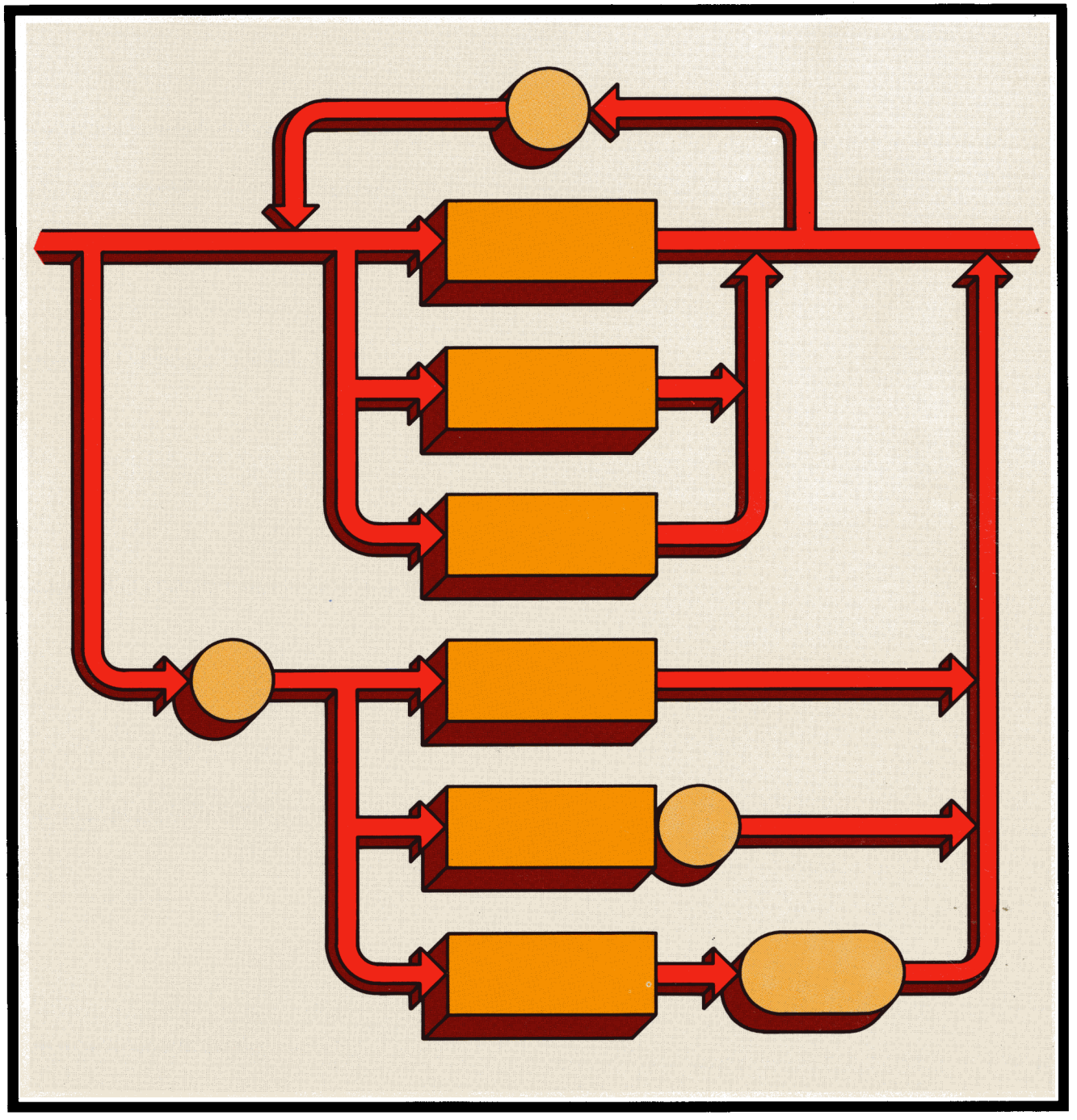
HP 9000 Series 200/300 Computers

HEWLETT PACKARD

BASIC 4.0 Language Reference



BASIC 4.0 Language Reference

for the HP 9000 Series 200/300 Computers

Manual Reorder No. 98613-90051

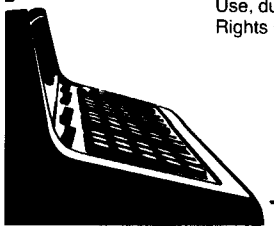
© Copyright 1985, Hewlett-Packard Company.

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 language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).



Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

July 1985...Edition 1

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 MANUAL, 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 direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Table of Contents

Language History 1
Language History 1
Keyboards 7
Using the Keyword Dictionary 7
Legal Usage Table 7
Syntax Drawings Explained 8
Keywords and Spaces 8

BASIC Language Reference for HP Series 200/300 SRM Workstations 473
Syntax for Remote File and Directory Specification 474
Remote File Specifier 474
Directory Path 475
Remote msus 476
Directory Specifier 478
Access Capability Requirements 479
Table of Access Capabilities Required for Keyword Use 480
Using Protected Files Created on a Pascal Workstation 481
Summary of BASIC Keyword Use on SRM 482
ASSIGN 483
CAT 484
CHECKREAD 488
CONTROL 489
COPY 490
CREATE ASCII 491
CREATE BDAT 492
CREATE DIR 493
ENTER 494
GET 495
INITIALIZE 496
LOAD 497
LOADSUB 498
LOCK 499
MASS STORAGE IS (MSI) 500
ON TIMEOUT 501
OUTPUT 502
PLOTTER IS 503
PRINTER IS 504
PROTECT 505
PURGE 507
RENAME 508
RE-SAVE 509
RESET 510
RE-STORE 511
SAVE 512

SCRATCH A ... 513
STATUS ... 514
STORE ... 515
STORE SYSTEM ... 516
SYSTEM$ ... 517
TRANSFER ... 518
UNLOCK ... 519
SRM BASIC Error Codes for HP Series 200/300 Computers ... 520

Glossary ... 521

Interface Registers ... 533
I/O Path Status and Control Registers ... 533
CRT Status and Control Registers ... 535
Keyboard Status and Control Registers ... 538
HP-IB Status and Control Registers ... 541
RS-232 Status and Control Registers ... 546
GPIO Status and Control Registers ... 551
BCD Status and Control Registers ... 553
Data Communications Status and Control Registers ... 556
Powerfail Status and Control Registers ... 565
EPROM Programmer Status and Control Registers ... 567
PARITY, CACHE and FLOAT Status and Control Registers ... 569
Summary of SRM Status Registers ... 570

Useful Tables ... 571
Option Numbers ... 571
Interface Select Codes ... 571
Display-Enhancement Characters ... 572
US ASCII Character Codes ... 573
U.S./European Display Characters ... 575
U.S./European Display Characters ... 577
U.S./European Display Characters ... 579
Katakana Display Characters ... 581
Katakana Display Characters ... 583
Master Reset Table ... 585
Graphic Reset Table ... 588
Interface Reset Table ... 589
Second Byte of Non-ASCII Key Sequences ... 591
Selected High-Precision Metric Conversion Factors ... 592

Error Messages ... 593

Keyword Summary ... 603
Vocabulary ... 609

Manual Comment Sheet Instructions ... 610

Language History

Language History

This manual documents the BASIC 4.0 Language System used on HP 9000 Series 200/300 computers. There are several versions (other than 4.0) of this language in use today. The following table is provided for those users who have more than one BASIC version, or who are upgrading to BASIC 4.0. The table lists each statement available on any version and notes where optional BIN files are needed.

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
ABORT		IO
ABORTIO	AP2.0	TRANS
ABS		
ACS		
ALLOCATE		
ALPHA		GRAPH
AND		
AREA	GRAPH2.1	GRAPHX
ASN		
ASSIGN		
ATN		
AXES		GRAPH
BASE	AP2.0	MAT
BEEP		
BINAND		
BINCMP		
BINEOR		
BINIOR		
BIT		
BREAK	AP2.0	IO
CALL		
CAT		
CHANGE	AP2.0	PDEV
CHECKREAD	AP2.0	MS
CHR$		
CLEAR		IO
CLIP		GRAPH
COM		
CONT		
CONTROL		
COPY		
COPYLINES	AP2.0	PDEV
COS		
CREATE ASCII		
CREATE BDAT		
CREATE DIR	SRM	SRM
CRT	AP2.0	
CSIZE		GRAPH

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
DATA		
DATE	AP2.0	CLOCK
DATE$	AP2.0	CLOCK
DEALLOCATE		
DEF FN		
DEG		
DEL		
DELSUB		
DET	AP2.0	MAT
DIGITIZE	GRAPH2.0	GRAPHX
DIM		
DISABLE		
DISABLE INTR		IO
DISP		
DIV		
DOT	AP2.0	MAT
DRAW		GRAPH
DROUND		
DUMP ALPHA		
DUMP GRAPHICS		GRAPH
DUMP DEVICE IS		GRAPH
DVAL	AP2.0	
DVAL$	AP2.0	
EDIT		
EDIT KEY	AP2.0	KBD
ENABLE		
ENABLE INTR		IO
END		
ENTER		
ERRDS	AP2.0	
ERRL		
ERRM$	AP2.0	
ERRN		
EXOR		
EXP		
FIND	AP2.0	PDEV
FN		
FOR...NEXT		
FRACT	AP2.0	
FRAME		GRAPH
GCLEAR		GRAPH
GESCAPE	GRAPH2.1	GRAPHX
GET		
GINIT		GRAPH
GLOAD		GRAPH
GOSUB		
GOTO		
GRAPHICS		GRAPH
GRAPHICS INPUT IS	GRAPH2.0	GRAPHX
GRID		GRAPH
GSTORE		GRAPH

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
IDRAW		GRAPH
IF...THEN		
IMAGE		
IMOVE		GRAPH
INDENT	AP2.0	PDEV
INITIALIZE		
INPUT		
INT		
INTEGER		
IPLOT		GRAPH
IPLOT array		GRAPHX
IVAL	AP2.0	
IVAL$	AP2.0	
KBD	AP2.0	
KBD$		
KNOBX		
KNOBY	n.a.	
LABEL		GRAPH
LDIR		GRAPH
LEN		
LET		
LEXICAL ORDER IS	AP2.0	LEX
LGT		
LINE TYPE		GRAPH
LINPUT		
LIST		
LIST BIN	n.a.	
LIST KEY	AP2.0	KBD
LOAD		
LOAD BIN		
LOAD KEY	AP2.0	KBD
LOADSUB		
LOCAL		IO
LOCAL LOCKOUT		IO
LOCK	SRM	SRM
LOG		
LOOP		
LORG		GRAPH
LWC$	AP2.0	
MASS STORAGE IS		
MAT	AP2.0	MAT
MAT REORDER	AP2.0	MAT
MAT SORT	AP2.0	MAT
MAX	AP2.0	MAT
MAXREAL	n.a.	
MIN	AP2.0	MAT
MINREAL	n.a.	
MOD		
MODULO	n.a.	
MOVE		GRAPH
MOVELINES	AP2.0	PDEV

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
NOT		
NPAR		
NUM		
ON/OFF CYCLE	AP2.0	CLOCK
ON/OFF DELAY	AP2.0	CLOCK
ON/OFF END		
ON/OFF EOR	AP2.0	TRANS
ON/OFF EOT	AP2.0	TRANS
ON/OFF ERROR		
ON/OFF INTR		IO
ON/OFF KBD		
ON/OFF KEY		
ON/OFF KNOB		
ON/OFF SIGNAL	AP2.0	IO
ON/OFF TIME	AP2.0	CLOCK
ON/OFF TIMEOUT		
ON		
OPTION BASE		
OR		
OUTPUT		
PASS CONTROL	AP2.0	IO
PAUSE		
PEN		GRAPH
PENUP		GRAPH
PDIR	n.a.	GRAPH
PI		
PIVOT		GRAPH
PLOT		GRAPH
PLOT array	GRAPH2.1	GRAPHX
PLOTTER IS		GRAPH
PLOTTER IS file	n.a.	GRAPH
POLYGON	GRAPH2.1	GRAPHX
POLYLINE	GRAPH2.1	GRAPHX
POS		
PPOLL		IO
PPOLL CONFIGURE		IO
PPOLL RESPONSE	AP2.0	IO
PPOLL UNCONFIGURE		IO
PRINT		
PRINT LABEL	n.a.	MS
PRINTALL IS		
PRINTER IS		
PRINTER IS file	n.a.	
PROTECT		
PROUND	AP2.0	
PRT	AP2.0	
PURGE		

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
RAD		
RANDOMIZE		
RANK	AP2.0	MAT
RATIO		GRAPH
READ		
READIO		
READ LABEL	n.a.	MS
READ LOCATOR	GRAPH2.0	GRAPHX
REAL		
RECTANGLE	GRAPH2.1	GRAPHX
REDIM	AP2.0	MAT
REM		
REMOTE		IO
REN		
RENAME		
REPEAT...UNTIL		
REQUEST	AP2.0	IO
RE-SAVE		
RES	n.a.	
RESET	AP2.0	IO
RESTORE		
RE-STORE		
RE-STORE BIN		n.a.
RE-STORE KEY	AP2.0	KBD
RESUME INTERACTIVE		
RETURN		
REV$	AP2.0	
RND		
ROTATE		
RPLOT	GRAPH	GRAPH
RPLOT array	AP2.0	GRAPHX
RPT$		
RUN		
SAVE		
SC	AP2.0	
SCRATCH		
SCRATCH BIN	n.a.	
SCRATCH KEY	AP2.0	KBD
SECURE	n.a.	PDEV
SELECT...CASE		
SEND		IO
SET ECHO	GRAPH2.0	GRAPHX
SET LOCATOR	n.a.	GRAPHX
SET PEN	GRAPH2.1	GRAPHX
SET TIME		
SET TIMEDATE		
SGN		
SHIFT		

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0
SHOW		GRAPH
SIGNAL	AP2.0	IO
SIN		
SIZE	AP2.0	MAT
SPOLL		IO
SQR		
STATUS		
STOP		
STORE		
STORE BIN		n.a.
STORE KEY	AP2.0	KBD
STORE SYSTEM	n.a.	
SUB		
SUBEXIT		
SUM	AP2.0	MAT
SUSPEND INTERACTIVE		
SYMBOL	GRAPH2.1	GRAPHX
SYSBOOT	n.a.	
SYSTEM PRIORITY	AP2.0	
SYSTEM$	AP2.0	
SYSTEM$ PLOTTER IS	GRAPH2.0	GRAPH
SYSTEM$ GRAPHICS INPUT IS	GRAPH2.0	GRAPH
SYSTEM$ LEXICAL ORDER IS	AP2.0	LEX
SYSTEM$ KEYBOARD LANGUAGE	AP2.0	LEX
TAN		
TIME	AP2.0	CLOCK
TIME$	AP2.0	CLOCK
TIMEDATE		
TRACE ALL		PDEV
TRACE OFF		PDEV
TRACE PAUSE		PDEV
TRACK	GRAPH2.0	GRAPHX
TRANSFER	AP2.0	TRANS
TRIGGER		IO
TRIM$	AP2.0	
UNLOCK	SRM	SRM
UPC$	AP2.0	
VAL		
VAL$		
VIEWPORT		GRAPH
WAIT		
WAIT FOR EOR	AP2.0	TRANS
WAIT FOR EOT	AP2.0	TRANS
WHERE	GRAPH2.1	GRAPHX
WHILE		
WINDOW		GRAPH
WRITEIO		
XREF	AP2.0	XREF

Keyboards

The Series 200/300 Computers support three keyboard styles:

- HP 98203B
- HP 98203A
- HP 46020A

Throughout the manuals which document the BASIC Language System, specific keys are mentioned. Because many key labels are different on each keyboard, you will not have all the keys mentioned. For example, [EXECUTE] and [RETURN] normally have the same meaning, but only one of them appears on any one keyboard. The *BASIC User's Guide* discusses the keyboards.

Within this manual, the keys for each keyboard are listed the first time they are used in a statement description. Thereafter, only one keyboard's keys are used.

Using the Keyword Dictionary

This section contains an alphabetical reference to all the keywords currently available with the BASIC language system of the Series 200/300 computers. Each entry defines the keyword, shows the proper syntax for its use, gives some example statements, and explains relevant semantic details. A cross reference is provided in the back that groups the keywords into several functional categories.

Legal Usage Table

Above each drawing is a small table indicating the legal uses of the keyword. Option Required indicates what must be resident in the computer (other than BASIC 4.0) in order to use the keyword. Specific headings under Semantics may list a requirement for the specific feature being discussed if the keyword has expanded semantics with binary extensions. Shaded areas of the syntax diagram flag syntactic changes which depend upon the binary extensions to the language.

"Keyboard Executable" means that a properly constructed statement containing that keyword can be typed into the keyboard input line and executed by a press of the [EXECUTE], [ENTER], or [RETURN] key. "Programmable" means that a properly constructed statement containing that keyword can be placed after a line number and stored in a program. Certain non-programmable keywords can be "forced" into a program by sending them to the keyboard buffer with an OUTPUT 2 statement. This is **not** what is meant by "Programmable".

"In an IF...THEN..." means that a properly constructed statement containing that keyword can be placed after "THEN" in a **single-line** IF...THEN statement. Keywords that are prohibited in a single-line IF...THEN are not necessarily prohibited in a multiple-line IF...THEN structure. Constructs such as IF...THEN, REPEAT...UNTIL, and FOR...NEXT statements are executed conditionally when they are included in a multiple-line IF...THEN structure. All other prohibited statements (see IF...THEN) are used only during pre-run. Therefore, the action of those statements will not be conditional, even though the IF...THEN wording may make them appear to be conditional.

Syntax Drawings Explained

Statement syntax is represented pictorially. All characters enclosed by a rounded envelope must be entered exactly as shown. Words enclosed by a rectangular box are names of items used in the statement. A description of each item is given either in the table following the drawing, another drawing, or the Glossary. Statement elements are connected by lines. Each line can be followed in only one direction, as indicated by the arrow at the end of the line. Any combination of statement elements that can be generated by following the lines in the proper direction is syntactically correct. An element is optional if there is a path around it. Optional items usually have default values. The table or text following the drawing specifies the default value that is used when an optional item is not included in a statement.

Comments may be added to any valid line. A comment is created by placing an exclamation point after a statement, or after a line number or line label. The text following the exclamation point may contain any characters in any order.

The drawings do not necessarily deal with the proper use of spaces (ASCII blanks). In general, whenever you are traversing a line, any number of spaces may be entered. If two envelopes are touching, it indicates that no spaces are allowed between the two items. However, this convention is not always possible in drawings with optional paths, so it is important to understand the following rules for spacing.

Keywords and Spaces

The computer uses spaces, as well as required punctuation, to distinguish the boundaries between various keywords, names, and other items. In general, at least one space is required between a keyword and a name if they are not separated by other punctuation. Spaces cannot be placed in the middle of keywords or other reserved groupings of symbols. Also, keywords are recognized whether they are typed in uppercase or lowercase. Therefore, to use the letters of a keyword as a name, the name entered must contain some mixture of uppercase and lowercase letters. The following are some examples of these guidelines.

Space Between Keywords and Names

The keyword `NEXT` and the variable `Count` are properly entered with a space between them, as in `NEXT Count`. Without the space, the entire group of characters is interpreted as the name `Nextcount`.

No Spaces in Keywords or Reserved Groupings

The keyword `DELSUB` cannot be entered as `DEL SUB`. The array specifier `(*)` cannot be entered as `( * )`. A function call to "A$" must be entered as `FNA$`, not as `FN A $`. The I/O path name "@Meter" must be entered as `@Meter`, not as `@ Meter`. The "exceptions" are keywords that contain spaces, such as `END IF` and `OPTION BASE`.

Using Keyword Letters for a Name

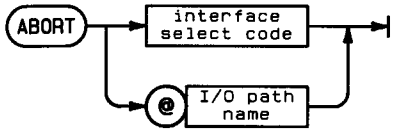
Attempting to store the line `IF X=1 THEN END` will generate an error because `END` is a keyword not allowed in an IF...THEN. To create a line label called "End", type `IF X=1 THEN ENd`. This or any other mixture of uppercase and lowercase will prevent the name from being recognized as a keyword.

Also note that names may begin with the letters of an infix operator (such as MOD, DIV, and EXOR). In such cases, you should type the name with a case switch in the infix operator portion of the name (e.g., MOdULE, DiVISOR).

ABORT

Option Required	IO
Keyboard Executable	Yes
Programmagle	Yes
In an IF...THEN...	Yes

This statement ceases activity on the specified interface.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	5 , 7 thru 31
I/O path name	name assigned to an HP-IB interface select code	any valid name (see ASSIGN)

Example Statements

```
ABORT 7
IF Stop_code THEN ABORT @Source
```

Semantics

HP-IB Interfaces

Executing this statement ceases activity on the specified HP-IB interface; other interfaces may not be specified. If the computer is the system controller but not currently the active controller, executing ABORT causes the computer to assume active control.

Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	IFC (duration ≥100μsec) REN $\overline{\text{ATN}}$	Error	ATN MTA UNL $\overline{\text{ATN}}$	Error
Not Active Controller	IFC (duration ≥100 μsec)* REN $\overline{\text{ATN}}$		No Action	

* The IFC message allows a non-active controller (which is the system controller) to become the active controller.

Data Communications Interfaces

Directing this statement to a Data Communications interface clears the buffers and disconnects the interface.

ABORTIO

Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement terminates a TRANSFER which is currently taking place through an I/O path assigned to a device, group of devices, or mass storage file.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name

Example Statements

```
ABORTIO @Interface
IF Stop_flag THEN ABORTIO @Device
```

Semantics

This statement terminates a TRANSFER (in either direction) currently taking place through the specified I/O path name. The I/O path name must be assigned to an interface select code, device selector, or mass storage file; if the I/O path name is assigned to a buffer, error 170 is reported.

An end-of-transfer (EOT) branch is initiated if an ON EOT branch is currently defined for the I/O path name; however, no currently defined EOR branch will be initiated.

The ABORTIO has no effect if no TRANSFER is taking place through the I/O path name.

