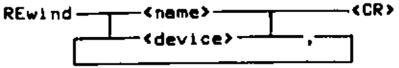
REPort Status {of} Command ␍	ESC,cREP S C ␍	Sends result of last command executed or current device state to the datacomm, if in remote. If last command was EXecute, REPort will be returned after EXecute terminates.

```
RESume--+---Command file---+---<CR>
        +---Application----+
```

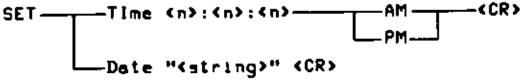
RESume Command file ␍	ESC,cRES C ␍	Resumes execution of the command file after a SUspend command is encountered.
RESume Application ␍	ESC,cRES A ␍	Resume operation of the subsystem after a SUspend command is encountered.

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

DEVICE CONTROL (Continued)



REwind Left tape ␍	ESC,cRE L ␍ or ESC &p 1u 0C	Rewinds left tape.
REwind Right tape ␍	ESC,cRE R or ESC &p 2u 0C	Rewinds right tape.
REwind <name> ␍	ESC,cRE <name> ␍	Rewinds tape that has a user-assigned name.



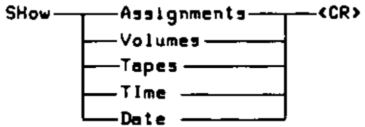
Example: Set time to 2:15:45 PM.

```
SET TIme 2:15:45 PM <CR>
```

Example: Set date to Tuesday, August 8, 1978.

```
SET Date "Tuesday, August 8, 1978" <CR>
```

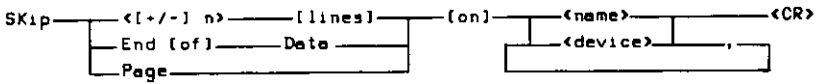
SET TIme <n>:<n>:<n>: AM (or PM) ␍	ESC,cSET TI <n>:<n>:<n> AM or PM ␍	Sets time on terminal.
SET Date "<date string>" ␍	ESC,cSET D "<date string>" ␍	Sets data on terminal. (Maximum length of date string is 30 characters.



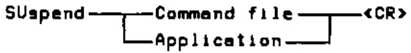
SHow Assignments ␍	ESC,cSH A ␍	Lists current Source, Destination, and Log device assignments on Log device.
SHow Volumes ␍	ESC,cSH V ␍	Lists HP-IB device addresses on log device.
SHow Tapes ␍	ESC,cSH T ␍	Lists current tape file and remaining tape space on log device.
SHow TIme ␍	ESC,cSH TI ␍	Lists current time on log device.
SHow Date ␍	ESC,cSH D ␍	Lists current date on log device.

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

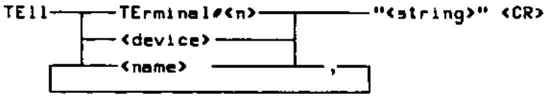
DEVICE CONTROL (Continued)



SKip <+, −n> {lines} {on} Right tape ␍	ESC,cSK <+, −n> R ␍ or ESC &p (+, −n)p 2u 1C	Positions right tape to a relative (+, −n) line.
SKip <+, −n> {lines} {on} EXternal printer ␍	ESC,cSK <+, −n> EX ␍ or ESC &p (+, −n)p 4u 1C	Positions external printer to a relative (+, −n) line.
SKip <+, −n> {lines} {on} SHared printer#5 ␍	ESC,cSK <+, −n> SH#5 ␍	Positions SHared printer#5 to a relative (+, −n) line.
SKip <+, −n> {lines} {on} <name> ␍	ESC,cSK <+, −n> <name> ␍	Positions device that has user-assigned device name to a relative (+, −n) line.
SKip Page {on} EXternal printer ␍	ESC,cSK P EX ␍ or ESC &p (n)p 4u 2C	Positions external printer to a top-of-form.
SKip Page {on} SHared printer#5 ␍	ESC,cSK P SH#5	Positions shared printer to a top-of-form.
SKip End {of} Data {on} Left tape ␍	ESC,cSK E D L ␍	Positions left tape beyond end-of-data mark.
SKip End {of} Data {on} Right tape ␍	ESC,cSK E D R ␍	Positions right tape beyond end-of-data mark.



SUspend Command file ␍	ESC,cSU C ␍	Suspends execution of a command file.
SUspend Application ␍	ESC,cSU A ␍	Suspends operation of the subsystem.