If a TRANSFER to or from an I/O path name was terminated by an error, executing ABORTIO on that I/O path name causes the error to be reported.

ABS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the absolute value of its argument. The result will be of the same type (REAL or INTEGER) as the argument. (Except for the ABS of the INTEGER −32 768, which causes an error).



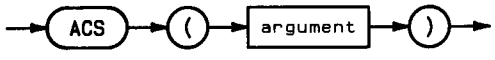
Example Statements

```
Magnitude=ABS(Vector)
PRINT "Value =";ABS(X1)
```

ACS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a cosine equal to the argument. This is the arccosine function.



Item	Description/Default	Range Restrictions
argument	numeric expression	−1 thru +1

Example Statements

```
Angle=ACS(Cosine)
PRINT "Angle =";ACS(X1)
```

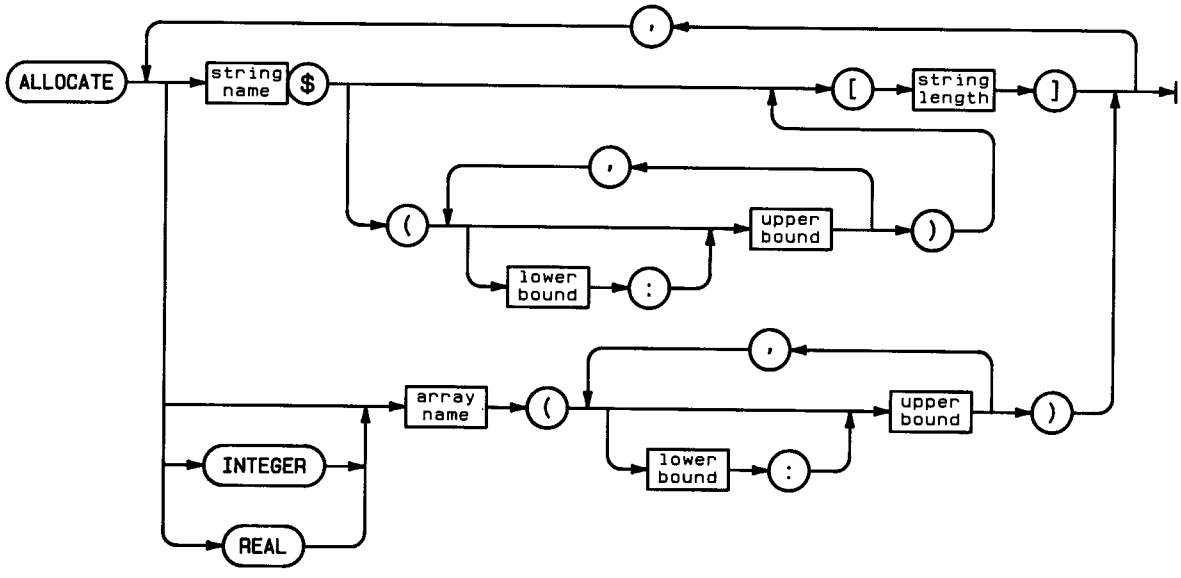
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is 0 thru 180 degrees. If the current angle mode is RAD, the range of the result is 0 thru π radians. The angle mode is radians unless you specify degrees with the DEG statement.

ALLOCATE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement dynamically allocates memory for arrays and string variables during program execution.



Item	Description/Default	Range Restrictions
array name	name of a numeric array	any valid name
lower bound	numeric expression, rounded to an integer; Default = OPTION BASE value (0 or 1)	−32 768 thru +32 767 (see "array" in Glossary)
upper bound	numeric expression, rounded to an integer	−32 768 thru +32 767 (see "array" in Glossary)
string name	name of a string variable	any valid name
string length	numeric expression, rounded to an integer	1 thru 32 767

Example Statements

```
ALLOCATE Temp(Low:High)
ALLOCATE R$[LEN(A$)+1]
```

Semantics

Memory reserved by the ALLOCATE statement can be freed by the DEALLOCATE statement. However, because of the stack discipline used when allocating, the freed memory space does not become available unless all subsequently allocated items are also deallocated. For example, assume that A$ is allocated first, then B$, and finally C$. If a DEALLOCATE A$ statement is executed, the memory space for A$ is not reclaimed until B$ and C$ are deallocated. This same stack is used for setting up ON-event branches, so subsequent ON-event statements can also block the reclamation of deallocated memory.

The variables in an ALLOCATE statement cannot have appeared in COM, DIM, INTEGER or REAL declaration statements. If variable(s) are to be allocated in a subprogram, the variable(s) cannot have been included in the subprogram's formal parameter list. Implicitly declared variables cannot be allocated. Numeric variables which are not specified as INTEGER are assumed to be REAL. A variable can be re-allocated in its program context only if it has been deallocated and its type and number of dimensions remain the same.

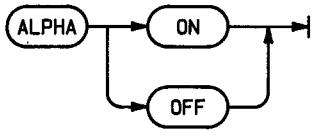
Exiting a subprogram automatically deallocates any memory space allocated within that program context.

ALLOCATE can be executed from the keyboard while a program is running or paused. However, the variable must have been declared in an ALLOCATE statement in the current program context, and the variable must have already been allocated and deallocated.

ALPHA

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement turns the alphanumeric display on or off.



Example Statements

```
ALPHA ON
IF Graph THEN ALPHA OFF
```

Semantics

Items sent to the printout area while the alphanumeric display is disabled are placed in the display memory even though they are not visible. Items sent to the keyboard input line, the display line, or the system message line will turn on the alphanumeric display. The alphanumeric and graphic displays can both be on at the same time.

The alphanumeric area is enabled after power-on, RESET and SCRATCH A. Pressing the ALPHA key on the keyboard also enables the alphanumeric display.

This statement has no effect on a bit-mapped display when the alpha write-enable mask specifies all planes. This is the default state on those displays.

AND

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based upon the logical AND of the arguments.



Example Statements

```
IF Flag AND Test2 THEN Process
Final=Initial AND Valid
```

Semantics

A non-zero value (positive or negative) is treated as a logical 1; only zero is treated as a logical 0.

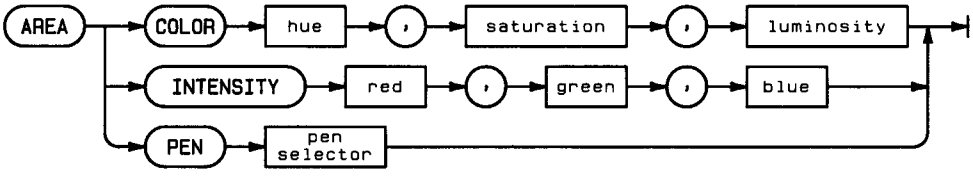
The logical AND is shown in this table:

A B	A AND B
0 0	0
0 1	0
1 0	0
1 1	1

AREA

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines or selects an area fill color. The fill color is used in all subsequent graphics operations requiring area fill.



Item	Description/Default	Range Restrictions
hue	numeric expression	0 thru 1
saturation	numeric expression	0 thru 1
luminosity	numeric expression	0 thru 1
red	numeric expression	0 thru 1
green	numeric expression	0 thru 1
blue	numeric expression	0 thru 1
pen selector	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
AREA COLOR Hue,Saturation,Luminosity
AREA COLOR X*.3,RND,A^2
AREA INTENSITY Red(I),Green(I),Blue(I)
AREA INTENSITY X*.3,RND,A^2
AREA PEN 1
AREA PEN -Pen
```

Semantics

The default fill color is the color specified by Pen 1. This color is solid white after power-up, SCRATCH A, or GINIT.

A fill color remains in effect until the execution of an AREA, GINIT, or SCRATCH A. Other statements which may alter the current fill color (depending on the data passed to them) are SYMBOL, PLOT, RPLOT, or IPLOT when used with an array. SET PEN affects pen colors, and therefore can also affect fill colors specified with AREA statements.

Specifying color with the SET PEN and AREA PEN statements (resulting in non-dithered color) results in a much more accurate representation of the desired color than the same color requested with an AREA COLOR or AREA INTENSITY statement. To see the difference, compare the five color plates shown in this entry with the corresponding plates in the SET PEN statement.

Note

The following color plates do not exactly represent what your eye would see on the CRT. The reason for this is that photographic film cannot capture all the colors a CRT can produce, and the printing process cannot reproduce all the colors that film can capture.

AREA PEN

A fill color specified with AREA PEN is guaranteed to be non-dithered, and the AREA PEN statement executes much faster than AREA COLOR or AREA INTENSITY.

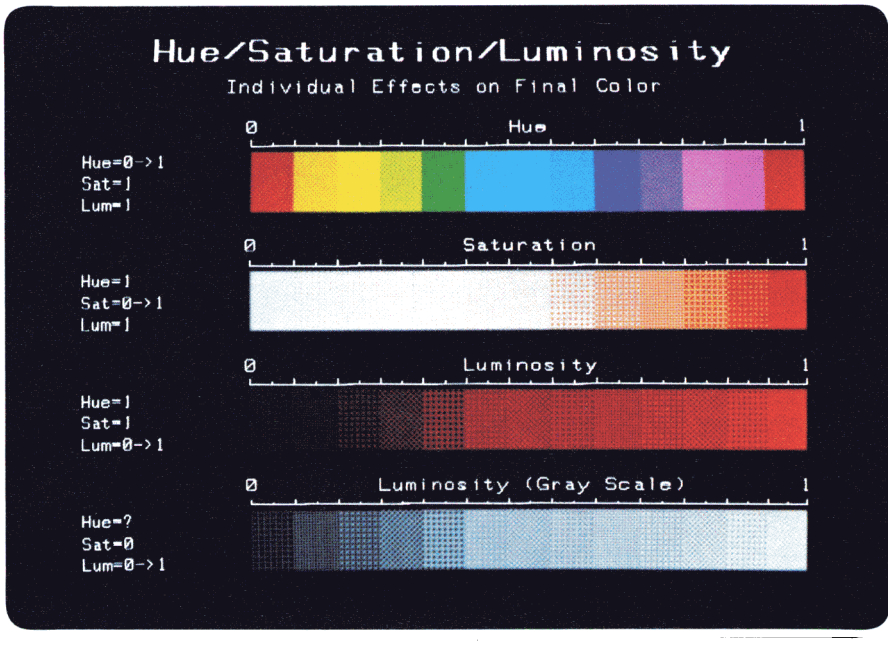
The pen numbers have the same effect as described in the PEN statement for line color except that in the alternate pen mode, negative pens erase as in the normal pen mode; they do not complement. Pen 0 in normal pen mode erases; it does not complement.

AREA COLOR

When AREA COLOR is executed on a color monitor, the HSL parameters are converted to RGB values. Then, if the color requested is not available in the color map, the computer creates the closest possible color in RGB color space to the one requested by filling the 4×4 dither cell with the best combination of colors from the color map.

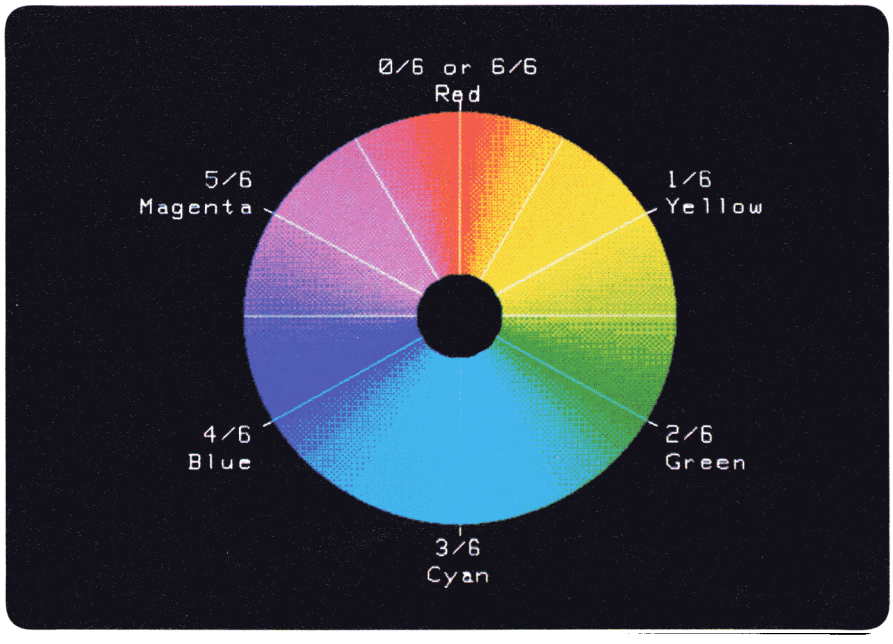
In non-color map mode, there are eight colors total, and they cannot be redefined. This simulates the operation of the HP98627A. In color map mode, there are sixteen or 256 total colors depending on your hardware, and they can be redefined with SET PEN.

The following plate of the screen shows the changes brought about by varying one of the HSL parameters at a time. The bottom bar shows that when saturation (the amount of color) is zero, hue makes no difference, and varying luminosity results in a gray scale.



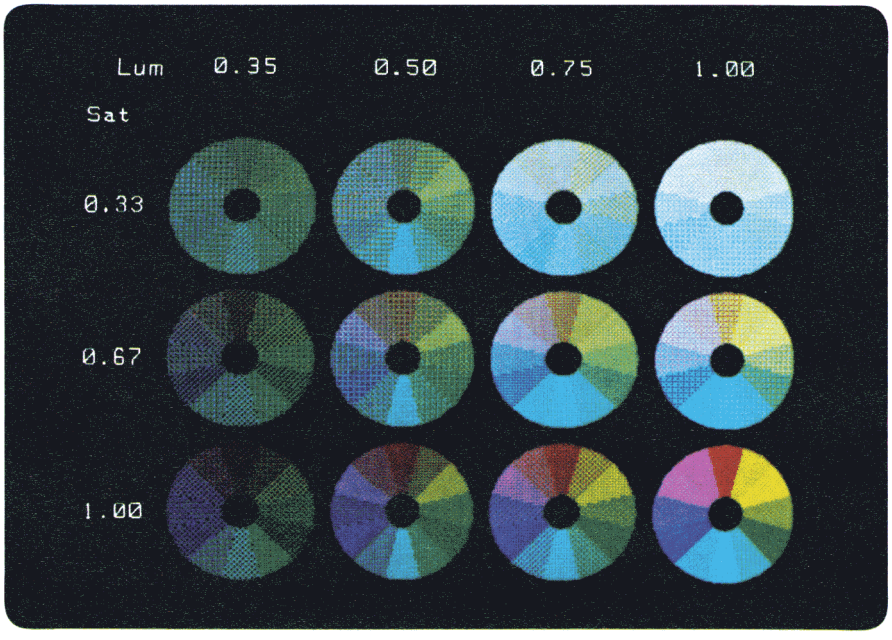
The following color wheel represents the colors selected as the hue value goes from 0 through 1. Any value between zero and one, inclusive, can be chosen to select color. The resolution (the amount the value can change before the color on the screen changes) depends on what the value of the hue is as well as the values of the other two parameters.

HSL Color Wheel



The next plate shows the effect that varying saturation and luminosity have on the color produced. Each of the small color wheels is a miniature version of the large one above, except it has fewer segments.

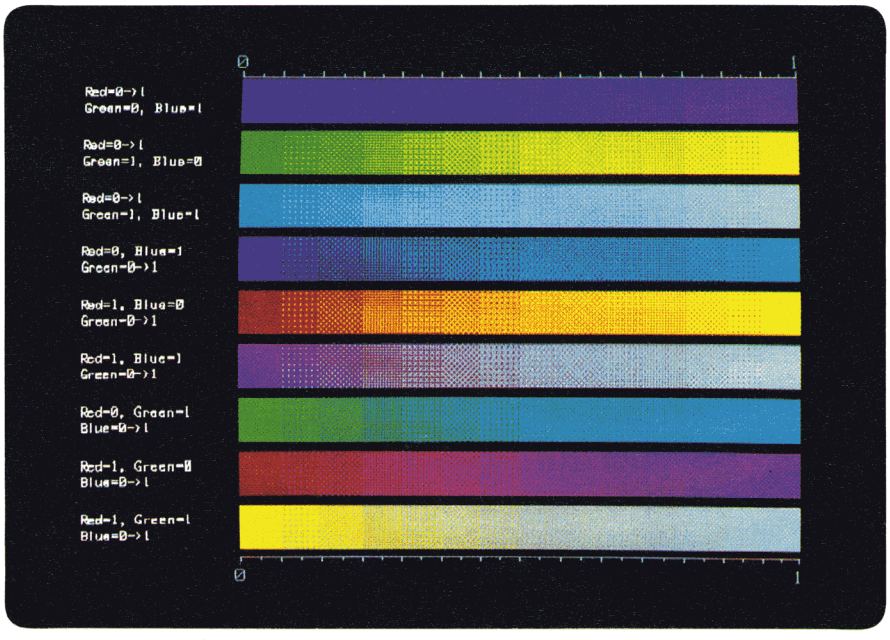
Effects of Saturation and Luminosity on Color



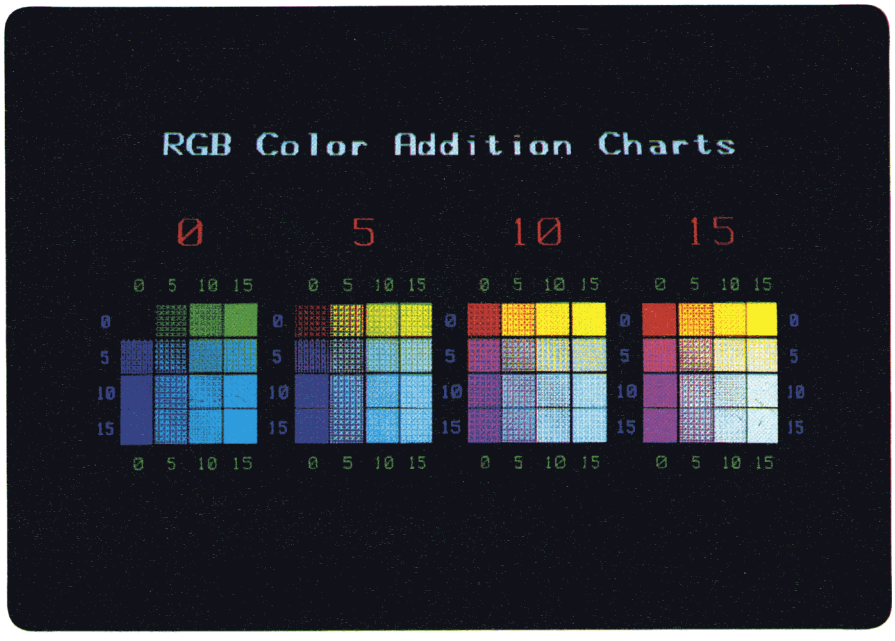
AREA INTENSITY

The following plate demonstrates the effect of varying the intensity of one color component when the other two remain constant.

RGB Addition: One Color at a Time



The next plate shows combinations of red, green and blue. The values are given in fifteenths: 0 fifteenths, 5 fifteenths, 10 fifteenths, and 15 fifteenths—every fifth value. The values for each color component are represented in that color.



The HP98627A

When an HP98627A is used, the HSL values specified in an AREA COLOR statement are converted to RGB. The parameters of an AREA INTENSITY statement are already in RGB. The RGB values specify the fraction of dots per 4 × 4-pixel area to be turned on in each memory plane. The red value corresponds to memory plane 1, the green value to memory plane 2, and the blue value to memory plane 3.

The AREA PEN selects one of the eight non-dithered colors available with no intensity control on the color guns. See the PEN entry for the order of these colors.

The HP98627A dithers in a very similar way to the color monitors when the color map is not enabled (see PLOTTER IS), using only eight colors when calculating the closest combination.

Monochromatic CRTs

When doing shading on a monochromatic CRT, dithering is always used. Dithering takes place in a 4 × 4 cell, which allows zero through sixteen of the dots to be turned on, for a total of seventeen shades of gray.

Since AREA PEN does not use dithering, only black and white are available. If the pen selector is positive, the resulting fill color is white; if zero or negative, the resulting fill color is black.

When an AREA COLOR is executed, the hue and saturation parameters are ignored. Only the luminosity value is used to determine the fraction of pixels to be turned on.

When an AREA INTENSITY is executed, the largest of the three values is used, and it specifies the fraction of pixels to be turned on.

Alternate Pen Mode Fills
If the alternate drawing mode is in effect when the fill is performed, the area will be filled with non-dominant color. See GESCAPE operation selectors 4 and 5.

In the alternate pen mode, negative pens erase as in the normal pen mode; they do not complement.

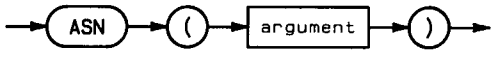
ASCII

See the CREATE ASCII and LEXICAL ORDER IS statements.

ASN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a sine equal to the argument. This is the arcsine function.



Item	Description/Default	Range Restrictions
argument	numeric expression	−1 thru +1

Example Statements