Example: Tell the right tape: "End of program."

```
TEll Right tape "End of Program" <CR>
```

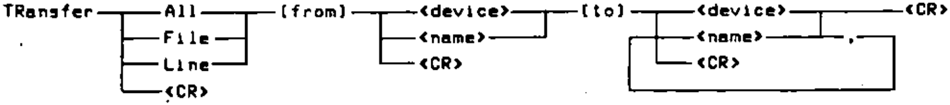
TEll <device> "<string>" ␍	ESC,cTE <device> "<string>" ␍	Transfers data string to the specified device.
TEll TErminal#<n> "<string>" ␍	ESC,cTE TE#<n> "<string>" ␍	Transfers data string to the specified terminal in network.

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION

DEVICE CONTROL (Continued)

```
Test ─┬────────────────┬─<CR>
      ├TErminal#<p>────┤
      ├Hp-ib#<m>───────┤
      ├DAtacomm────────┤
      └Tapes ──────────┘
```

Test ␍	ESC,cT ␍ or ESC z	Tests terminal.
Test DAtacomm ␍	ESC,cT DA ␍ or ESC x	Tests terminal datacomm. (Hood connector must be used.)
Test Tapes ␍	ESC,cT T ␍ or ESC &p (1 or 2)u 7C	Tests terminal cartridge tape units.
Test TErminal#<p> ␍	ESC,cT TE#<p> ␍	Tests data path to other terminal in network.
Test Hp-ib#<m> ␍	ESC,cT H#<m> ␍	Tests a specific HP-IB interface PCA in terminal.



Example: Copy the contents of graphics memory to the left tape.

```
TRansfer File from Graphics to Left tape <CR>
```

NOTE: The TRANSFER command sequence copies data in 8-bit binary form. For 7-bit ASCII transfers, use the COPY command sequence.

TRansfer File {from} Source {to} Display ␍	ESC,cTR F S DI ␍	In REMOTE, transfers data from Source device to computer. In LOCAL, transfers one file from Source device to display.
TRansfer File {from}Display {to} Destination ␍	ESC,cTR F DI D ␍	In REMOTE, transfers data from computer to Destination device. In LOCAL, transfers all data from display workspace to Destination device.
TRansfer All ␍	ESC,cTR A ␍	All files (current position) from previously-assigned Source device are transferred to previously-assigned Destination device.
TRansfer ␍	ESC,cTR ␍	One file (current position) from previously-assigned Source device is transferred to previously-assigned Destination device.
TRansfer Line ␍	ESC,cTR L ␍	One line (current position) from previously-assigned Source device is transferred to previously-assigned Destination device.
TRansfer All {from}<device>{to}(device(s)> ␍	ESC,cTR A <device> <device> ␍	All files (current position) from a specified device are transferred to one or more specified device(s).

DEVICE CONTROL (Continued)

COMMAND SYNTAX <command sequence>	ESC,c<abbreviated command sequence> ␍ or ESC &p <parameters>	DESCRIPTION
TRansfer File {from}<device>{to}<device(s)> ␍	ESC,cTR F <device> <device> ␍	One file (current position) from a specified device is transferred to one or more specified device(s).
TRansfer Line {from}<device>{to}<device(s)> ␍	ESC,cTR L <device> <device> ␍	One line (current position) from a specified device is transferred to one or more specified device(s).
TRansfer All {from}<name>{to}<name> ␍	ESC,cTR A <name> <name> ␍	All files (current position) from user-assigned device name are transferred to user-assigned device name(s).
TRansfer File {from}<name>{to}<name> ␍	ESC,cTR F <name> <name> ␍	One file (current position) from user-assigned device name is transferred to user-assigned device name(s).
TRansfer Line {from}<name>{to}<name> ␍	ESC,cTR L <name> <name> ␍	One line (current position) from user-assigned device name is transferred to user-assigned device name(s).

KEY	CODE	FUNCTION
	MULTIPLOT	
MULTIPLOT	ESC &f 8E	Turn MULTIPLOT on.
MULTIPLOT or SYSTEM	RS (as first byte of line)	Turn MULTIPLOT off.
SHIFT AXIS	ESC &f 7E	Draw MULTIPLOT axes. (Linear charts only.)
———	ESC W ESC,c DI W#2␍ ESC H ESC J	Clear menu.
SHIFT MULTIPLOT MENU	ESC,c DI W#2␍	Turn menu on.
SHIFT MULTIPLOT MENU	ESC,c DI W#1␍	Turn menu off.

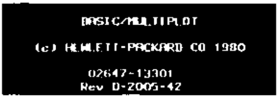
Running Multiplot

The MULTIPLOT programs execute under control of 2647A BASIC.

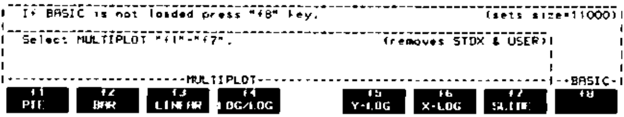
To load and execute a MULTIPLOT program, do the following steps:

Step 1. Place the Terminal BASIC/MULTIPLOT Tape in the left tape slot.