```
Angle=ASN(Sine)
PRINT "Angle =";ASN(X1)
```

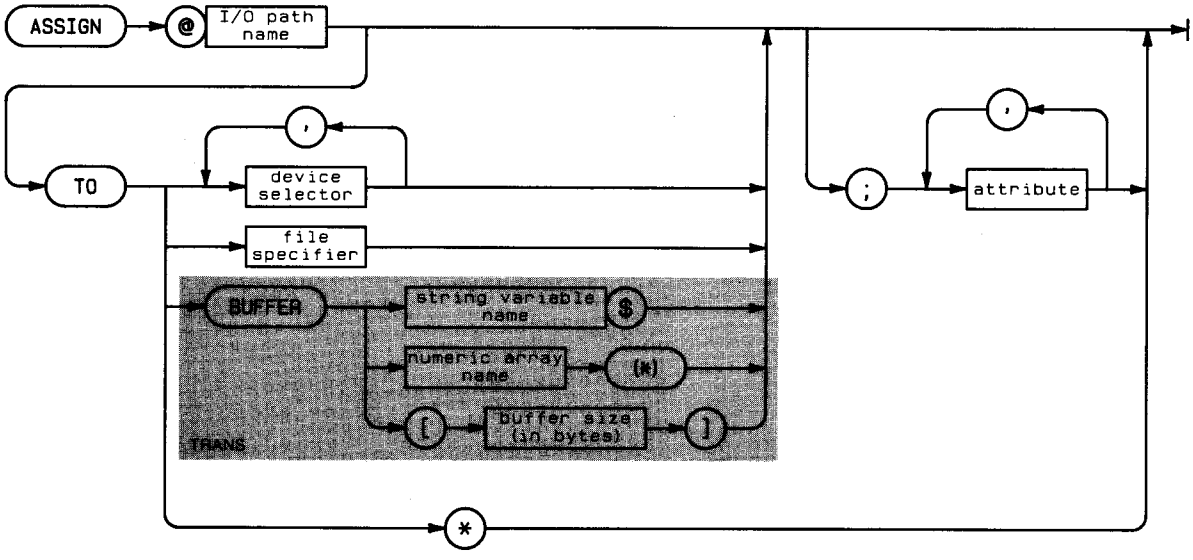
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is −90 thru +90 degrees. If the current angle mode is RAD, the range of the result is $-\pi/2$ thru $+\pi/2$ radians. The angle mode is radians unless you specify degrees with the DEG statement.

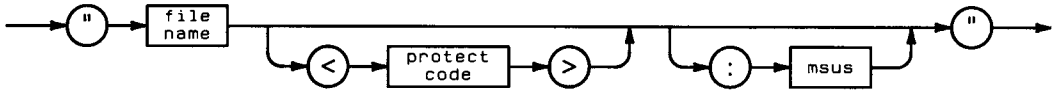
ASSIGN

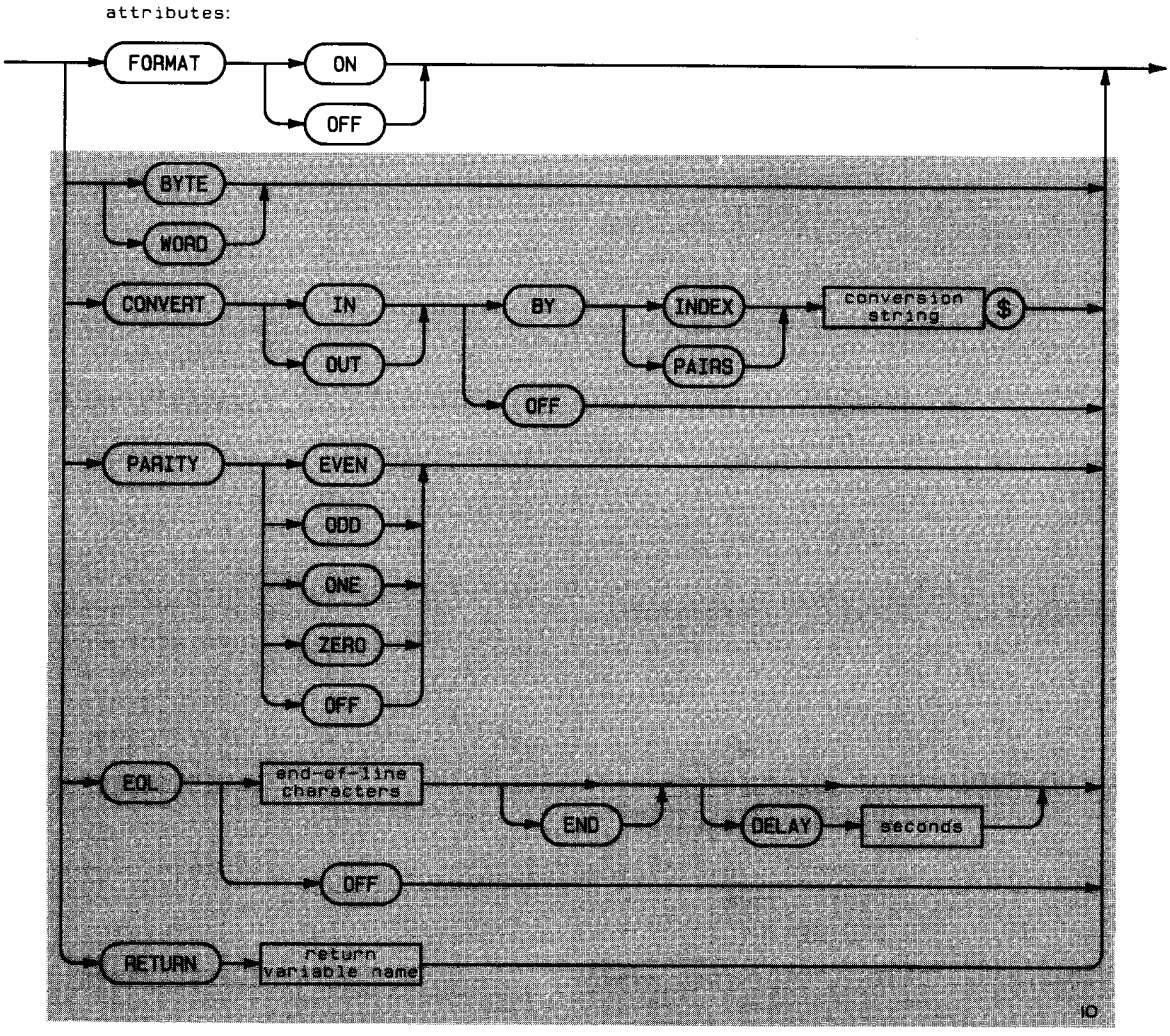
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to perform one of the following actions: assign an I/O path name and attributes to a device, a group of devices, a mass storage file, or a buffer; change attributes; or close an I/O path name. (If using ASSIGN with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:




attributes:
FORMAT
ON
OFF
BYTE
WORD
CONVERT
IN
OUT
BY
INDEX
PAIRS
conversion string
$
OFF
PARITY
EVEN
ODD
ONE
ZERO
OFF
EOL
end-of-line characters
END
DELAY
seconds
OFF
RETURN
return variable name
10

Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path	any valid name
device selector	numeric expression	(see Glossary)
file specifier	string expression	(see drawing)
string variable name	name of a string variable	any valid name
numeric array name	name of a numeric array	any valid name
buffer size (in bytes)	numeric expression, rounded to an integer	1 thru available memory minus 490
file name	literal	any valid file name
protect code	literal, first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)
conversion string	name of a string variable	up to 256 characters (with INDEX); even number of characters (with PAIRS)
end-of-line characters	string expression; Default = CR and LF	up to 8 characters
time period	numeric expression, rounded to the nearest 0.001 seconds; Default = 0	0.001 thru 32.767
return variable name	name of a numeric variable	any valid name

Example Statements

```
ASSIGN @File TO Name$&Msus$
ASSIGN @Source TO Isc;FORMAT OFF
ASSIGN @Source;FORMAT ON
ASSIGN @Device TO 724
ASSIGN @Listeners TO 711,712,715
ASSIGN @Dest TO *

ASSIGN @Buf_1 TO BUFFER String_variable$
ASSIGN @Buf_2 TO BUFFER Numeric_array(*)
ASSIGN @Buf_3 TO BUFFER [128]

ASSIGN @Resource TO Gpio;WORD,CONVERT IN BY INDEX In$
ASSIGN @Resource;CONVERT OUT BY INDEX Out$
ASSIGN @Resource TO Hpib;EOL Eol$ END DELAY .05
ASSIGN @Resource TO Rs_232;PARITY ODD
```

Semantics

The ASSIGN statement has three primary purposes. Its main purpose is to create an I/O path name and assign that name to an I/O resource and attributes that describe the use of that resource. The statement is also used to change the attributes of an existing I/O path and to close an I/O path.

Associated with an I/O path name is a unique data type that uses about 200 bytes of memory. I/O path names can be placed in COM statements and can be passed by reference as parameters to subprograms. They cannot be evaluated in a numeric or string expression and cannot be passed by value.

Once an I/O path name has been assigned to a resource, OUTPUT, ENTER, TRANSFER, STATUS, and CONTROL operations can be directed to that I/O path name. This provides the convenience of re-directing I/O operations in a program by simply changing the appropriate ASSIGN statement. The resource assigned to the I/O path name may be an interface, a device, a group of devices on HP-IB, a mass storage file or a buffer. Note that the Status and Control registers of an I/O path are different from the Status and Control registers of an interface. All Status and Control registers are summarized in the "Interface Registers" section at the back of the book.

The FORMAT Attributes

Assigning the FORMAT ON attribute to an I/O path name directs the computer to use its ASCII data representation while sending and receiving data through the I/O path. Assigning the FORMAT OFF attribute to an I/O path name directs the computer to use its internal data representation when using the I/O path.

LIF ASCII format (similar to ASCII representation) is always used with ASCII files; thus, if either FORMAT ON or FORMAT OFF is specified for the I/O path name of an ASCII file, it will be ignored.

If a FORMAT attribute is not explicitly given to an I/O path, a default is assigned. The following table shows the default FORMAT attribute assigned to computer resources.

Resource	Default Attributes
interface/device	FORMAT ON
ASCII file	(always ASCII format)
BDAT file	FORMAT OFF
buffer	FORMAT ON

The FORMAT OFF attribute cannot be assigned to an I/O path which currently possesses any non-default CONVERT or PARITY attribute(s), and vice versa.

Using Devices

I/O path names are assigned to devices by placing the device selector after the keyword TO. For example, `ASSIGN @Display TO 1` creates the I/O path name "@Display" and assigns it to the internal CRT. The statement `ASSIGN @Meters TO 710,711,712` creates the I/O path name "@Meters" and assigns it to a group of three devices on HP-IB. When multiple devices are specified, they must be on the same interface.

When an I/O path name which specifies multiple devices is used in an OUTPUT statement, all devices referred to by the I/O path name receive the data. When an I/O path name which specifies multiple devices is used in an ENTER statement, the first device specified sends the data to the computer and to the rest of the devices. When an I/O path name which specifies multiple HP-IB devices is used in either CLEAR, LOCAL, PPOLL CONFIGURE, PPOLL UNCONFIGURE, REMOTE, or TRIGGER statement, all devices associated with the I/O path name receive the HP-IB message.

A device can have more than one I/O path name associated with it. Each I/O path name can have different attributes, depending upon how the device is used. The specific I/O path name used for an I/O operation determines which set of attributes is used for that operation.

Using Files

Assigning an I/O path name to a file name associates the I/O path with a file on the mass storage media. The mass storage file must be a data file, either ASCII or BDAT. The file must already exist on the media, as ASSIGN does not do an implied CREATE.

ASCII and BDAT files have a position pointer which is associated with each I/O path name. The position pointer identifies the next byte to be written or read, and the value of the position pointer is updated with each ENTER or OUTPUT that uses that I/O path name. The position pointer is reset to the beginning of the file when the file is opened. A file is opened by any ASSIGN statement that includes the file specifier. It is best if a file is open with only one I/O path name at a time.

BDAT files have an additional pointer for end-of-file. The end-of-file value from the media is read when the file is opened. The end-of-file pointer is updated on the media at the following times:

- When the current end-of-file changes.
- When END is specified in an OUTPUT statement directed to the file.
- When a CONTROL statement directed to the I/O path name changes the position of the end-of-file pointer.

Using Buffers (Requires TRANS)

The ASSIGN statement is also used to create a buffer (called an "unnamed" buffer) and assign an I/O path name to it or to assign an I/O path name to a buffer (called a "named" buffer) which has been previously declared in a COM, DIM, INTEGER, or REAL declaration statement. Once assigned an I/O path name, a buffer may be the source or destination of a TRANSFER, the destination of an OUTPUT, or the source of an ENTER statement.

I/O path names assigned to buffers contain information describing the buffer, such as buffer capacity, current number of bytes, and empty and fill pointers. This information can be read from STATUS registers of the I/O path name; some of this information may be modified by writing to CONTROL registers. See the "Interface Registers" tabbed section for I/O path register definitions.

The ASSIGN statement that assigns the I/O path name to a named buffer (or creates an unnamed buffer) sets these registers to their initial values: the buffer type is set to either 1 (named buffer) or 2 (unnamed buffer); the empty and fill pointers are set to 1; the current-number-of-bytes register and all other registers are set to 0.

Named buffers can also be accessed through their variable names in the same manner that other variables of that data type can be accessed. However, with this type of access, the buffer registers are not updated; only the data in the buffer changes. For example, using LET to place characters in a named string-variable buffer does not change the empty and fill pointers or the current-number-of-bytes register; only the buffer contents and string's current length can be changed. It is highly recommended that the string's current length (set to the string's dimensioned length by ASSIGN) not be changed in this manner. Unnamed buffers can be accessed only through their I/O path names.

Using ENTER, OUTPUT, or TRANSFER to access a named buffer through its I/O path name updates the appropriate buffer registers automatically; this is unlike accessing a named buffer through its declared variable name (as above).

An I/O path name cannot be assigned to a buffer which will not exist for as long as the I/O path name; this "lifetime" requirement has several implications. Buffers cannot be declared in ALLOCATE statements. If a buffer's I/O path name is to appear in a COM block, the buffer must appear in the same COM block; thus, I/O path names assigned to unnamed buffers cannot appear in COM. If a buffer's I/O path name is to be used as a formal parameter of a subprogram, the buffer to which it will be assigned must appear in the same formal parameter list or appear in a COM which is accessible to that subprogram context. An I/O path name which is a formal parameter to a subprogram cannot be assigned to an unnamed buffer in the subprogram.

Addition Attributes (Requires IO)

The BYTE attribute specifies that all data is to be sent and received as bytes when the I/O path name is used in an ENTER, OUTPUT, PRINT, or TRANSFER statement that accesses a device, file, or buffer and when the I/O path name is specified as the PRINTER IS or PRINTALL IS device. In a TRANSFER, the attribute of BYTE or WORD associated with the non-buffer I/O path name determines how the data is sent.

When neither BYTE nor WORD is specified in any ASSIGN statement for an I/O path, BYTE is the default attribute. Once the BYTE attribute is assigned (either explicitly or by default) to an I/O path name, it cannot be changed to the WORD attribute by using the normal method of changing attributes (see Changing Attributes below); the converse is also true for the WORD attribute.

The WORD attribute specifies that all data is to be sent and received as words (in the same situations as with BYTE above). If the interface to which the I/O path is assigned cannot handle 16-bit data, an error will be reported when the ASSIGN is executed; similarly, if the buffer has a capacity which is an odd number of bytes, an error will be reported. If the FORMAT ON attribute is in effect, the data will be buffered to allow sending words. The first byte is placed in a two-character buffer; when the second byte is placed in this buffer, the two bytes are sent as one word. A Null character, CHR$(0), may be sent to this buffer to force alignment on word boundaries at the following times: before the first byte is sent, before a numeric item is sent with a W image, after an EOL sequence, or after the last byte is sent to the destination. These Nulls may be converted to another character by using the CONVERT attribute (see below). If WORD has been set explicitly, it remains in effect even when the other defaults are restored (see Changing Attributes). The only way to change the WORD attribute is to explicitly close the path name.

The CONVERT attribute is used to specify a character-conversion table to be used during OUTPUT and ENTER operations; OUT specifies conversions are to be made during all OUTPUTs through the I/O path, and IN specifies conversions with all ENTERs. The default attributes are CONVERT IN OFF and CONVERT OUT OFF, which specify that no conversions are to be made in either direction. No non-default CONVERT attribute can be assigned to an I/O path name that currently possesses the FORMAT OFF attribute, and vice versa.

CONVERT...BY INDEX specifies that each original character's code is used to index the replacement character in the specified conversion string, with the only exception that CHR$(0) is replaced by the 256th character in the string. For instance, CHR$(10) is replaced by the 10th character, and CHR$(0) is replaced by the 256th character in the conversion string. If the string contains less than 256 characters, characters with codes that do not index a conversion-string character will not be converted.

CONVERT...BY PAIRS specifies that the conversion string contains pairs of characters, each pair consisting of an original character followed by its replacement character. Before each character is moved through the interface, the original characters in the conversion string (the odd characters) are searched for the character's occurrence. If the character is found, it will be replaced by the succeeding character in the conversion string; if it is not found, no conversion takes place. If duplicate original characters exist in the conversion string, only the first occurrence is used.

The conversion-string variable must exist for as long as the I/O path name (see explanation of the "lifetime" requirement in the preceding section on Using Buffers). Changes made to the value of this variable immediately affect all subsequent conversions which use the variable.

When CONVERT OUT is in effect, the specified conversions are made after any end-of-line (EOL) characters have been inserted into the data but before parity generation is performed (if in effect). When CONVERT IN is in effect, conversions are made after parity is checked but before the data is checked for any item-terminator or statement-terminator characters.

The EOL attribute specifies the end-of-line (EOL) sequence sent after all data during normal OUTPUT operations and when the "L" image specifier is used. Up to eight characters may be specified as the EOL characters; an error is reported if the string contains more than eight characters. The characters are put into the output data before any conversion is performed (if CONVERT is in effect). If END is included in the EOL attribute, an interface-dependent END indication is sent with the last character of the EOL sequence; however, if no EOL sequence is sent, the END indication is also suppressed. If DELAY is included, the computer delays the specified number of seconds (after sending the last character) before continuing. END and DELAY apply only to devices; both are ignored when a file or buffer is the destination. The default EOL sequence consists of sending a carriage-return and a line-feed character with no END indication and no delay period. This default is restored when EOL is OFF.

The PARITY attribute specifies that parity is to be generated for each byte of data sent by OUTPUT and checked for each byte of data received by ENTER. The parity bit is the most significant bit of each byte (bit 7). The default mode is PARITY OFF. No non-default PARITY attribute can be assigned to an I/O path name which currently possesses the FORMAT OFF attribute, and vice versa.

The following PARITY options are available:

Option	Effect on Incoming Data	Effect on Outbound Data
OFF	No check is performed	No parity is generated
EVEN	Check for even parity	Generate even parity
ODD	Check for odd parity	Generate odd parity
ONE	Check for parity bit set (1)	Set parity bit (1)
ZERO	Check for parity bit clear (0)	Clear parity bit (0)

Parity is generated after conversions have been made on outbound data and is checked before conversions on inbound data. After parity is checked on incoming data, the parity bit is cleared; however, when PARITY OFF is in effect, bit 7 is not affected.

If a PARITY attribute is in effect with the WORD attribute, the most-significant bit of each byte of the word is affected.

Determining the Outcome of an ASSIGN (Requires IO)

Although RETURN is not an attribute, including it in the list of attributes directs the system to place a code in a numeric variable to indicate the ASSIGN operation's outcome. If the operation is successful, a 0 is returned. If a non-zero value is returned, it is the error number which otherwise would have been reported. When the latter occurs, the previous status of the I/O path name is retained; the default attributes are not restored. If more than one error occurs during the ASSIGN, the outcome code returned may not be either the first or the last error number.

If RETURN is the only item in an ASSIGN statement, the default attributes are not restored to the I/O path (see Changing Attributes below). For example, executing a statement such as `ASSIGN @Io_path;RETURN Outcome` does not restore the default attributes.

Changing Attributes

The attributes of a currently valid I/O path may be changed, without otherwise disturbing the state of that I/O path or the resource(s) to which it is assigned, by omitting the "TO resource" clause of the ASSIGN statement. For example, `ASSIGN @File;FORMAT OFF` assigns the FORMAT OFF attribute to the I/O path name "@File" without changing the file pointers (if assigned to a mass storage file). The only exception is that once either the BYTE or WORD attribute is assigned to the I/O path name, the attribute cannot be changed in this manner; the I/O path name must either be closed and then assigned to the resource or be re-assigned to change either of these attributes.

A statement such as `ASSIGN @Device` restores the default attributes to the I/O path name, if it is currently assigned. As stated in the preceding paragraph, the only exception is that once the WORD attribute is explicitly assigned to an I/O path name, the default BYTE attribute cannot be restored in this manner.

Closing I/O Paths

There are a number of ways that I/O paths are closed and the I/O path names rendered invalid. Closing an I/O path cancels any ON-event actions for that I/O path. I/O path names that are **not** included in a COM statement are closed at the following times:

- When they are explicitly closed; for example, `ASSIGN @File TO *`
- When a currently assigned I/O path name is re-assigned to a resource, the original I/O path is closed before the new one is opened. The re-assignment can be to the same resource or a different resource. No closing occurs when the ASSIGN statement only changes attributes and does not include the "TO..." clause.
- When an I/O path name is a local variable within a subprogram, it is closed when the subprogram is exited by SUBEND, SUBEXIT, RETURN <expression>, or ON <event> RECOVER.
- When SCRATCH, SCRATCH A, or SCRATCH C is executed, any form of STOP occurs, or an END, LOAD, or GET is executed.

I/O path names that **are** included in a COM statement remain open and valid during a LOAD, GET, STOP, END, or simple SCRATCH. I/O path names in COM are only closed at the following times:

- When they are explicitly closed; for example, `ASSIGN @File TO *`
- When SCRATCH A or SCRATCH C is executed
- When a LOAD, GET, or EDIT operation brings in a program that has a COM statement that does not exactly match the COM statement containing the open I/O path names.

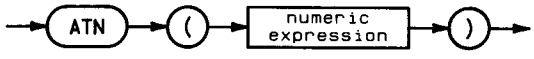
Additionally, when RESET is pressed, all I/O path names are rendered invalid without going through some of the updating steps that are normally taken to close an I/O path. This is usually not a problem, but there are rare situations which might leave file pointers in the wrong state if their I/O path is closed by a RESET. Explicit closing is preferred and recommended.

When ASSIGN is used to close either the source or destination I/O path name of a currently active TRANSFER, the I/O path is not actually closed until the TRANSFER is completed. When I/O path names are closed in this manner, any pending (logged but not serviced) EOR or EOT events are lost (they do not initiate their respective branches). With buffers' I/O path names, the I/O path name might not be closed until two TRANSFERs (one in each direction) are completed.

ATN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a tangent equal to the argument. This is the arctangent function.



Example Statements

```
Angle=ATN(Tangent)
PRINT "Angle =";ATN(X1)
```

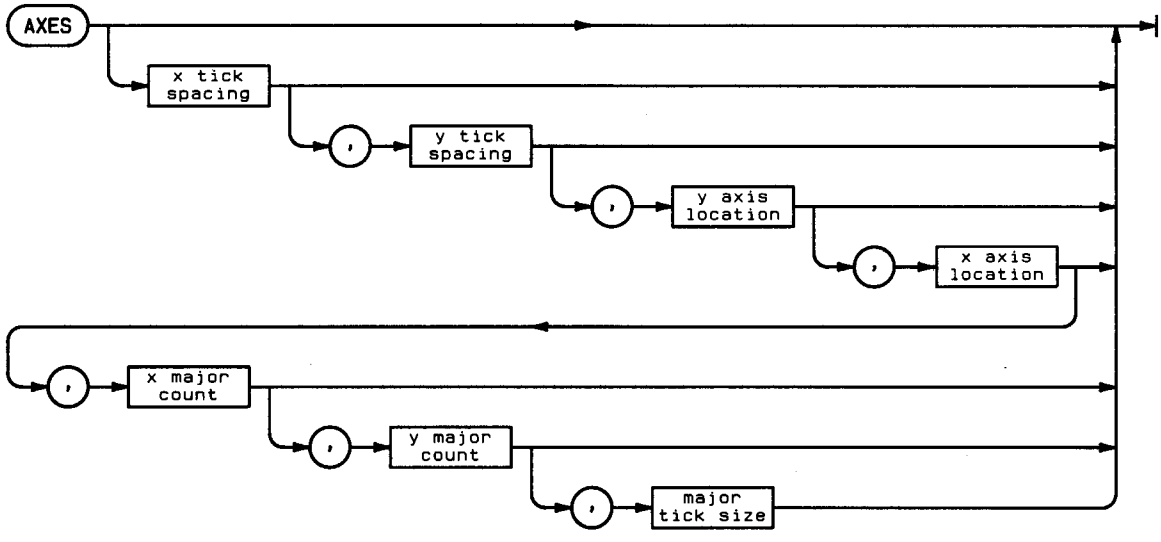
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is -90 thru $+90$ degrees. If the current angle mode is RAD, the range of the result is $-\pi/2$ thru $+\pi/2$ radians. The angle mode is radians unless you specify degrees with the DEG statement.

AXES

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a pair of axes, with optional, equally-spaced tick marks.



Item	Description/Default	Range Restrictions
x tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y axis location	numeric expression specifying the location of the y axis in x-axis units; Default = 0	—
x axis location	numeric expression specifying the location of the x axis in y-axis units; Default = 0	—

Item	Description/Default	Range Restrictions
x major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
y major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
major tick size	numeric expression in graphic display units; Default = 2	—

Example Statements

```
AXES 10,10
AXES X,Y,Midx,Midy,Maxx/10,Maxy/10
```

Semantics

The axes are drawn so they extend across the soft clip area. The tick marks are symmetric about the axes, but are clipped by the soft clip area. Tick marks are positioned so that a major tick mark coincides with the axis origin, whether or not that intersection is visible. Both axes and tick marks are drawn with the current line type and pen. Minor tick marks are drawn half the size of major tick marks.

The X and Y tick spacing must not generate more than 32 768 tick marks in the clip area (including the axis), or error 20 will be generated.

If either axis lies outside the current clip area, that portion of the tick mark which extends into the non-clipped area is drawn.

Applicable Graphics Transformations

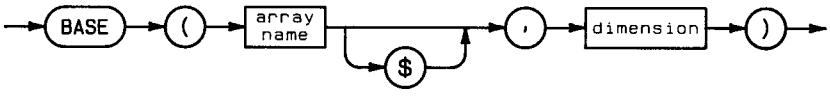
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

BASE

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the lower subscript bound of a dimension of an array. This value is always an INTEGER. (See also OPTION BASE.)



Item	Description/Default	Range Restrictions
array name	name of an array	any valid name
dimension	numeric expression, rounded to an integer	1 thru 6; ≤ the RANK of the array

Example Statements

```
Lowerbound=BASE(Array$,1)
Upperbound(2)=BASE(A,2)+SIZE(A,2)-1
```

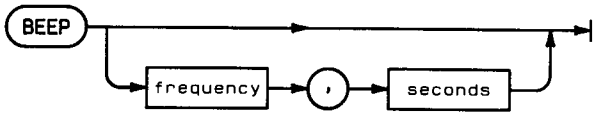
BDAT

See the CREATE BDAT statement.

BEEP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement produces one of 64 audible tones.



Item	Description/Default	Range Restrictions	Recommended Range
frequency	numeric expression, rounded to the nearest tone; Default = 1220.7 Hz	—	81 thru 5208
seconds	numeric expression, rounded to the nearest hundredth; Default = 0.2	—	0.01 thru 2.55

Example Statements

```
BEEP 81.38*Tone,.5
BEEP
```

Semantics

The frequency and duration of the tone are subject to the resolution of the built in tone generator. The frequency specified is rounded to the nearest frequency shown below. For example, any specified frequency from 40.7 to 122.08 produces a beep of 81.38 Hz. If the frequency specified is larger than 5167.63, a tone of 5208.32 is produced. If it is less than 40.69, it is considered to be a 0 and no tone is produced.

The frequency changes when sent to an HP 46020A keyboard. Rounding is performed by the system to produce the number in the first column of the following table. When sent to the HP 46020A keyboard the frequencies change to the corresponding number in the second column.

Standard	HP 46020A	Standard	HP 46020A
81.38	81.45	2685.54	2688.16
162.76	162.12	2766.92	2777.77
244.14	244.37	2848.30	2873.55
325.52	324.25	2929.68	2976.18
406.90	408.49	3011.06	2976.18
488.28	496.03	3092.44	3086.41
569.66	578.70	3173.82	3205.12
651.04	651.03	3255.20	3205.12
732.42	744.04	3336.58	3333.32
813.80	833.33	3417.96	3472.21
895.18	905.79	3499.34	3472.21
976.56	992.06	3580.72	3623.17
1057.94	1096.49	3662.10	3623.17
1139.32	1157.40	3743.48	3787.86
1220.70	1225.49	3824.86	3787.86
1302.08	1302.08	3906.24	3968.24
1383.46	1388.88	3987.62	3968.24
1464.84	1461.98	4069.00	4166.65
1546.22	1543.20	4150.38	4166.65
1627.60	1633.98	4231.76	4166.65
1708.98	1700.67	4313.14	4385.95
1790.36	1773.04	4394.52	4385.95
1871.74	1851.84	4475.90	4385.95
1953.12	1937.98	4557.28	4629.61
2034.50	2032.51	4638.66	4629.61
2115.88	2136.74	4720.04	4629.61
2197.26	2192.97	4801.42	4901.94
2278.64	2252.24	4882.80	4901.94
2360.02	2380.94	4964.18	4901.94
2441.40	2450.97	5045.56	4901.94
2522.78	2525.24	5126.94	5208.31
2604.16	2604.16	5208.32	5208.31

The resolution of the seconds parameter is .01 seconds. Any duration shorter than .005 seconds is treated as near zero. Any duration longer than 2.55 seconds is treated as 2.55 seconds.

BIN

See the LOAD statement.

BINAND

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit logical-and of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Low_bits=BINAND(Byte,15)
IF BINAND(Stat,3) THEN Bit_set
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is anded with the corresponding bit in the other argument. The results of all the ands are used to construct the integer which is returned.

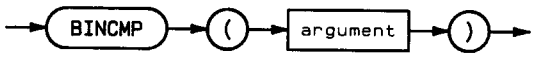
For example, the statement `Ctrl_word=BINAND(Ctrl_word,-9)` clears bit 3 of Ctrl_word without changing any other bits.

	bit 15 bit 0	
12 =	00000000 00001100	old Ctrl_word
−9 =	11111111 11110111	mask to clear bit 3
4 =	00000000 00000100	new Ctrl_word

BINCMP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of the bit-by-bit complement of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
True=BINCMP(Inverse)
PRINT X,BINCMP(X)
```

Semantics

The argument for this function is represented as a 16-bit two's-complement integer. Each bit in the representation of the argument is complemented, and the resulting integer is returned.

For example, the complement of −9:

bit 15 bit 0

$$-9 = \underline{11111111\ 11110111}$$
$$00000000\ 00001000 = 8$$

BINEOR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit exclusive-or of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Toggle=BINEOR(Toggle,1)
True_byte=BINEOR(Inverse_byte,255)
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is exclusively ored with the corresponding bit in the other argument. The results of all the exclusive ors are used to construct the integer which is returned.

For example, the statement `Ctrl_word=BINEOR(Ctrl_word,4)` inverts bit 2 of Ctrl_word without changing any other bits.

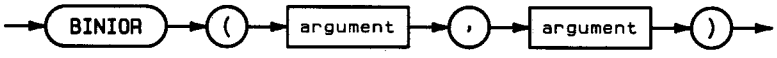
```
                bit 15                  bit 0

12 = 00000000 00001100   old Ctrl_word
 4 = 00000000 00000100   mask to invert bit 2
 8 = 00000000 00001000   new Ctrl_word
```

BINIOR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit inclusive-or of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Bits_set=BINIOR(Value1,Value2)
Top_on=BINIOR(All_bits,2^15)
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is inclusively ored with the corresponding bit in the other argument. The results of all the inclusive ors are used to construct the integer which is returned.

For example, the statement `Ctrl_word=BINIOR(Ctrl_word,6)` sets bits 1 & 2 of Ctrl_word without changing any other bits.

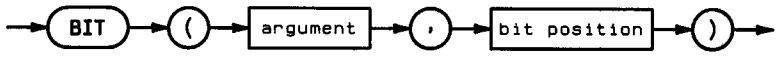
```
                bit 15          bit 0

19 =  00000000 00010011   old Ctrl_word
 6 =  00000000 00000110   mask to set bits 1 & 2
23 =  00000000 00010111   new Ctrl_word
```

BIT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a 1 or 0 representing the value of the specified bit of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767
bit position	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
Flag=BIT(Info,0)
IF BIT(Word,Test) THEN PRINT "Bit #";Test;"is set"
```

Semantics

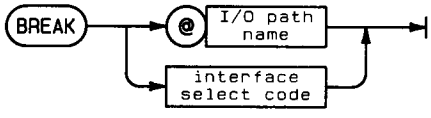
The argument for this function is represented as a 16-bit two's-complement integer. Bit 0 is the least-significant bit and bit 15 is the most-significant bit. The following example reads the controller status register of the internal HP-IB and takes a branch to "Active" if the interface is currently the active controller.

```
100  STATUS 7,3;S              ! Reg 3 = control status
110  IF BIT(S,6) THEN Active   ! Bit 6 = active control
```

BREAK

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement directs a serial or datacomm interface to send a Break sequence.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface select code	any valid name
interface select code	numeric expression, rounded to an integer	8 thru 31

Example Statements

```
BREAK 9
BREAK @Data_comm
```

Semantics

A Break sequence is a signal sent on the Data Out signal line. On the HP 98626 Serial Interface, a logic High of 400-ms duration followed by a logic Low of 60-ms duration is sent. If an outbound TRANSFER is taking place through this interface, the Break is sent after the TRANSFER is finished; the Break is sent immediately if an inbound TRANSFER is taking place. On the HP 98628 Datacomm Interface, the Break is sent immediately; the operation is identical to writing to CONTROL Register 6.

If the interface is not a serial-type interface, error 170 is reported. If an I/O path name assigned to a device selector with addressing information, error 170 is reported. If the specified interface is not present, error 163 is reported.

BUFFER

See the DIM, REAL, INTEGER, COM, ASSIGN, SUB, and DEF FN statements.

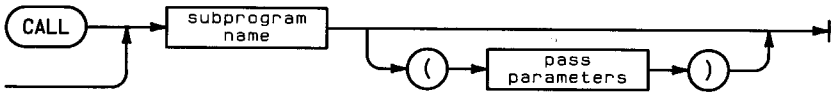
BYTE

See the ASSIGN statement.

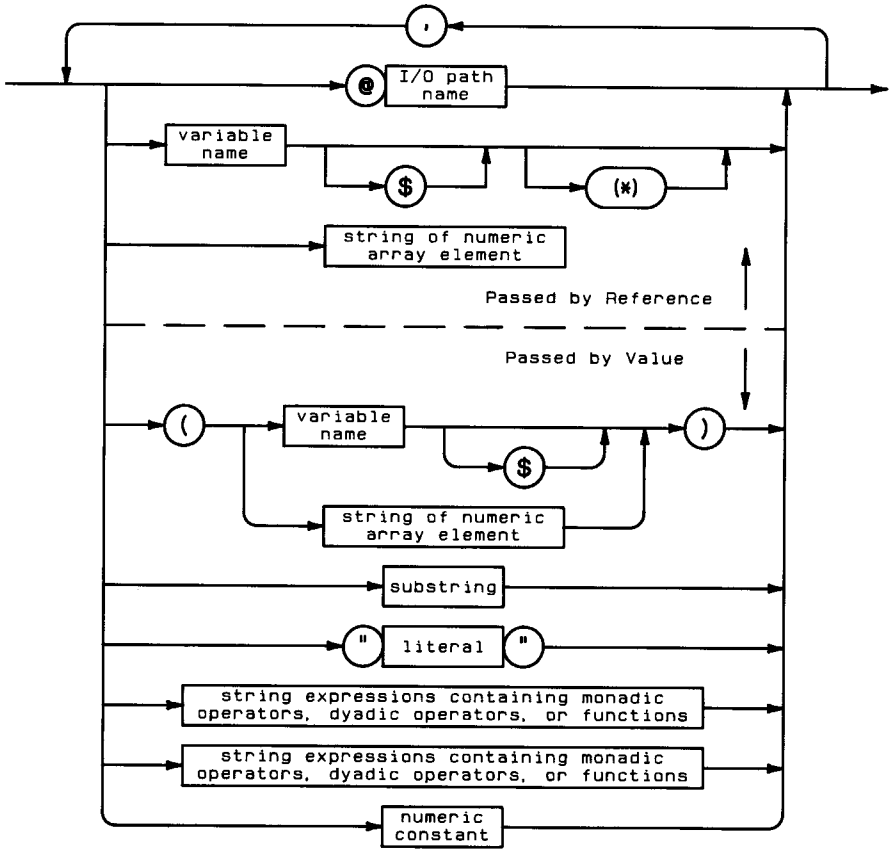
CALL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to the specified SUB or CSUB subprogram and may pass items to the subprogram. SUB programs are created with the SUB statement. (Also see the ON... statements.)



pass parameters:



Item	Description/Default	Range Restrictions
subprogram name	name of the SUB or CSUB subprograms to be called	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
variable name	name of a string or numeric variable	any valid name
substring	string expression containing substring notation	(see Glossary)
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—

Example Statements

```
CALL Process(Ref,(Value),@Path)
CALL Transform(Array(*))
IF Flag THEN CALL Special
```

Semantics

A subprogram may be invoked by a stored program line, or by a statement executed from the keyboard. Invoking a subprogram changes the program context. Subprograms may be invoked recursively. The keyword CALL may be omitted if it would be the first word in a program line. However, the keyword CALL is required in all other instances (such as a CALL from the keyboard and a CALL in an IF...THEN... statement).

The pass parameters must be of the same type (numeric, string, or I/O path name) as the corresponding parameters in the SUB or CSUB statement. Numeric values passed by value are converted to the numeric type (REAL or INTEGER) of the corresponding formal parameter. Variables passed by reference must match the corresponding parameter in the SUB statement exactly. An entire array may be passed by reference by using the asterisk specifier.

If there is more than one subprogram with the same name, the lowest-numbered subprogram is invoked by a CALL.

Program execution generally resumes at the line following the subprogram CALL. However, if the subprogram is invoked by an event-initiated branch (ON END, ON ERROR, ON INTR, ON KEY, ON KNOB, or ON TIMEOUT), program execution resumes at the point at which the event-initiated branch was permitted.

When CALL is executed from the keyboard, the current state of the computer determines the computer's state when the subprogram executes a STOP. If the computer was paused or stopped when CALL was executed, its state does not change. If the computer was running when the CALL was executed, the program pauses at the program line which was interrupted by the CALL for the subprogram, and resumes execution at that point after the subprogram is exited.

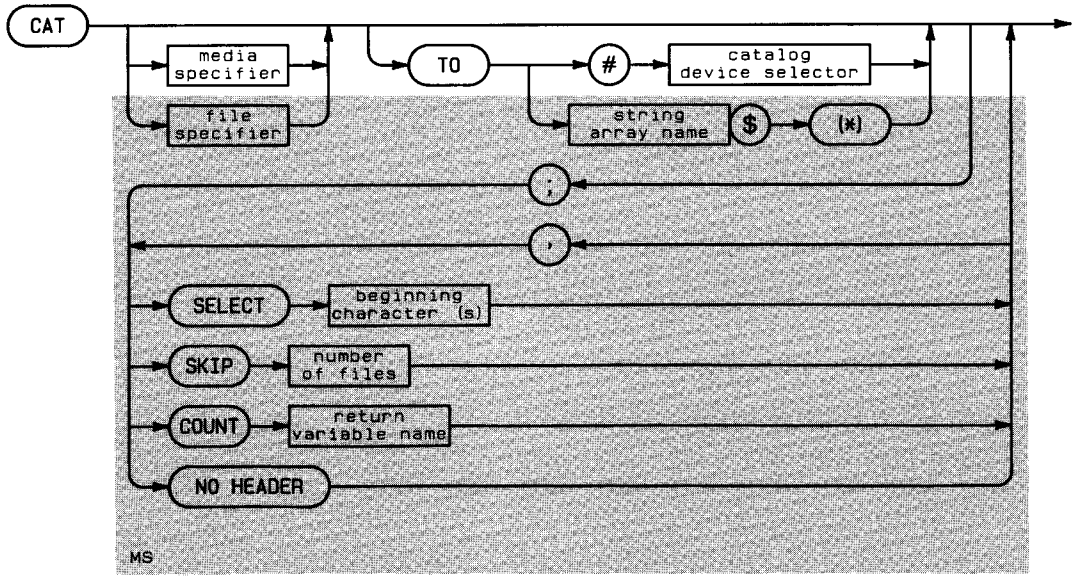
CASE

See the SELECT...CASE construct.

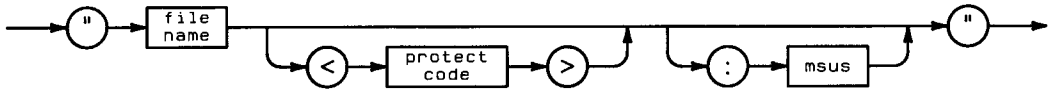
CAT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lists all or specified portions of the contents of a mass storage directory or information regarding a specified PROG file. (If using CAT with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
media specifier	string expression; Default = MASS STORAGE IS device	(see MASS STORAGE IS)
file specifier	string expression	(see drawing)
catalog device selector	numeric expression, rounded to an integer; Default = PRINTER IS device	(see Glossary)
string array name	name of a string array (see text)	any valid name
beginning character(s)	string expression	1 to 10 characters
number of files	numeric expression, rounded to an integer	1 thru 32 767
return variable name	name of a numeric variable	any valid name
msus	literal	(see MASS STORAGE IS)
file name	name of a file	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed

Example Statements

```
CAT
CAT TO #701
CAT ":INTERNAL,4,1"
CAT "Prog1"
CAT;SELECT "D",SKIP Ten_files
CAT TO Directory$(*);NO HEADER
```

Semantics

A directory entry is listed for each file on the media. The catalog shows the name of each file, whether or not it is protected, the file's type and length, the number of bytes per logical record. The types recognized in BASIC are ASCII, BDAT (BASIC data), BIN (binary program), PROG (BASIC program), or SYSTM (language system).

CAT to a Device

A protected file has an asterisk in the PRO column entry when the catalog is directed to a device. An ID number is listed for any unrecognized file types. The starting location (address) is also shown. The standard catalog format is shown below.

```
:INTERNAL
VOLUME LABEL: B9836
FILE NAME PRO TYPE  REC/FILE BYTE/REC   ADDRESS

SYSTEM_BAS    SYSTM     1024      256        16
AUTOST        PROG         5      256      1045
```

CAT to a String Array (Requires MS)

The catalog can be sent to a string array. The array must be one-dimensional, and each element of the array must contain at least 80 characters for a directory listing or 45 characters for a PROG file listing. If the directory information does not fill the array, the remaining elements are set to null strings. If the directory information "overflows" the array, the overflow is not reported as an error. When a CAT of a mass storage directory is sent to a string array, the catalog's format is different than when sent to a device. This format (the SRM directory format) is shown below. Protect status is shown by letters, instead of an asterisk. An unprotected file has the entry MRW in the PUB ACC (public access) column. A protected BDAT file has no entry in that column. Other types of protected files show R (read access). In addition to the standard information, this format also shows OPEN in the OPEN STAT column when a file is currently assigned.

```
:INTERNAL, 4
LABEL:  B9836
FORMAT: LIF
AVAILABLE SPACE:    11
                              SYS  FILE   NUMBER   RECORD     MODIFIED      PUB OPEN
FILE NAME                LEV TYPE  TYPE  RECORDS   LENGTH DATE         TIME ACC STAT
======================== === ==== ===== ======== ======== ================ === ====
SYSTEM_BAS                 1 98X6 SYSTM     1024      256                  MRW
AUTOST                     1 98X6 PROG         5      256                  MRW
```

To aid in accessing the catalog information in a string, the following table gives the location of some important fields in the string.

Field	Position (in String)
File Name	1 thru 21
File Type	32 thru 36
Number of Records	38 thru 45
Record Length	47 thru 54

Catalogs of PROG Files (Requires MS)

If the file specifier is for a PROG file, the following information is included: a list of binary programs in the file, a list of all contexts in the program, and each context's type and size. If any binary programs have a version code different from the BASIC version code, both a warning and the version codes of the binary program and BASIC system are included with the listing. CAT of a PROG file uses the same format, whether the destination is a device or a string.