Step 2. With the REMOTE key up (Local Mode), press READ. The following display will appear on the screen.



```
BASIC/MULTIPLOT
(c) HEWLETT-PACKARD CO 1980
02647-13301
Rev D-2005-42
```



```
I+ BASIC is not loaded press "f8" key.                      (sets size=11000)
Select MULTIPLOT "f1"-"f7".                      (removes STDX & USER)
-------------------MULTIPLOT-------------------------------------  --BASIC-
```

Step 3. If you are in the midst of an interactive session on a remote computer and you will be filling in menus and/or plotting data from a program executing in the host computer, put the REMOTE key back down (Remote Mode).

Step 4. If the BASIC Interpreter is not already loaded, press **f8**.

Step 5. Choose the desired plotting function and then press the appropriate softkey (**f1** — **f6**).

Controlling Multiplot from a Computer

To fill the entire menu from a program executing in the host computer, first clear the menu using the following code sequence:

```
␛ W ␛,c DI W/2 ␍ ␛ H ␍ J
```

Then transmit the menu data in order line-by-line, left to right within each line. If a particular data item does not entirely fill the associated menu field, transmit a TAB code (␉) to cause the next data item to be directed to the next menu field. Similarly, to leave a particular field blank and skip to the next menu field, transmit an ␉ code.

To change selected fields within a completed menu, first position the cursor at the beginning of the first data field in the menu using the following code sequence:

```
␛ W ␛,c DI W/2 ␍ ␛ H
```

Then, using ␉ codes as needed to position the cursor at the start of the desired fields, transmit the selected data items. For example, to change the content of the second and fifth fields of a Bar chart menu (the "Main" and "Y Axis" fields) you would use the following sequence:

`␛ W ␛,c DI W/2 ␍ ␛ H`	This sequence positions the cursor at the first character position of the first menu field.
`␉ Main Title`	This sequence moves the cursor to the first position of the second menu field and enters the string "Main Title" into that field.
`␉␉␉ Y-Axis Label`	This sequence moves the cursor to the first position of the fifth menu field and enters the string "Y-Axis Label" into that field.

To run any of the MULTIPLOT programs remotely from a program executing in a host computer, the MULTIPLOT data cartridge must be present in the left tape slot of the terminal.

To load terminal's softkeys with the MULTIPLOT sequences:

```
␛,c C F L DI ␍
```

To load Pie chart program:

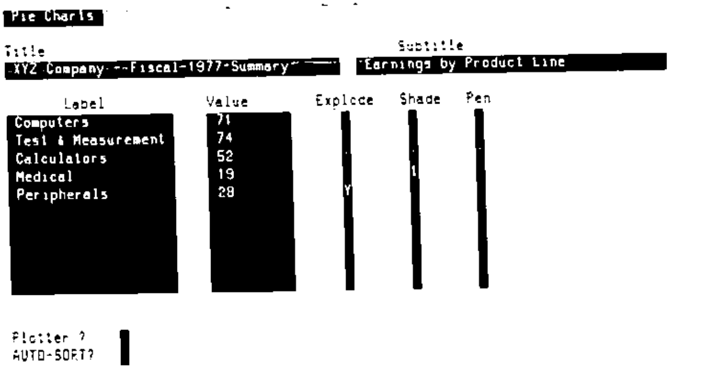
```
␛&f1E
```

To load Bar chart program:

```
␛&f2E
```

To load Linear chart program:

```
␛&f3E
```



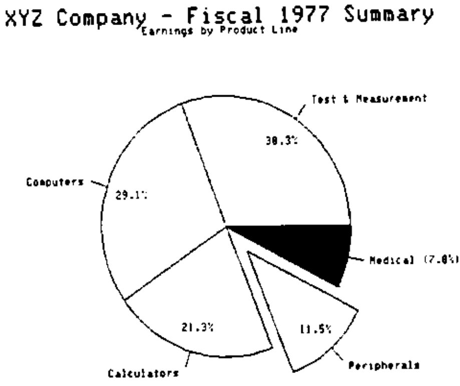
Pie Charts

Title
XYZ Company --Fiscal 1977 Summary

Subtitle
Earnings by Product Line

Label	Value	Explode	Shade	Pen
Computers	71			
Test & Measurement	74			
Calculators	52			
Medical	19		1	
Peripherals	28	Y		

Plotter ?
AUTO-SORT?



The following sequence fills the Pie chart menu (as shown above) and then plots the Pie chart:

```
␛ W ␛,c DI W#2 ␍ ␛ H ␛ J
XYZ Company _ Fiscal 1977 Summary ␉ Earnings by Product Line ␉
Computers ␉ 71 ␉␉␉␉ Test & Measurement ␉ 74 ␉␉␉␉
Calculators ␉ 52 ␉␉␉␉ Medical ␉ 19 ␉␉ 1 ␉
Peripherals ␉ 28 ␉ Y ␛&fBE
```

The same technique is used for filling in the Bar and Linear chart menus remotely.

INDEX

ABS(X) 38
AGL 43
AGL Terminology 45
ASCII Code Chart A-14
ASSIGN 15
Assign command A-17
ATN(X) 39
AUTO 5
AXES 71

BYE command A-19

CALL 16
CHR$(X) 40
CLIP 64
CLIPOFF/CLIPON 65
Clipping 49
CLOse command A-19
COMMAND 16
Command syntax A-17
Commands 5
COmpare command A-19
Computation, direct 4
CONdition command A-21
Copy command A-21
COS(X) 39
CSAVE 6
CSENA(R,C) 38
CSENS(R,C) 38
CSIZE 79
CURSOR 87

DATA 17
DELETE 6
DIGITIZE 88
DIM 17
Direct Computation 4
DIsplay command A-23
DRAW 84
DSIZE 89
DSPIN$(L,X) 42
DSTAT 90

Edit Mode Command A-24
END 17
ERRL 42
ERRN 42
ERROR 18
Escape sequences A-1
EXecute command A-24
EXIT 6
EXIT command A-25
EXP(X) 38
EXTEND 7

Find command A-25
FOR NEXT 18
FRAME 75
FRE 42
FXD 76

GCLR 59
GDU's 52
GET 7
GETDCM ON/OFF 19
GETDCM(S$) 42
GETKBD ON/OFF 19
GETKBD(X) 42
GO 7
GOSUB 20
GOTO 20
GPMM 89
GPON 56
Graph limits 46
Graphic display space 47
GRID 74
GSTAT 90

Hard clip limits 49
HELLO command A-26

IF THEN ELSE 20
IMAGE 21
INPUT 21
Input/output functions 42
Installing Terminal BASIC 1
INT(X) 39
INTEGER 21
IPLOT 85

KEYCDE 22

LAXES 73
LDIR 78
LEN(S$) 40
LET 26
LGRID 75
LIMIT 58
LIN(N) 41
LINE 82
LINPUT 26
LINPUT# 26
LIST 8
LOCATE 60
LOG(X) 39
Logical address space 45
LONG 26
LONG(X) 39
LORG 76
LXAXIS 69
LYAXIS 70

MARGIN 61
Mark file command A-26
Mechanical limits 46
MERGE 8
Metric units 52
MOVCA(R,C) 41
MOVCR(R,C) 41
MOVCS(R,C) 41
MOVE 84
MSCALE 64
Multiplot A-35

NUM(S$) 40
Numeric Functions 38

OFF KEY# 27
ON 28
ON END# 27
ON ERROR 29
ON KEY# 30

PDIR 86
PEN 80
PENUP/PENDN 81
PLOT 83

PLOTR 55
POINT 87
PORG 86
POS(S1$,S2$) 40
PRINT 30
Print functions 41
PRINT USING 32
PRINT# 31
PRINT#0 85
PROC$(X) 41
PUTDCM(S$) 42

READ 33
READ# 33
Record Mode command A-24
REMARK 33
REMOVE 9
RENUM 9
REPort Status command A-27
RESTORE 33
RESTORE# 34
RESume command A-27
RETURN 35
REwind command A-28
RND 39
RPLOT 85
RPT$(S$,X) 40
RUN 11

SAVE 11
SCALE 62
SCRATCH 11
SET 12
SET time/date command A-28
SETAR 57
SETGU/SETUU 65
SGN(X) 39
SHORT 35
SHORT(X) 39
SHOW 63
SHow command A-29
SIN(X) 39
SKip command A-30
SLEEP 36
Soft clip limits 50
SPA(N) 41

SQR(X) 39
STOP 36
String functions 40
SUB 37
SUBEND 37
SUspend command A-31

TAB(N) 41
TAN(X) 39
TELL command A-31
Test command A-32
TRansfer command A-33
Trig functions 39
TRIM$(S$) 40

UDU's 50
Units 50
UPC$(S$) 41
Using Terminal BASIC 2

VAL$(X) 40
VAL(S$) 40
Verify Mode command A-24

WAKEUP 37
WHERE 87

XAXIS 66

YAXIS 68

NOTES

NOTES

NOTES

NOTES

NOTES

Part Number 02647—90006

Printed in U.S.A.