```
SAMPLE
NAME                    SIZE TYPE
====================  =====  ===============
MAIN                     692 BASIC
Esc                      924 COMPILED UTILITY
FNDummy                  166 BASIC
  AVAILABLE ENTRIES = 0
```

Partial Catalogs (Requires MS)

Including the SELECT option directs the computer to list only the files that begin with or match the value of the specified string expression. If the string expression contains more than 10 characters, only the first 10 are used. If SELECT is not included, all files are sent to the destination (if possible).

Including the SKIP option directs the computer to skip the specified number of (selected) file entries before sending entries to the destination. If SKIP is not included, no files are skipped.

How Many Entries? (Requires MS)

Including COUNT provides a means of determining the number of lines sent to the destination. The variable that follows COUNT receives the sum of the number of lines in the catalog header (and trailer for PROG files) plus the number of selected files; keep in mind that the number of selected files includes the number of files sent to the destination plus the number of files skipped, if any. Catalogs sent to external devices have a five-line header; catalogs sent to string arrays have a seven-line header; and catalogs of individual PROG files have a three-line header and a one-line trailer. If an "overflow" of a string array occurs, the count is set to the number of string-array elements plus the number of files skipped. If a value of 0 is returned, no entries were sent to the destination (i.e., the number of files skipped is greater than or equal to the number of files selected).

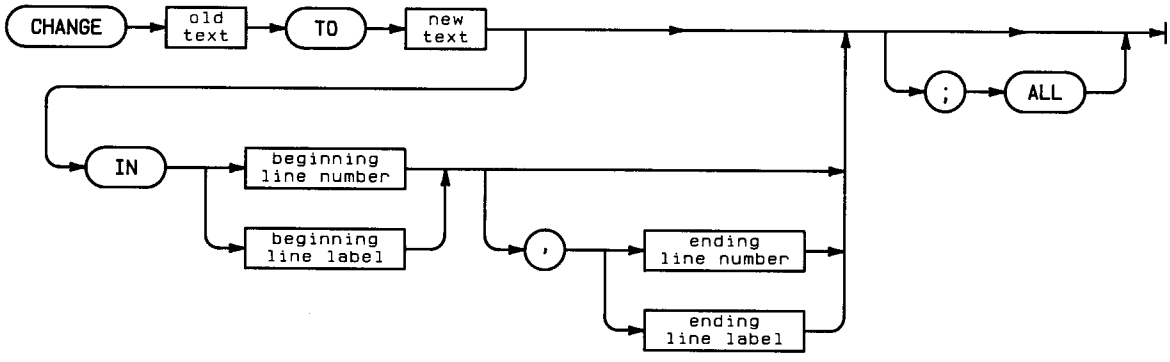
Suppressing the Heading (Requires MS)

Including the NO HEADER option directs the computer to omit the directory header (and trailer) that would otherwise be included. If NO HEADER is specified, the lines of the header (and trailer) are then omitted from the COUNT variable.

CHANGE

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No

This command allows you to search for and replace one character sequence with another while editing a program.



Item	Description/Default	Range Restrictions
old text	literal	—
new text	literal	—
beginning line number	integer constant identifying a program line	1 to 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line	1 to 32 766
ending line label	name of a program line	any valid name

Example Statements

```
CHANGE "Row" TO "Column" IN 2560,3310
CHANGE "November" TO "December";ALL
CHANGE "TREE" TO ""                    (A "delete" function)
CHANGE "his car" TO "his ""car"""      (Quotes allowed)
```

Semantics

The CHANGE command allows you to find all occurrences of a specified character sequence and replace it with another. This occurs whether they are variable names, keywords, literals, or line numbers. Note that if line numbers are changed, unexpected results may occur.

If ALL is specified, all legal changes are made automatically, without additional keyboard intervention. If ALL is not specified, the computer finds each occurrence, tentatively changes Old String to New String, and asks you to confirm the change. You confirm a particular change by pressing [ENTER] or [RETURN], and bypass a particular change by pressing [CONTINUE], [CONT], [CLEAR LINE] [ENTER] or [SHIFT] [CLEAR LINE] [ENTER] on HP 46020A. When the specified range is exhausted or the end of the program is reached, the CHANGE command is terminated and the message "<New String> not found" is displayed. [↑] and [↓] exit from the CHANGE command. [EXECUTE] confirms a change and exits the CHANGE mode.

During the course of a CHANGE, if a syntax error is caused by the altered text, the appropriate error message is displayed. When the line is corrected and entered, the CHANGE command continues.

If a change causes a line to become longer than the maximum length of a line of code, a syntax error is generated, the offending change will not take place, and the CHANGE command is aborted. The CHANGE command will also be aborted if a replacement results in the alteration of a line number, although the line whose number was changed now exists in two locations.

If the starting line number does not exist, the next line is used. If the ending line number does not exist, the previous line is used. If a line label doesn't exist, an error occurs and the CHANGE is cancelled.

If there were no occurrences found, the cursor is left at the end of the first line searched. If one or more occurrences were found, the cursor is left at the end of the line containing the last occurrence.

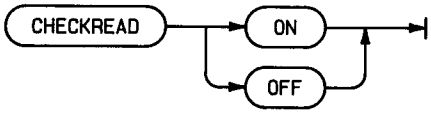
CHANGE is not allowed while a program is running; however, it may be executed while a program is paused. The program is continuable if it has not been altered by pressing [ENTER] or [DEL LN].

While in the CHANGE mode, keyboard execution is only possible with the [EXECUTE] key. Using [ENTER] or [RETURN] causes an error.

CHECKREAD

Option Required	MS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement enables or disables optional read-after-write verification of data sent to mass storage media. (If using CHECKREAD with SRM, also refer to the "SRM" section of this manual.)



Example Statements

```
IF Important_data THEN CHECKREAD ON
CHECKREAD OFF
```

Semantics

Executing CHECKREAD ON directs the computer to perform a read-after-write verification of every sector of data sent to mass storage files by any of the following statements (executed in any program context):

COPY	PRINT LABEL	RE-SAVE
CREATE ASCII	PROTECT	STORE
CREATE BDAT	PURGE	RE-STORE
OUTPUT	RENAME	TRANSFER
	SAVE	

If the bit-by-bit comparison does not detect an exact match, an error is reported.

Executing CHECKREAD OFF cancels this optional verification.

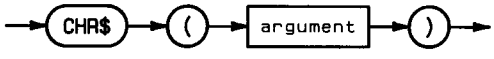
Keep in mind that using this feature may increase data reliability, but at the expense of reduced disc-access speed and increased disc wear.

CHECKREAD does not affect PRINTER IS file or PLOTTER IS file.

CHR$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a numeric value into an ASCII character. The low order byte of the 16-bit integer representation of the argument is used; the high order byte is ignored. A table of ASCII characters and their decimal equivalent values may be found in the back of this book.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	0 thru 255

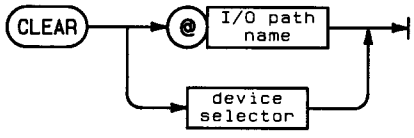
Example Statements

```
A$[Marker;1]=CHR$(Digit+128)
Esc$=CHR$(27)
```

CLEAR

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement clears HP-IB devices or Data Communications interfaces.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
CLEAR 7
CLEAR Isc*100+Address
CLEAR @Source
```

Semantics

HP-IB Interfaces

This statement allows the computer to put all or only selected HP-IB devices into a defined, device-dependent state. The computer must be the active controller to execute this statement. When primary addresses are specified, the bus is reconfigured and the SDC (Selected Device Clear) message is sent to all devices which are addressed by the LAG message.

Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN DCL	ATN MTA UNL LAG SDC	ATN DCL	ATN MTA UNL LAG SDC
Not Active Controller	Error			

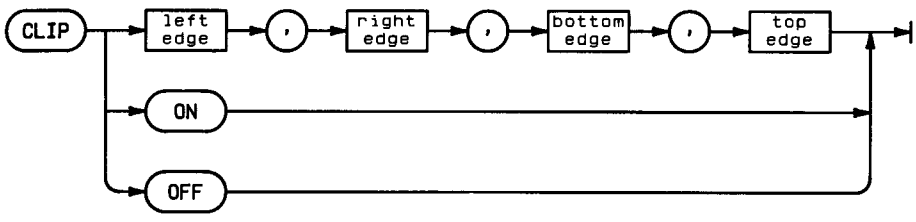
Data Communications Interfaces

CLEAR may also be directed to a Data Communications interface. The result is to clear the interface buffers; if the interface is suspended, a disconnect is also executed.

CLIP

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement redefines the soft clip area and enables or disables the soft clip limits.



Item	Description/Default	Range Restrictions
left edge	numeric expression in current units	—
right edge	numeric expression in curent units	—
bottom edge	numeric expression in current units	—
top edge	numeric expression in current units	—

Example Statements

```
CLIP Left,Right,0,100
CLIP OFF
```

Semantics

Executing CLIP with parameters allows the soft clip area to be changed from the boundary set by PLOTTER IS and VIEWPORT to the soft clip limits. If CLIP is not executed, the area most recently defined by either VIEWPORT or the PLOTTER IS statement is the clipping area. All plotted points, lines or labels are clipped at this boundary.

The hard clip area is specified by the PLOTTER IS statement. The soft clip area is specified by the VIEWPORT and CLIP statements. CLIP ON sets the soft clip boundaries to the last specified CLIP or VIEWPORT boundaries, or to the hard clip boundaries if no CLIP or VIEWPORT has been executed. CLIP OFF sets the soft clip boundaries to the hard clip limits.

CMD

See the SEND statement.

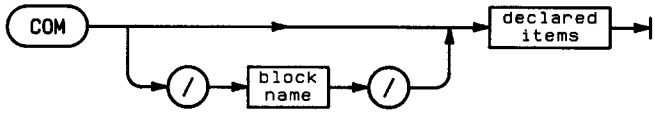
COLOR

See the AREA and SET PEN statements. See the PLOTTER IS statement for "COLOR MAP".

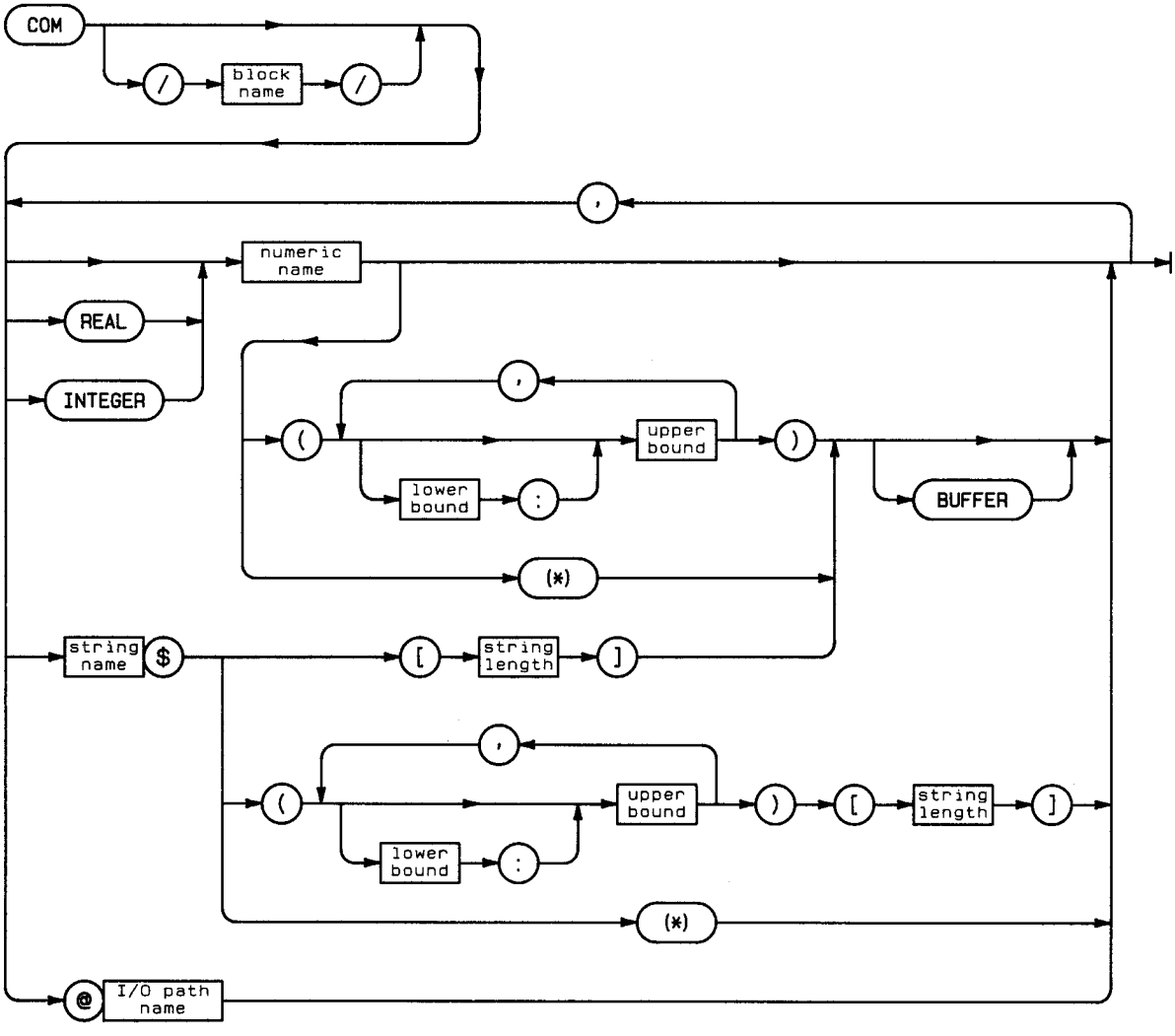
COM

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement dimensions and reserves memory for variables in a special "common" memory area so more than one program context can access the variables.



Expanded diagram:



Item	Description/Default	Range Restrictions
block name	name identifying a labeled COM area	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or 1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)
string length	integer constant	1 thru 32 767
I/O path name	name assigned to a device, devices, mass storage file, or buffer	any valid name(see ASSIGN)

Example Statements

```
COM X,Y,Z
COM /Graph/ Title$,@Device,INTEGER Points(*)
COM INTEGER I,J,REAL Array(-128:127)
COM INTEGER Buf(127) BUFFER,C$[256] BUFFER
```

Semantics

Storage for COM is allocated at prerun time in an area of memory which is separate from the data storage used for program contexts. This reserved portion of memory remains allocated until SCRATCH A or SCRATCH C is executed. Changing the definition of the COM space is accomplished by a full program prerun. This can be done by:

- Pressing the [RUN] or [STEP] key when no program is running.
- Executing a RUN command when no program is running.
- Executing any GET or LOAD from a program.
- Executing a GET or LOAD command that tells program execution to to begin.

When COM allocation is performed at prerun, the new program's COM area is compared against the COM area currently in memory. When comparing the old and new areas, the computer looks first at the types and structures declared in the COM statements. If the "text" indicates that there is no way the areas could match, then those areas are considered mismatched. If the declarations are consistent, but the shape of an array in memory does not match the shape in a new COM declaration, the computer takes the effect of REDIM into account. If the COM areas could be matched by a REDIM, they are considered to be in agreement. When this happens, the treatment of the arrays in memory depends upon the program state. If the COM matching occurred because of a programmed LOADSUB, the arrays in memory keep their current shape. If the COM matching occurred for any other reason (such as RUN or programmed LOAD), the arrays in memory are redimensioned to match the declarations. Any variable values are left intact. All other COM areas are rendered undefined, and their storage area is not recovered by the computer. New COM variables are initialized at prerun: numeric variables to 0, string variables to the null string.

Each context may have as many COM statements as needed (within the limits of computer memory), and COM statements may be interspersed between other statements. If there is an OPTION BASE statement in the context, it must appear before COM statement. COM variables do not have to have the same names in different contexts. Formal parameters of subprograms are not allowed in COM statements. A COM mismatch between contexts causes an error.

If a COM area requires more than one statement to describe its contents, COM statements defining that block may not be intermixed with COM statements defining other COM areas.

Numeric variables in a COM list can have their type specified as either REAL or INTEGER. Specifying a variable type implies that all variables which follow in the list are of the same type. The type remains in effect until another type is specified. String variables and I/O path names are considered a type of variable and change the specified type. Numeric variables are assumed to be READ unless their type has been changed to INTEGER.

COM statements (blank or labeled) in different contexts which refer to an array or string must specify it to be of the same size and shape. The lowest-numbered COM satement containing an array or string name must explicitly specify the subscript bounds and/or string length. Subsequent COM statements can reference a string by name only or an array only by using an asterisk specifier.

No array can have more than six dimensions. The total number of elements is limited by the computer's memory size. The lower bound value must be less than or equal to the upper bound value. The default lower bound is specified by the OPTION BASE statement.

Any LOADSUB which attempts to define or change COM areas while a program is running generates ERROR 145.

Unlabeled or Blank COM
Blank COM does not contain a block name in its declaration. Blank COM (if it is used) must be created in a main context. The main program can contain any number of blank COM statements. Blank COM areas can be accessed by subprograms, if the COM statements in the subprograms agree in type and shape with the main program COM statements.

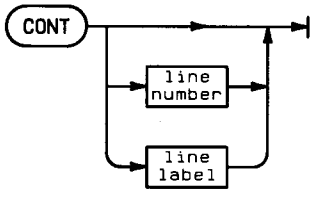
Labeled COM
Labeled COM contains a name for the COM area in its declaration. Memory is allocated for labeled COM at prerun time according to the lowest-numbered occurrence of the labeled COM statement. Each context which contains a labeled COM statement with the same label refers to the same labeled COM block.

Declaring Buffers
To declare COM variables to be buffers, each variable's name must be followed by the keyword BUFFER; the designation BUFFER applies only to the variable which it follows.

CONT

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command resumes execution of a paused program at the specified line. (For information about CONT as a secondary keyword, see the TRANSFER statement.)



Item	Description/Default	Range Restrictions
line number	integer constant identifying a program line; Default = next program line	1 thru 32 766
line label	name identifying a program line	any valid name

Example Statements

```
CONT 550
CONT Sort
```

Semantics

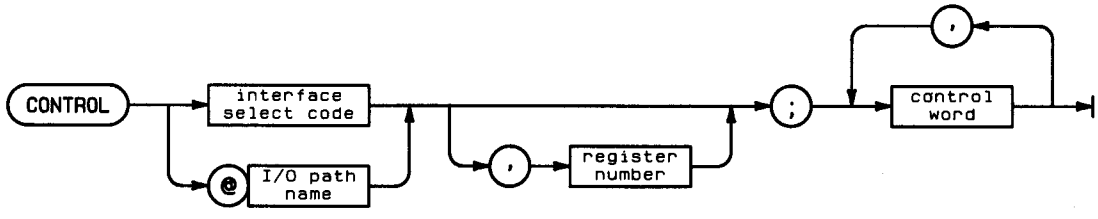
Continue can be executed by pressing the CONTINUE key or by typing a CONT command and pressing EXECUTE, ENTER or RETURN. Variables retain their current values whenever CONT is executed. CONT causes the program to resume execution at the next statement which would have occurred unless a line is specified.

When a line label is specified, program execution resumes at the specified line, provided that the line is in either the main program or the current subprogram. If a line number is specified, program execution resumes at the specified line, provided that the line is in the current program context. If there is no line in the current context with the specified line number, program execution resumes at the next higher-numbered line. If the specified line label does not exist in the proper context, an error results.

CONTROL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends control information to an interface or to the internal table associated with an I/O path name. (This keyword is also used in PASS CONTROL.)(If using CONTROL with SRM, also refer to the "SRM" section of this manual.)



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	numeric expression, rounded to an integer	1 thru 40	1 thru 32 (interface dependent)
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)	—
register number	numeric expression, rounded to an integer; Default = 0	interface dependent	—
control word	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	0 thru 65 535 (interface dependent)

Example Statements

```
CONTROL @Rand_file,7;File_length
CONTROL 1;Row,Column
CONTROL 7,3;29
```

Semantics

When the Destination is an I/O Path Name

The only time CONTROL is allowed to an I/O path name is when the I/O path name is assigned to a BDAT file or a buffer. I/O path names have an association table that can be thought of as a set of registers. Control words are written to the association table, starting with the specified "register" and continuing in turn through the remaining "registers" until all control words are used. The number of control words must not exceed the number of "registers" available. Register assignments can be found in the Interface Registers section at the back of the book.

When the Destination is an Interface

Control words are written to the interface registers, starting with the specified register number, and continuing in turn through the remaining registers until all the control words are used. The number of control words must not exceed the number of registers available. Register assignments can be found in the Interface Registers section at the back of the book.

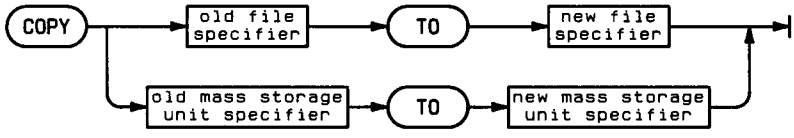
CONVERT

See the ASSIGN statement.

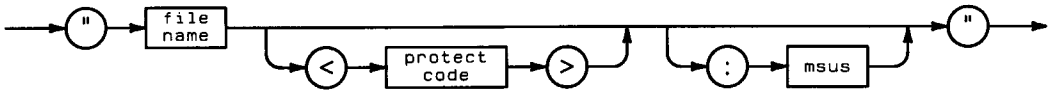
COPY

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows copying of individual files or entire discs. (If using COPY with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
mass storage unit specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
COPY "OLD_FILE" TO "New_file"
COPY File$ TO File$&Msus$
COPY ":INTERNAL,4,0" TO ":INTERNAL,4,1"
COPY Int_disc$ TO Ext_disc$
```

Semantics

Copying a File

The contents of the old file is copied into the new file, and a directory entry is created. A protect code, to prevent accidental erasure, may be specified for the new file. The old file and the new file can exist on the same device, but the new file name must be unique.

COPY is canceled and an error is returned if there is not enough room on the destination device or if the new file name already exists in the destination directory.

If the mass storage unit specifier (msus) is omitted from a file specifier, the MASS STORAGE IS device is assumed.

Copying an Entire Disc

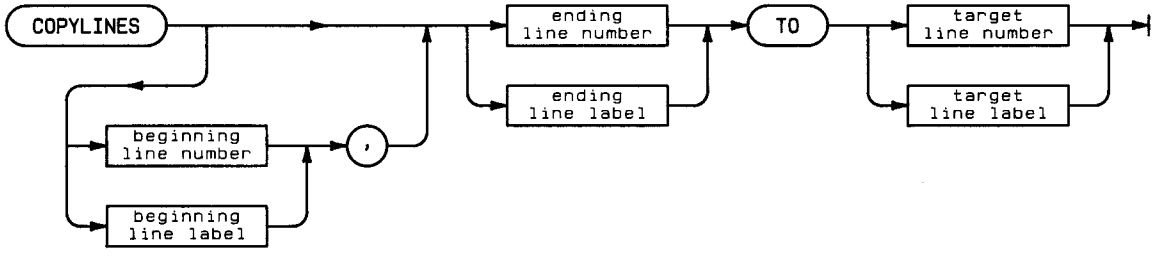
Discs can be duplicated if the destination media is as large as, or larger than, the source media. COPY from a larger capacity media to a smaller capacity media is only possible when the amount of data on the larger will fit on the smaller.. The directory and any files on the destination media are destroyed. The directory size on the destination media becomes the same size as that on the source media.

When copying a disc, msus's must be specified and unique. File names are not allowed. Disc-to-disc copy time is dependent on media type and interleave factors.

COPYLINES

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command allows you to copy one or more contiguous program lines to another location while editing a program.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying program line	1 to 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line	1 to 32 766
ending line label	name of a program line	any valid name
target line number	integer constant identifying program line	1 to 32 766
target line label	name of a program line	any valid name

Example Statements

```
COPYLINES 1200 TO 2350
COPYLINES 100,230 TO Label1
COPYLINES Util_start,Util_end TO 16340
```

Semantics

If the beginning line identifier is not specified, only one line is copied.

The target line identifier will be the line number of the first line of the copied program segment. Copied lines are renumbered if necessary. The code (if any) which is "pushed down" to make room for the copied code is renumbered if necessary.

Line number references to the copied code are updated as they would be for a REN command, with these exception: Line number references in lines not being copied remain linked to the source lines rather than being renumbered. Also, references to non-existent lines are renumbered as if the lines existed.

If there are any DEF FN or SUB statements in the copied code, the target line number must be greater than any existing line number.

If you try to copy a program segment to a line number **contained** in the segment, an error will result and no copying will occur.

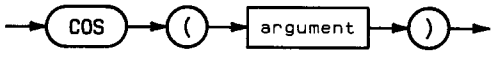
If the starting line number does not exist, the next line is used. If the ending line number does not exist, the previous line is used. If a line label doesn't exist, an error occurs and no copying occurs.

If an error occurs *during* a COPYLINES (for example, a memory overflow), the copy is terminated and the program is left partially modified.

COS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the cosine of the argument. The range of the returned real value is -1 thru $+1$.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 1.708 312 772 2 E+10 deg. or in radians: 2.981 568 244 292 04 E+8

Example Statements

```
Cosine=COS(Angle)
PRINT COS(X+45)
```

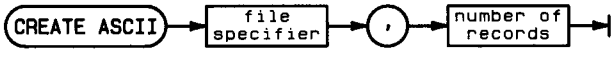
COUNT

See the CAT and TRANSFER statements.

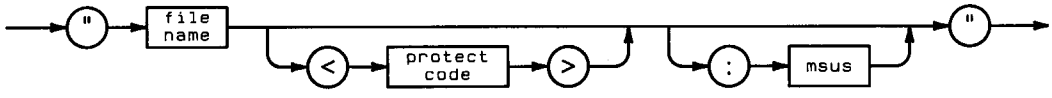
CREATE ASCII

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file on the mass storage media. (If using CREATE ASCII with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
msus	literal	(see MASS STORAGE IS)
number of records	numeric expression, rounded to an integer	1 thru $2^{31}-1$

Example Statements

```
CREATE ASCII "TEXT",100
CREATE ASCII Name$&":INTERNAL",Length
```

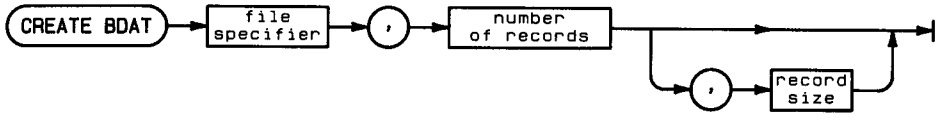
Semantics

CREATE ASCII creates a new ASCII file and directory entry on the mass storage media. CREATE ASCII does not open the file. Opening of files is done by the ASSIGN statement. The physical records of an ASCII file have a fixed length of 256 bytes; logical records have variable lengths, which are automatically determined when the OUTPUT, SAVE, or RE-SAVE statements are used. In the event of an error, no directory entry is made and the file is not created.

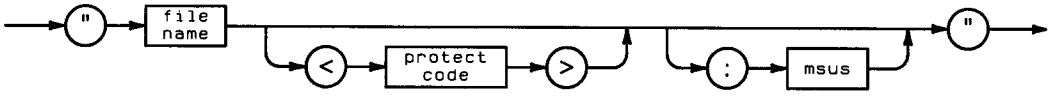
CREATE BDAT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a BDAT file on the mass storage media. (If using CREATE BDAT with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)
number of records	numeric expression, rounded to an integer	1 thru $2^{31}-256$
record size	numeric expression, rounded to next even integer (except 1). Specifies bytes/record. Default = 256	1 thru 65 534

Example Statements

```
CREATE BDAT "George",48
CREATE BDAT "Special<PC>",Length,128
CREATE BDAT Name$&Msus$,Bytes,1
```

Semantics

CREATE BDAT creates a new BDAT file and directory entry on the mass storage media. CREATE BDAT does not open the file. Opening of files is done by the ASSIGN statement. If a protect code is included after the file name, the first two characters become the protect code of the file. In the event of an error, no directory entry is made and the file is not created. A sector is created at the beginning of the file for system use. This sector cannot be accessed by BASIC programs.

CRT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This INTEGER function returns 1, the device selector of the alpha display.



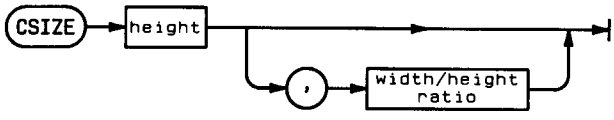
Example Statements

```
PRINTER IS CRT
ENTER CRT;Array(*)
```

CSIZE

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sets the size and aspect (width/height) ratio of the character cell used by the LABEL and SYMBOL statements.



Item	Description/Default	Range Restrictions
height	numeric expression; Default = 5	—
width/height ratio	numeric expression; Default = 0.6	—

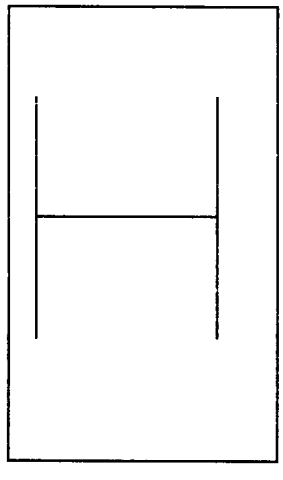
Example Statements

```
CSIZE 10
CSIZE Size,Width
```

Semantics

At power-on, RESET, and GINIT, the height is 5 graphic-display-units (GDUs), and the aspect ratio is 0.6 (width = 3 GDUs, or 0.6 × 5 GDUs). A negative number for either parameter inverts the character along the associated dimension. The drawing below shows the relation between the character cell and a character.

Character in a Character Cell



CSUB

This keyword stands for "Compiled SUBprogram". CSUB statements are created in Pascal using a special CSUB preparation utility. They are loaded using the LOADSUB statement and can be deleted using the DELSUB statement. When viewed in BASIC's edit mode, these subprograms look like SUB statements, except for the keyword. They are invoked with CALL, just like normal SUB subprograms.

Because of their special nature, certain rules must be followed when editing a program containing CSUB statments. These lines will not be recognized if entered in BASIC (they must be created in Pascal). Therefore, any operation which requires the line to be checked for proper syntax will fail. This includes such operations as GET, MOVELINES, and the EXECUTE, ENTER, RETURN keys. Operations which do not check syntax are allowed. This includes things like scrolling and re-numbering.

Sometimes a CSUB will appear as multiple CSUB statements because of multiple entry points. In these cases, the group of statements cannot be broken; you cannot insert a comment line between the statements, delete a single statement in the group, or interfere with the order in any way.

CSUM

See the MAT statement.

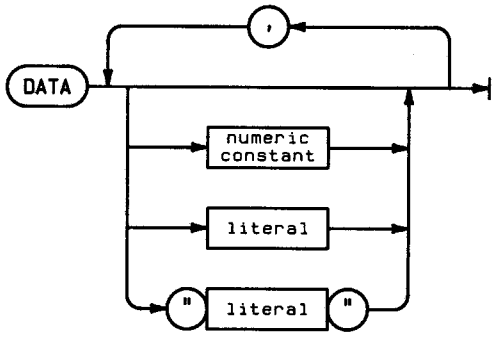
CYCLE

See the OFF CYCLE and the ON CYCLE statements.

DATA

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement contains data which can be read by READ statements. (For information about DATA as a secondary keyword, see the SEND statement.)



Item	Description/Default	Range Restrictions
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—

Example Statements

```
DATA 1,1.414,1.732,2
DATA word1,word2,word3
DATA "ex-point(!)","quote("")","comma(,)"
```

Semantics

A program or subprogram may contain any number of DATA statements at any locations. When a program is run, the first item in the lowest numbered DATA statement is read by the first READ statement encountered. When a subprogram is called, the location of the next item to be read in the calling context is remembered in anticipation of returning from the subprogram. Within the subprogram, the first item read is the first item in the lowest numbered DATA statement within the subprogram. When program execution returns to the calling context, the READ operations pick up where they left off in the DATA items.

A numeric constant must be read into a variable which can store the value it represents. The computer cannot determine the intent of the programmer; although attempting to read a string value into a numeric variable will generate an error, numeric constants will be read into string variables with no complaint. In fact, the computer considers the contents of all DATA statements to be literals, and processes items to be read into numeric variables with a VAL function, which can result in error 32 if the numeric data is not of the proper form (see VAL).

Unquoted literals may not contain quote marks (which delimit strings), commas (which delimit data items), or exclamation marks (which indicate the start of a comment). Leading and trailing blanks are deleted from unquoted literals. Enclosing a literal in quote marks enables you to include any punctuation you wish, including quote marks, which are represented by a set of two quote marks.

DATE

Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts the formatted date (DD MMM YYYY) into a numeric value used to set the clock.



literal form of formatted date:



Item	Description/Default	Range Restrictions	Recommended Range
formatted date	string expression	(see drawing)	(see text)
day	integer constant	1 thru the end-of-month	—
month	literal; letter case ignored	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC	—
year	integer constant	−1 469 899 thru 1 469 899	1900 thru 2079
delimiter	literal; single character	(see text)	space

Example Statements

```
PRINT DATE("26 MAR 1982")
SET TIMEDATE DATE("1 Jan 1983")
Days=(DATE("1 JAN 1983")-DATE("11 NOV 1982")) DIV 86400
```

Semantics

Using a value from the DATE function as the argument for SET TIMEDATE will set the clock to midnight on the date specified. Results from the DATE and TIME functions must be combined to set the date and time of day.

If the DATE function is used as an argument for SET TIMEDATE to set the clock, the date must be in the range: 1 Mar 1900 thru 4 Aug 2079.

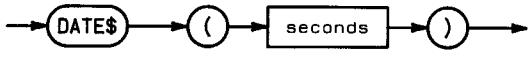
Specifying an invalid date, such as the thirty-first of February, will result in an error.

Leading blanks or non-numeric characters are ignored. ASCII spaces are recommended as delimiters between the day, month and year. However, any non-alphanumeric character, except the negative sign (–), may be used as the delimiter.

DATE$

Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function formats a number of seconds as a date (DD MMM YYYY).



Item	Description/Default	Range Restrictions	Recommended Range
seconds	numeric expression	−4.623 683 256 E+13 thru 4.653 426 335 039 9 E+13	2.086 629 12 E+11 thru 2.143 252 224 E+11

Example Statements

```
PRINT DATE$(TIMEDATE)
DISP DATE$(2.112520608E+11)
```

Semantics

The date returned is in the form: DD MMM YYYY, where DD is the day of the month, MMM is the month mnemonic, and YYYY is the year.

The day is blank filled to two character positions. Single ASCII spaces delimit the day, month, and year.

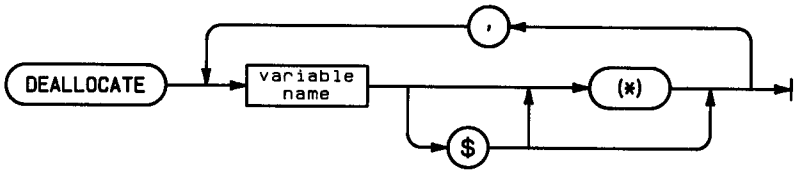
The first letter of the month is capitalized and the rest are lowercase characters.

Years less than the year 0 are expressed as negative years.

DEALLOCATE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deallocates memory space reserved by the ALLOCATE statement.



Item	Description/Default	Range Restrictions
variable name	name of an array or string variable	any valid name

Example Statements

```
DEALLOCATE A$,B$,C$
DEALLOCATE Array(*)
```

Semantics

Memory space reserved by ALLOCATE exists in the same section of memory as that used by ON-event statements. Since entries in this area are "stacked" as they come in, space for variables which have been DEALLOCATED may not be available immediately. It will not be available until all the space "above it" is freed. This includes variables allocated after it, as well as ON-event entries. Exiting a subprogram automatically deallocates space for variables which were allocated in that subprogram.

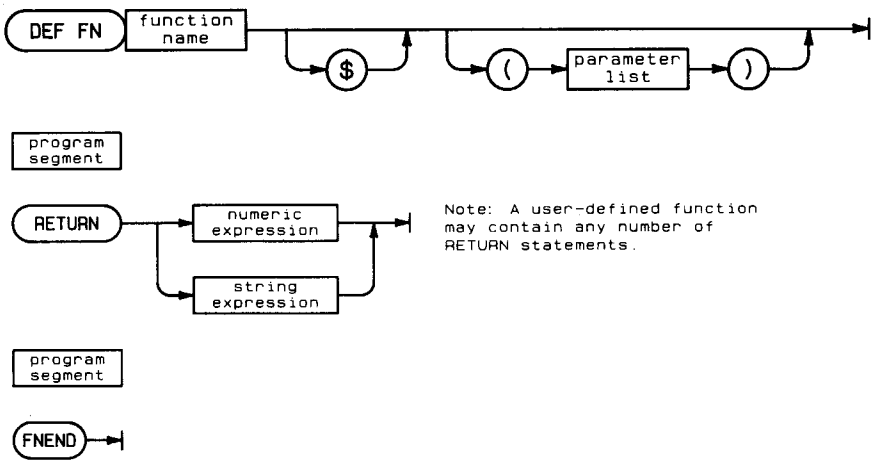
Strings and arrays must be deallocated completely. Deallocation of an array is requested by the (*) specifier.

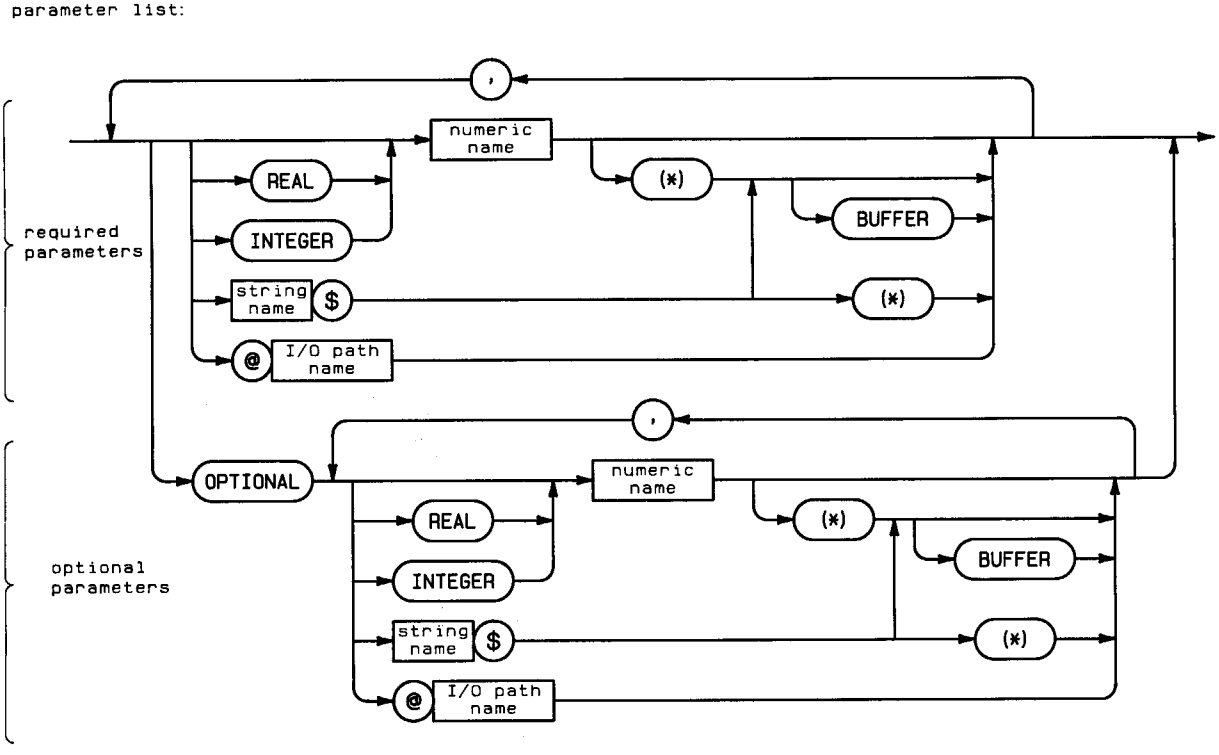
Attempting to DEALLOCATE a variable which is not currently allocated in the current context results in an error. When DEALLOCATE is executed from the keyboard, deallocation occurs within the current context.

DEF FN

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement indicates the beginning of a function subprogram. It also indicates whether the function is string or numeric and defines the formal parameter list.





Item	Description/Default	Range Restrictions
function name	name of the user-defined function	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Statements

```
DEF FNTrim$(String$)
DEF FNTransform(@Printer,INTEGER Array(*),OPTIONAL Text$)
```

Semantics

User-defined functions must appear after the main program. The first line of the function must be a DEF FN statement. The last line must be an FNEND statement. Comments after the FNEND are considered to be part of the function.

Parameters to the left of the keyword OPTIONAL are required and must be supplied whenever the user-defined function is invoked (see FN). Parameters to the right of OPTIONAL are optional, and only need to be supplied if they are needed for a specific operation. Optional parameters are associated from left to right with any remaining pass parameters until the pass parameter list is exhausted. An error is generated if the function tries to use an optional parameter which did not have a value passed to it. The function NPAR can be used to determine the number of parameters supplied by the function call.

Variables in a subprogram's formal parameter list may not be declared in COM or other declaratory statements within the subprogram. A user-defined function may not contain any SUB statements or DEF FN statements. User-defined functions can be called recursively and may contain local variables. A unique labeled COM must be used if the local variables are to preserve their values between invocations of the user-defined function.

The RETURN <expression> statement is important in a user-defined function. If the program actually encounters an FNEND during execution (which can only happen if the RETURN is missing or misplaced), error 5 is generated. The <expression> in the RETURN statement must be numeric for numeric functions, and string for string functions. A string function is indicated by the dollar sign suffix on the function name.

The purpose of a user-defined function is to compute a single value. While it is possible to alter variables passed by reference and variables in COM, this can produce undesirable side effects, and should be avoided. If more than one value needs to be passed back to the program, SUB subprograms should be used.

If you want to use a formal parameter as a BUFFER, it must be declared as a BUFFER in both the formal parameter list and the calling context.

DEG

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects degrees as the unit of measure for expressing angles.



Semantics

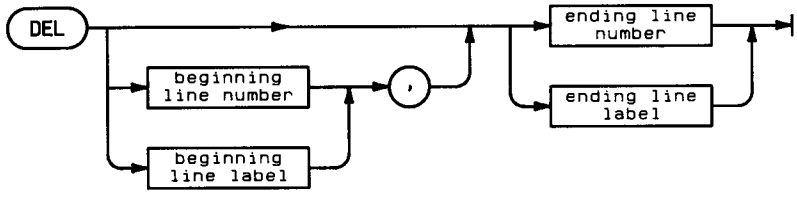
All functions which return an angle will return an angle in degrees. All operations with parameters representing angles will interpret the angle in degrees.

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context. If no angle mode is specified in a program, the default is radians (see RAD).

DEL

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command deletes program lines.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying a program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
DEL 15
DEL Sort,9999
```

Semantics

DEL cannot be executed while a program is running. If DEL is executed while a program is paused, the computer changes to the stopped state.

When a line is specified by a line label, the computer uses the lowest numbered line which has the label. If the label does not exist, error 3 is generated. An attempt to delete a non-existent program line is ignored when the line is specified by a line number. An error results if the ending line number is less then the beginning line number. If only one line is specified, only that line is deleted.

When deleting SUB and FN subprograms, the range of lines specified must include the statements delimiting the beginning and ending of the subprogram (DEF FN and FNEND for user-defined function subprograms; SUB and SUBEND for SUB subprograms), as well as all comments following the delimiting statement for the end of the subprogram. Contiguous subprograms may be deleted in one operation.

DELAY

See the ASSIGN, OFF DELAY, ON DELAY, PRINTALL IS, and PRINTER IS statements.

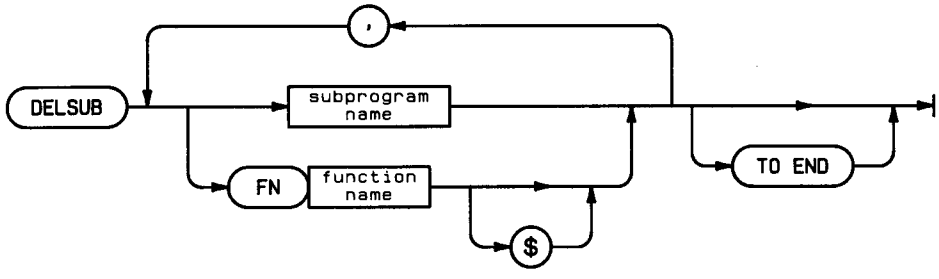
DELIM

See the TRANSFER statement.

DELSUB

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deletes one or more SUB subprograms or user-defined function subprograms from memory.



Item	Description/Default	Range Restrictions
subprogram name	name of a SUB or CSUB subprogram	any valid name
function name	name of a user-defined function	any valid name

Example Statements

```
DELSUB FNTrim$
DELSUB Special1,Special3
```

Semantics

Subprograms being deleted do not need to be contiguous in memory. The order of the names in the deletion list does not have to agree with the order of the subprograms in memory. If there are subprograms with the same name, the one occurring first (lowest line number) is deleted.

The lines deleted begin with the line delimiting the beginning of the subprogram (SUB or DEF FN) and include the comments following the line delimiting the end of the subprogram (SUBEND or FNEND). If TO END is included, all subprograms following the specified subprogram are also deleted, from the last subprogram to the specified subprogram.

You cannot delete:

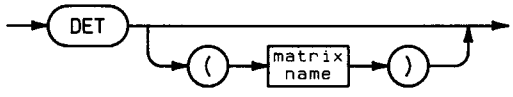
- Busy subprograms (ones being executed).
- Subprograms which are referenced by active ON-event CALL statements.

If an error occurs while attempting to delete a subprogram with a DELSUB statement, the subprogram is not deleted, and neither are subprograms listed to the right of the subprogram which could not be deleted.

DET

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This REAL function returns the determinant of a matrix.



Item	Description/Default	Range Restrictions
matrix name	name of a square, two-dimensional numeric array; Default = see text	any valid name

Example Statements

```
Determinant=DET
PRINT DET(A)
```

Semantics

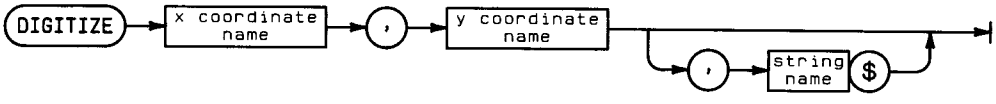
If you do not specify a matrix, DET returns the determinant of the most recently inverted matrix. This value is not affected by context switching. If no matrix has been inverted since power-on, pre-run, SCRATCH or SCRATCH A, 0 is returned.

The determinant is significant as an indication of whether an inverse is valid. If the determinant of a matrix equals 0, then the matrix has no inverse. If the determinant is very small compared with the elements of its matrix, then the inverse may be invalid and should be checked.

DIGITIZE

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement inputs the X and Y coordinates of a digitized point from the locator specified by GRAPHICS INPUT IS.



Item	Description/Default	Range Restrictions
x coordinate name	name of a numeric variable	any valid name
y coordinate name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name

Example Statements

```
DIGITIZE X,Y
IF Flag THEN DIGITIZE Xpos,Ypos,Status$
```

Semantics

The returned coordinates are in the unit-of-measure currently defined for the PLOTTER IS and GRAPHICS INPUT IS devices. The unit-of-measure may be default units or those defined by either the WINDOW or SHOW statement. If an INTEGER numeric variable is specified and the value entered is out of range, error 20 is reported.

If graphics input is from the keyboard, DIGITIZE is satisfied by pressing any of the following keys: EXECUTE, EXEC, ENTER, RUN, STOP, RETURN, PAUSE, STEP, CONTINUE, and CONT.

The optional string variable is used to input the device status of the GRAPHICS INPUT IS device. This status string contains eight bytes, defined as follows.

Byte	1	2	3	4	5	6	7	8
Meaning	Digitize Status	,	Point Significance	,	Tracking On/Off	,	Button Number	

Byte 1: Digitize status; If the locator device supports only single point digitizing, this byte is always a "1". If the locator device supports continuous digitizing, this byte is a "1" for all points in a stream of continuous points **except** the last point, which will be returned with a "0". The method of indicating the beginning and ending of a continuous point stream is device dependent. If the numeric value represented by this byte is used as the pen control value for a PLOT statement, continuous digitizing will be copied to the display device.

Bytes 2, 4, and 6: commas; used as delimiters.

Bytes 3: Significance of digitized point; "0" indicates that the point is outside the P1, P2 limits; "1" indicates that the point is outside the viewport, but inside the P1, P2 limits; "2" indicates that the point is inside the current viewport limits.

Byte 5: Tracking status; "0" indicates off, "1" indicates on.

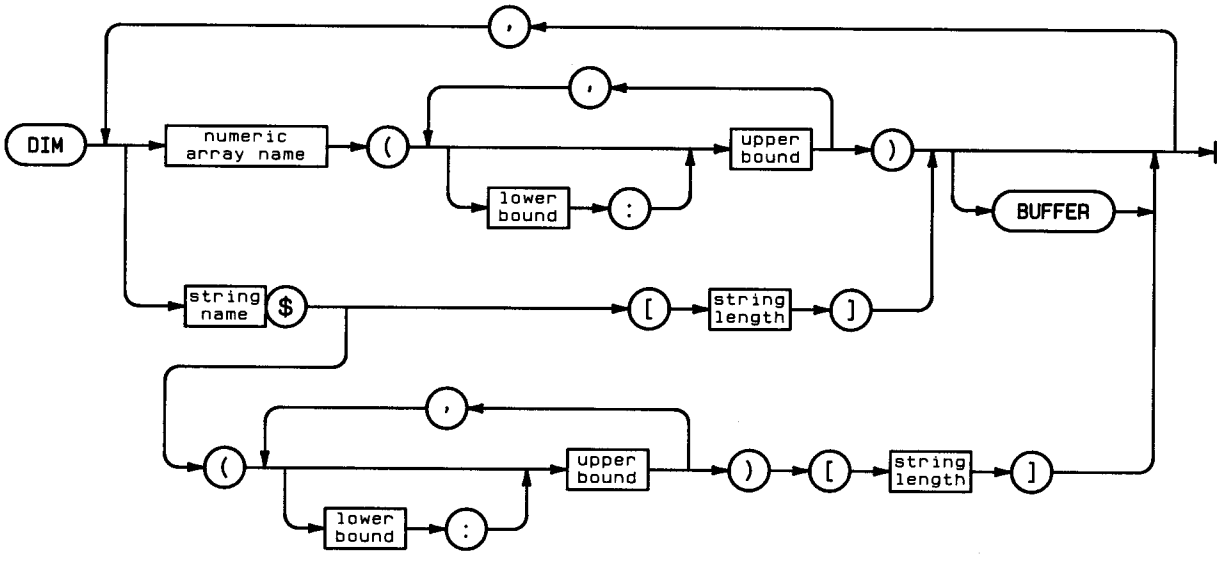
Byte 7 and 8: The number of the buttons which are currently down. To interpret the ASCII number returned, change the number to its binary form and look at each bit. If the bit is "1", the corresponding button is down. If the bit is "0", the corresponding button is not down.

If the locator device (e.g., stylus or puck) goes out of proximity, a "button 7" is indicated in the "button number" bytes. Bytes 7 and 8 will be exactly "64" regardless of whether any actual buttons are being held down at the time. Proximity is reported only from HP-HIL locators; the HP 9111A always returns "00" in bytes 7 and 8. On a 35723A TouchScreen, going out of proximity (i.e., removing your finger from the screen) will trigger a digitize. Coming into proximity on a tablet with a button pressed will also trigger a digitize, even if the button was originally pressed while in proximity.

DIM

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement dimensions and reserves memory for REAL numeric arrays, strings and string arrays.



Item	Description/Default	Range Restrictions
numeric array name	name of a numeric array	any valid name
string name	name of a string variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or 1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)
string length	integer constant	1 thru 32 767

Example Statements

```
DIM String$[100],Name$(12)[32]
DIM Array(-128:127,16)
DIM String_scaler$[256] BUFFER, Real_array(127) BUFFER
```

Semantics

A program can have any number of DIM statements. The same variable cannot be declared twice within a program (variables declared in a subprogram are distinct from those declared in a main program, except those declared in COM). The DIM statements can appear anywhere within a program, as long as they do not precede an OPTION BASE statement. Dimensioning occurs at pre-run or subprogram entry time. Dynamic run time allocation of memory is provided by the ALLOCATE statement.

No array can have more than six dimensions. Each dimension can have a maximum of 32 767 elements. The actual maximum number of elements for an array depends on available memory.

All numeric arrays declared in a DIM statement are REAL, and each element of type REAL requires 8 bytes of storage. A string requires one byte of storage per character, plus two bytes of overhead.

An undeclared array is given as many dimensions as it has subscripts in its lowest-numbered occurrence. Each dimension of an undeclared array has an upper bound of ten. Space for these elements is reserved whether you use them or not. Any time a lower bound is not specified, it defaults to the OPTION BASE value.

Declaring Buffers

To declare variables to be buffers, each variable's name must be followed by the keyword BUFFER; the designation BUFFER applies only to the variable which it follows. String arrays cannot be declared to be buffers.

DISABLE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables all event-initiated branches currently defined, except ON END, ON ERROR, and ON TIMEOUT.



Semantics

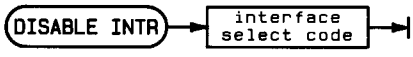
If an event occurs while the currently defined event-initiated branches are disabled, only the first occurrence of each event is logged; there is no record of how many of each type of event has occurred.

If event-initiated branches are enabled after being disabled, all logged events will initiate their respective branches if and when system priority permits. ON ERROR, ON END, as ON TIMEOUT branches are **not** disabled by DISABLE.

DISABLE INTR

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables interrupts from an interface by turning off the interrupt generating mechanism on the interface.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	5, and 7 thru 31

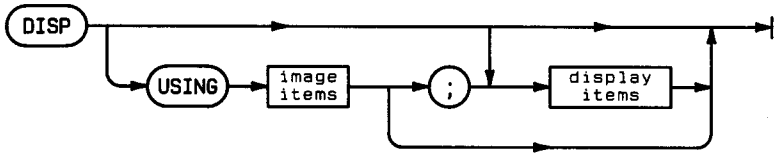
Example Statements

```
DISABLE INTR 7
DISABLE INTR Isc
```

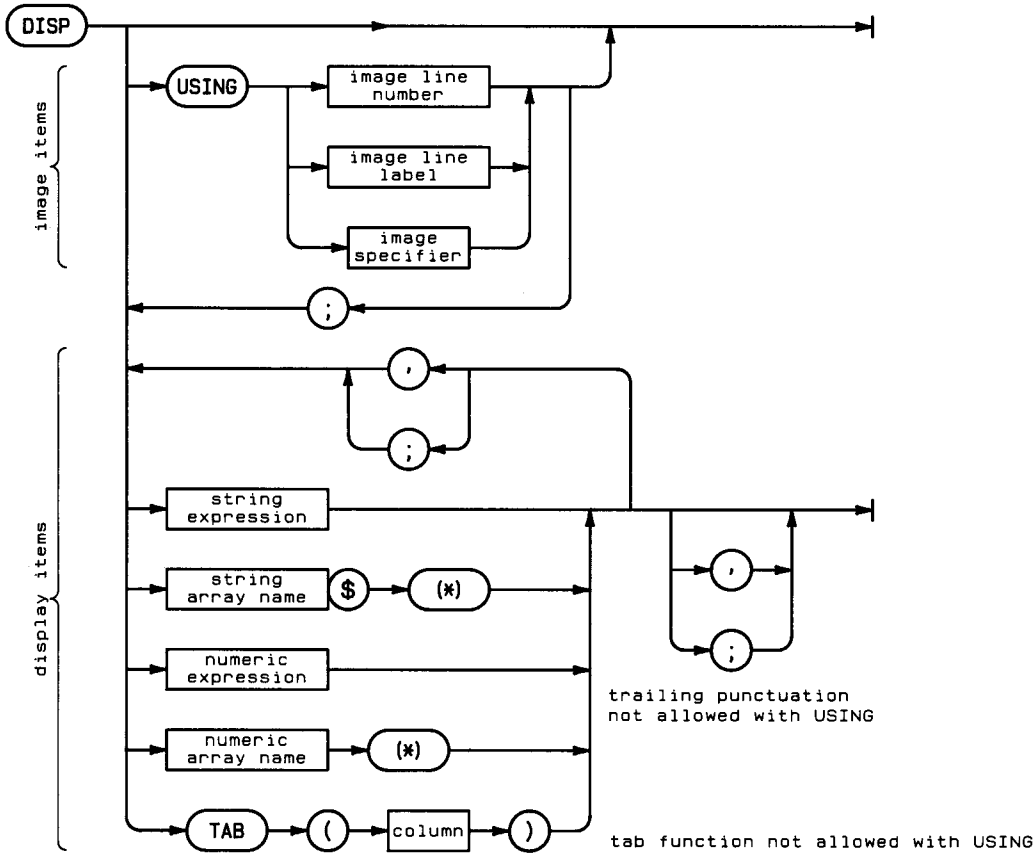
DISP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

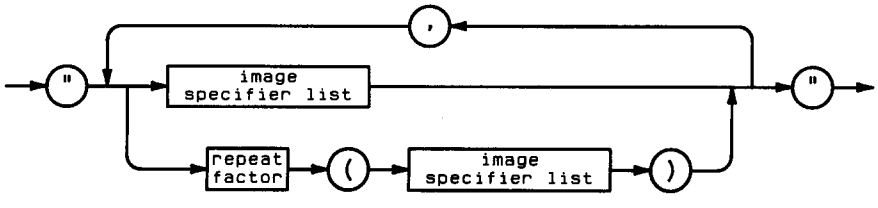
This statement causes the display items to be sent to the display line on the CRT.



Expanded diagram:



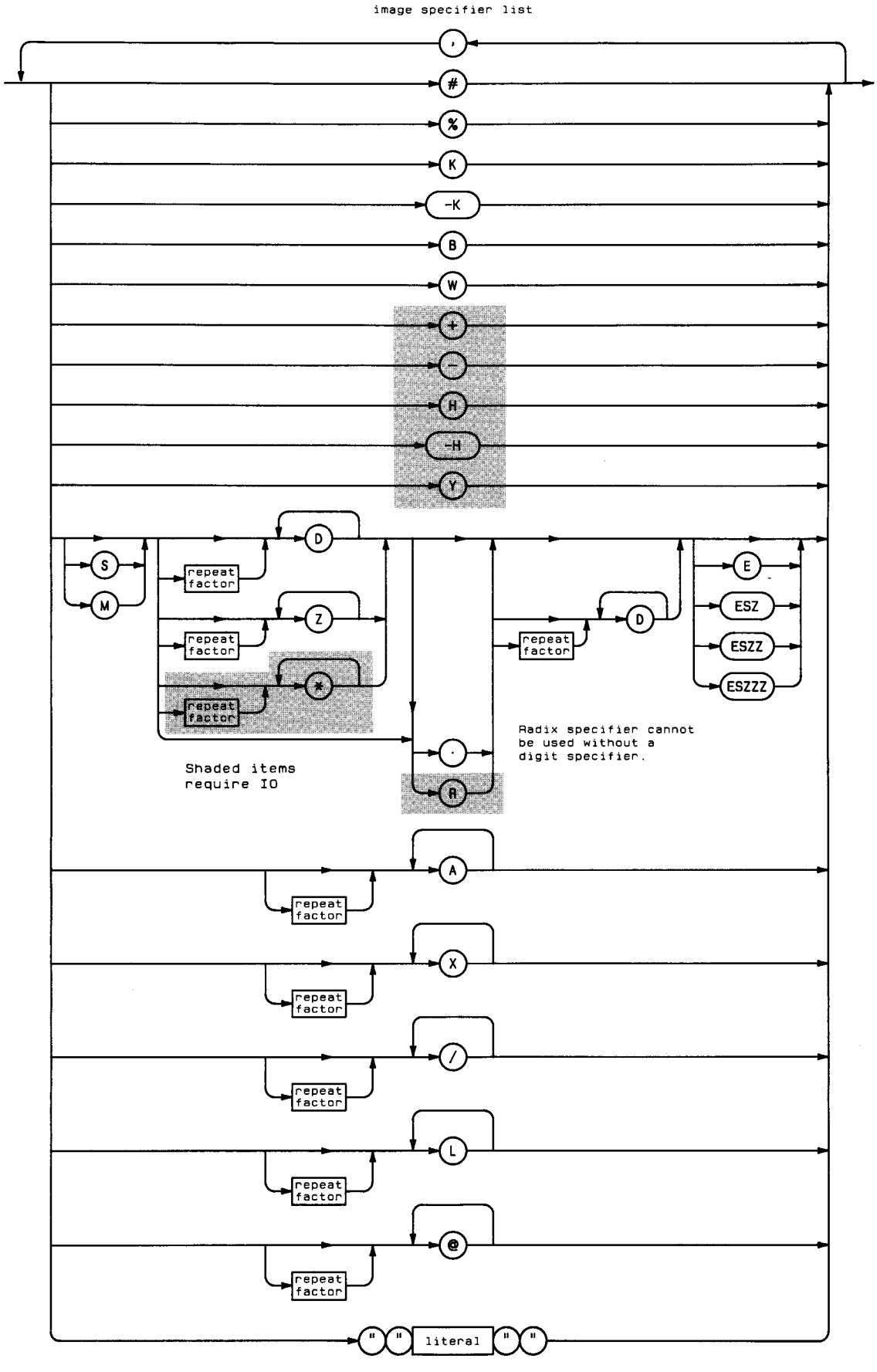
literal form of image specifier:



Item	Description/Default	Range Restrictions	Recommended Range
image line label	name identifying an IMAGE statement	any valid name	—
image line number	integer constant identifying an IMAGE statement	1 thru 32 766	—
image specifier	string expression	(see drawing)	—
string array name	name of a string array	any valid name	—
numeric array name	name of a numeric array	any valid name	—
column	numeric expression, rounded to an integer	−32 768 thru +32 767	1 thru screenwidth
image specifier list	literal	(see next drawing)	—
repeat factor	integer constant	1 thru 32 767	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed	—

Example Statements

```
DISP Prompt$;
DISP TAB(5),First,TAB(20),Second
DISP USING "5Z.DD";Money
```


image specifier list
,
#
%
K
-K
B
W
+
–
H
-H
Y
S
M
D
repeat factor
Z
*
E
ESZ
ESZZ
ESZZZ
.
R
Shaded items require IO
Radix specifier cannot be used without a digit specifier.
A
X
/
L
@
" " literal " "

Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E – 4 and less than 1E + 6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the display list is exhausted, an End Of Line (EOL) sequence is sent to the display line, unless it is suppressed by trailing punctuation or a pound-sign image specifier.

Control Codes

Some ASCII control codes have a special effect in DISP statements:

Character	Keystroke	Name	Action
CHR$(7)	CTRL-G	bell	Sound the beeper
CHR$(8)	CTRL-H	backspace	Move the cursor back one character.
CHR$(12)	CTRL-L	formfeed	Clear the display line.
CHR$(13)	CTRL-M	carriage return	Move cursor to column 1. The next character sent to the display clears the display line, unless it is a carriage return

CRT Enhancements

There are several character enhancements (such as inverse and underlining) available on some CRTs. They are accessed through characters with decimal values above 127. For a list of the characters and their effects, see the "Display Enhancement Characters" table in "Useful Tables" at the back of this book.

Arrays

Arrays may be displayed in their entirety by using the asterisk specifier. They are displayed in row-major order (right-most subscript varies most rapidly) and their format depends on the print mode selected.

Display Without USING

If DISP is used without USING, the punctuation following an item determines the width of the item's display field; a semicolon selects the compact field, and a comma selects the default display field. When the display item is an array with the asterisk array specifier, each array element is considered a separate display item. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the display field to be used for the display item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are displayed with one trailing blank. String items are displayed with no leading or trailing blanks.

The default display field displays items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is displayed with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

In the TAB function, a column parameter less than one is treated as one. A column parameter greater than the screen width (in characters) is treated as equal to the screen width.

Display With USING

When the computer executes a DISP USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the display items, the field specifier is acted upon without accessing the display list. When the field specifer requires characters, it accesses the next item in the display list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching display item. If the image specifiers are exhausted before the display items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

Effects of the image specifiers on the DISP statement are shown in the following table:

Image Specifier	Meaning
K	Compact field. Displays a number or string in standard form with no leading or trailing blanks.
-K	Same as K.
H	Similar to K, except the number is displayed using the European number format (comma radix). (Requires IO)
-H	Same as H. (Requires IO)
S	Displays the number's sign (+ or -).
M	Displays the number's sign if negative, a blank if positive.
D	Displays one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is displayed, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are displayed.
*	Same as Z, except that asterisks are displayed instead of leading zeros. (Requires IO)

Image Specifier	Meaning
.	Displays a decimal-point radix indicator.
R	Displays a comma radix indicator (European radix). (Requires IO)
E	Displays an E, a sign, and a two-digit exponent.
ESZ	Displays an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Displays an E, a sign, and a three-digit exponent.
A	Displays a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Displays a blank.
literal	Displays the characters contained in the literal.
B	Displays the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER, and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.
W	Displays two characters represented by the two bytes of a 16-bit, two's-complement integer. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is used; if it is less than −32 768, then −32 768 is used. The most-significant byte is sent first.
Y	Same as W. (Requires IO)
#	Suppresses the automatic output of an EOL (End-Of-Line) sequence following the last display item.
%	Ignored in DISP images.
+	Changes the automatic EOL sequence that normally follows the last display item to a single carriage-return. (Requires IO)
−	Changes the EOL automatic sequence that normally follows the last display item to a single line-feed. (Requires IO)
/	Sends a carriage-return and a line-feed to the display line.
L	Same as /.
@	Sends a form-feed to the display line.

DIV

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns the integer portion of the quotient of the dividend and the divisor.



Item	Description/Default	Range Restrictions
dividend	numeric expression	—
divisor	numeric expression	not equal to 0

Example Statements

```
Quotient=Dividend DIV Divisor
PRINT "Hours =";Minutes DIV 60
```

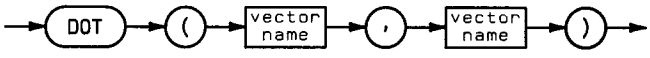
Semantics

DIV returns a REAL value unless both arguments are INTEGER. In the latter case the returned value is INTEGER. A DIV B is identical to SGN(A/B) × INT(ABS(A/B)).

DOT

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the inner (dot) product of two numeric vectors.



Item	Description/Default	Range Restrictions
vector name	name of a one-dimensional numeric array	any valid name

Example Statements

```
PRINT DOT(A,B)
B=DOT(A,A)
```

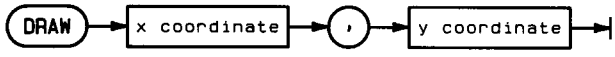
Semantics

The dot product is calculated by multiplying corresponding elements of the two vectors and then summing the products. The two vectors must be the same current size. If both vectors are INTEGER, the product will be an INTEGER. Otherwise, the product will be of type REAL.

DRAW

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a line from the pen's current position to the specified X and Y coordinate position using the current line type and pen number.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression, in current units	—
y coordinate	numeric expression, in current units	—

Example Statements

```
DRAW 10,90
DRAW Next_x,Next_y
```

Semantics

The X and Y coordinate information is interpreted according to the current unit-of-measure. Draw is affected by the PIVOT transformation.

A DRAW to the current position generates a point. DRAW updates the logical pen position at the completion of the DRAW statement, and leaves the pen down on an external plotter. The line is clipped at the current clipping boundary.

If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

DROUND

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function rounds a numeric expression to the specified number of digits. If the specified number of digits is greater than 15, no rounding takes place. If the number of digits specified is less than 1, 0 is returned.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression	—	—
number of digits	numeric expression, rounded to an integer	—	1 thru 15

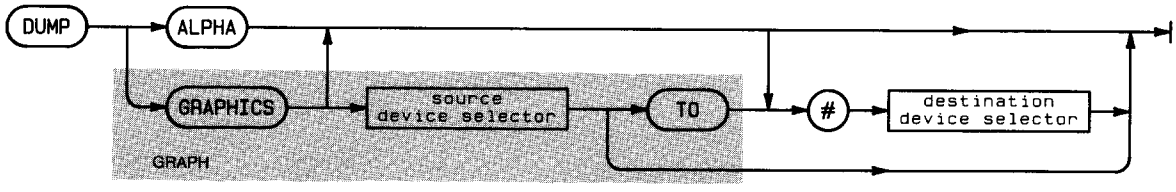
Example Statements

```
Test_real=DROUND(True_real,12)
PRINT "Approx. Volts =";DROUND(Volts,3)
```

DUMP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement copies the contents of an alphanumeric or graphics display to a printing device.



Item	Description/Default	Range Restrictions
source device selector	numeric expression, rounded to an integer; Default = last CRT plotter	see Glossary
destination device selector	numeric expression, rounded to an integer; Default = DUMP DEVICE IS device	external interfaces only (see Glossary)

Example Statements

```
DUMP ALPHA
DUMP GRAPHICS #702
DUMP GRAPHICS 28 TO #702
```

Semantics

DUMP ALPHA copies the contents of the alphanumeric display to a printer. With a bit-mapped display, the alpha buffer is sent to the printer as alphanumeric characters.

DUMP GRAPHICS copies the entire contents of the CRT graphics display, which may contain bit-mapped alpha, to a printer. Performing DUMP GRAPHICS to a device which does not support the HP Raster Interface Standard will produce unpredictable results. The HP 2631G, HP 9876, and the ThinkJet printers are among devices that support this standard.

If the destination device is not explicitly specified, it is assumed to the current DUMP DEVICE IS device.

If EXPANDED is specified in the DUMP DEVICE IS statement, the source graphics image is doubled in both X and Y directions before being sent to the destination device. However, if both source and destination devices are explicitly specified, the image is sent without being expanded.

If a DUMP GRAPHICS operation is stopped by pressing the (CLR I/O) key, the printer may or may not terminate its graphics mode. Sending the printer up to 192 null characters [CHR$(0)] can be used to terminate the graphics mode on a printer such as the HP 9876.

If the source has multiple planes of graphics memory associated with a pixel, an inclusive-OR is performed on all the bits corresponding to the pixel. This determines whether to print it as black or white.

If a currently active CRT is explicitly specified as the source, the CRT's contents are dumped to the printer; however, if the specified CRT has not been activated, error 708 is reported.

Plotters are de-activated by power-up, GINIT, SCRATCH A, or (RESET). A plotting device is activated when it is specified in a PLOTTER IS statement. In addition, the internal CRT is also (implicitly) activated by any of the following operations after de-activation: any pen movement; GCLEAR; GLOAD (to the current default destination); GSTORE (from the current default source); and DUMP GRAPHICS (from the current default source).

If a non-CRT source which is the current PLOTTER IS device is explicitly specified, the DUMP GRAPHICS is not performed; however, if an non-CRT source which is not the current PLOTTER IS device is explicitly specified, error 708 is reported.

On multi-plane bit-wrapped display devices, which use a graphics write-enable mask, only the bits indicated by 1s will be ORed together and dumped.

Displays with Nonsquare Pixels

For machines which have a display with nonsquare pixels (the HP 98542A and the HP 98543A), a non-expanded `DUMP GRAPHICS` will produce an image that matches the CRT *only* if no alpha appears in the graphics planes. Since most printers print square pixels, this routine treats graphics pixel pairs as single elements and prints one square for each pixel pair in the frame buffer. Because alpha works with individual pixels, and not with pixel pairs, mixed alpha and graphics will appear blurred on a `DUMP GRAPHICS` non-expanded output. Using the `EXPANDED` option causes the vertical length (the height on the CRT) to be doubled as before, but dumps each separate pixel. In this mode, mixed alpha and graphics will appear the same on the dump as on the CRT.

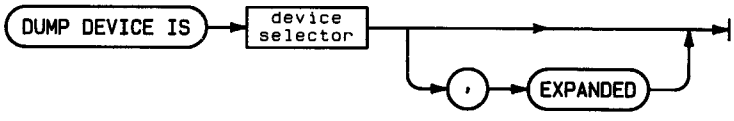
Note

Some printers are not capable of printing 1024 graphics dots per line, so images dumped will be truncated to fit the printer.

DUMP DEVICE IS

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies which device receives the data when either DUMP ALPHA or DUMP GRAPHICS is executed without a device selector.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer; Default = 701	external interfaces only (see Glossary)

Example Statements

```
DUMP DEVICE IS 721
DUMP DEVICE IS Printer,EXPANDED
```

Semantics

Doing a DUMP GRAPHICS to a printer which does not support the HP Raster Interface Standard will produce unpredictable results. The HP 9876 and the HP 2631G are among the devices which support the standard.

Specifying EXPANDED results in graphics dumps that are twice as big on each axis (except for displays with nonsquare pixels – see DUMP GRAPHICS for details) and turned sideways. This gives four dots on the printer for each dot on the display. The resulting picture does not fit on one page of an HP 9876 or HP 2631G printer.

DVAL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a binary, octal, decimal, or hexadecimal character string into a REAL whole number.



Item	Description/Default	Range Restrictions
string argument	string expression, containing digits valid for the specified base	(see tables)
radix	numeric expression, rounded to an integer	2, 8, 10, or 16

Example Statements

```
Address=DVAL("FF590004",16)
Real=DVAL("01010101010101010101010101010101",2)
Number=DVAL(Octal$,8)
```

Semantics

The radix is a numeric expression that will be rounded to an integer and must evaluate to 2, 8, 10, or 16.

The string expression must contain only the characters allowed for the particular number base indicated by the radix. ASCII spaces are not allowed.

Binary strings are presumed to be in two's complement form. If all 32 digits are specified and the leading digit is a 1, the returned value is negative.

Octal strings are presumed to be in the octal representation of two's complement form. If all 11 digits are specified, and the leading digit is a 2 or a 3, the returned value is negative.

Decimal strings containing a leading minus sign will return a negative value.

Hex strings are presumed to be in the hex representation of the two's complement binary form. The letters A through F may be specified in either upper or lower case. If all 8 digits are specified and the leading digit is 8 through F the returned value is negative.

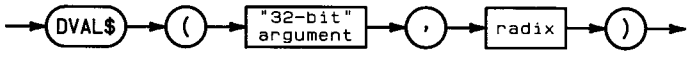
Radix	Base	String Range	String Length
2	binary	0 thru 11111111111111111111111111111111	1 to 32 characters
8	octal	0 thru 37777777777	1 to 11 characters
10	decimal	−2147483648 thru 2147483647	1 to 11 characters
16	hexadecimal	0 thru FFFFFFFF	1 to 8 characters

Radix	Legal Characters	Comments
2	+,0,1	—
8	+,0,1,2,3,4,5,6,7	Range restricts the leading character. Sign, if used, must be a leading character.
10	+,−,0,1,2,3,4,5,6,7,8,9	Sign, if used, must be a leading character.
16	+,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,a,b,c,d,e,f	A/a = 10, B/b = 11, C/c = 12, D/d = 13 E/e = 14, F/f = 15

DVAL$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a whole number into a binary, octal, decimal, or hexadecimal string.



Item	Description/Default	Range Restrictions
"32-bit" argument	numeric expression, rounded to an integer	-2^{31} thru $2^{31}-1$
radix	numeric expression, rounded to an integer	2, 8, 10, or 16

Example Statements

```
F$=DVAL$(-1,16)
Binary$=DVAL$(Count DIV 256,2)
```

Semantics

The rounded argument must be a value that can be expressed (in binary) using 32 bits or less.

The radix must evaluate to be 2, 8, 10, or 16; representing binary, octal, decimal, or hexadecimal notation.

If the radix is 2, the returned string is in two's complement form and contains 32 characters. If the numeric expression is negative, the leading digit will be 1. If the value is zero or positive there will be leading zeros.

If the radix is 8, the returned string is the octal representation of the two's complement binary form and contains 11 digits. Negative values return a leading digit of 2 or 3.

If the radix is 10, the returned string contains 11 characters. Leading zeros are added to the string if necessary. Negative values have a leading minus sign.

If the radix is 16, the returned string is the hexadecimal representation of the two's complement binary form and contains 8 characters. Negative values return with the leading digit in the range 8 thru F.

Radix	Base	Range of Returned String	String Length
2	binary	00000000000000000000000000000000 thru 11111111111111111111111111111111	32 characters
8	octal	00000000000 thru 37777777777	11 characters
10	decimal	−2147483648 thru 2147483647	11 characters
16	hexadecimal	00000000 thru FFFFFFFF	8 characters

ECHO

See the SET ECHO statement.

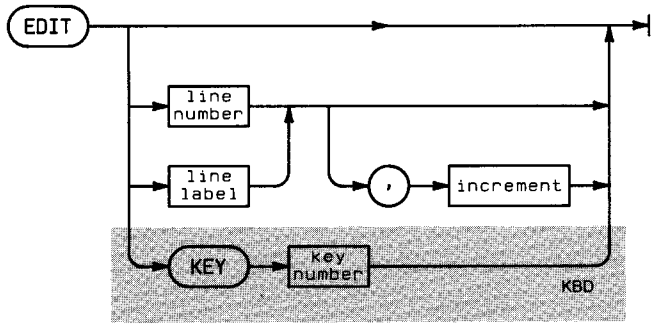
EDGE

See the IPLOT, PLOT, POLYGON, RECTANGLE, RPLOT and SYMBOL statements.

EDIT

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command allows you to enter or edit either a program or typing-aid key definitions.



Item	Description/Default	Range Restrictions
line number	integer constant identifying program line; Default (see Semantics)	1 thru 32 766
line label	name of a program line	any valid name
increment	integer constant; Default = 10	1 thru 32 766
key number	integer constant	0 thru 23

Example Statements

```
EDIT
EDIT 1000,5
EDIT KEY 4
```

Semantics

The EDIT command allows you to scroll through a program in the computer by using the arrow keys, [Prev], [Next], or the knob. Lines may be added to the end of a program by going to the bottom of the program. A new line number will be provided automatically. Lines may be added between existing program lines by using the insert line key, and lines may be deleted by using the delete line key. Lines may be modified by typing the desired characters over the existing line, using the insert character and delete character keys as necessary. [ENTER], [EXECUTE] or [RETURN] are used to store the newly created or modified lines.

Edit mode is exited by pressing [CONTINUE], [CLR SCR], [Clear display], [PAUSE], [Stop] (on HP 46020A), [RESET], [RUN], or [STEP] or by executing CAT, LIST, GET, or LOAD. If the program was changed while paused, pressing [CONTINUE] will generate an error, since modifying a program moves it to the stopped state.

EDIT Without Parameters

If no program is currently in the computer, the edit mode is entered at line 10, and the line numbers are incremented by 10 as each new line is stored. If a program is in the computer, the line at which the editor enters the program is dependent upon recent history. If an error has paused program execution, the editor enters the program at the line flagged by the error message. Otherwise, the editor enters the program at the line most recently edited (or the beginning of the program after a LOAD operation).

EDIT With Parameters

If no program is in the computer, a line number (not a label) must be used to specify the beginning line for the program. The increment will determine the interval between line numbers. If a program is in the computer, any increment provided is not used until lines are added to the end of the program. If the line specified is between two existing lines, the lowest-numbered line greater than the specified line is used. If a line label is used to specify a line, the lowest-numbered line with that label is used. If the label cannot be found, an error is generated.

EDIT KEY (Requires KBD)

To enter the EDIT KEY mode, type `EDIT KEY`, followed by the key number, and press (EXECUTE), (ENTER), or (RETURN). Also, the desired softkey can be pressed after typing or pressing EDIT. When EDIT KEY mode is entered, the current key definition (if any) is displayed. You then edit the contents as if it were any other keyboard line. Non-ASCII keys may be included in the key definition by holding (CTRL) while pressing the desired key. Non-ASCII keystrokes are represented by an inverse-video "K" followed by another character associated with the key. The table *Second Byte of Non-ASCII Key Sequences* in the "Useful Tables" section of this manual has a list of the characters associated with the special keys.

Note

On the HP 98203A keyboard, many non-ASCII keys cannot be accessed by the method of holding (CTRL) while pressing the desired key. However, any of the non-ASCII keys can be entered into a softkey definition by pressing (ANY CHAR) 255, followed by the character associated with that non-ASCII key.

To accept the modified key definition, press (ENTER) or (RETURN); to abort without changing the current definition, press (PAUSE), (CLR SCR), or (Clear display).

When a program is waiting for a response to an INPUT, LINPUT or ENTER, the typing aid definitions (defined with EDIT KEY) are in effect. When a program is running but not waiting for user input, the active ON KEY definitions supercede the typing aid definitions. Softkeys without ON KEY definitions retain their typing-aid function.

ELSE

See the IF...THEN statement.

ENABLE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

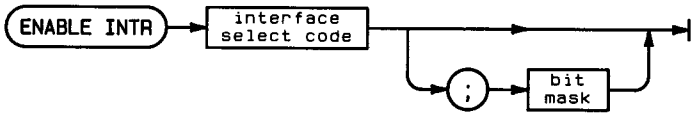
This statement re-enables all event-initiated branches which were suspended by DISABLE. ON END, ON ERROR, and ON TIMEOUT are not affected by ENABLE and DISABLE.



ENABLE INTR

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement enables the specified interface to generate an interrupt which can cause end-of-statement branches.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	5, and 7 thru 31
bit mask	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
ENABLE INTR 7
ENABLE INTR Isc;Mask
```

Semantics

If a bit mask is specified, its value is stored in the interface's interrupt-enable register. Consult the documentation provided with each interface for the correct interpretation of its bit mask values.

If no bit mask is specified, the previous bit mask for the select code is restored. A bit mask of all zeros is used when there is no previous bit mask.

END

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	No

This statement marks the end of the main program. (For information about END as a secondary keyword, see the OUTPUT and SEND statements.)



Semantics

END must be the last statement (other than comments) of a main program. Only one END statement is allowed in a program. (Program execution may also be terminated with a STOP statement, and multiple STOP statements are allowed.) END terminates program execution, stops any event-initiated branches, and clears any unserviced event-initiated branches. CONTINUE is not allowed after an END statement.

Subroutines used by the main program must occur prior to the END statement. Subprograms and user-defined functions must occur after the END statement.

END IF

See the IF...THEN statement.

END LOOP

See the LOOP statement.

END SELECT

See the SELECT...CASE construct.

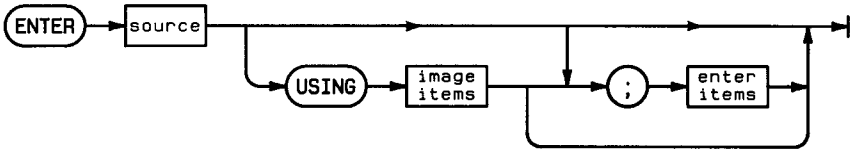
END WHILE

See the WHILE statement.

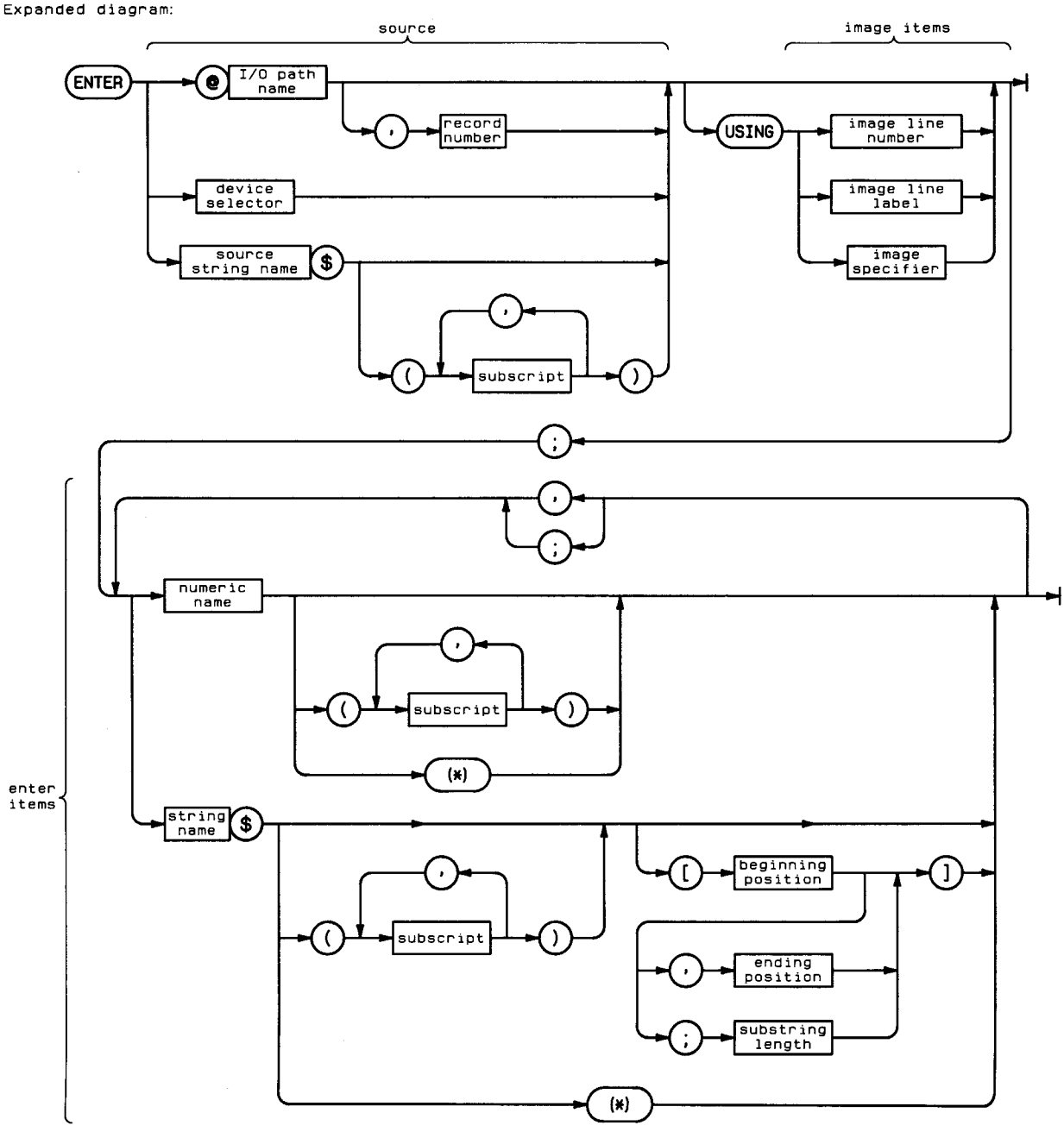
ENTER

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

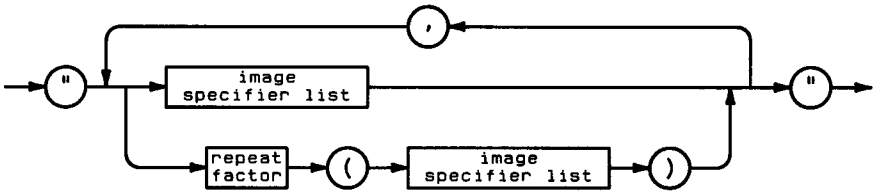
This statement is used to input data from a device, file, string, or buffer and assign the values entered to variables. (If using ENTER with SRM, also refer to the "SRM" section of this manual.)



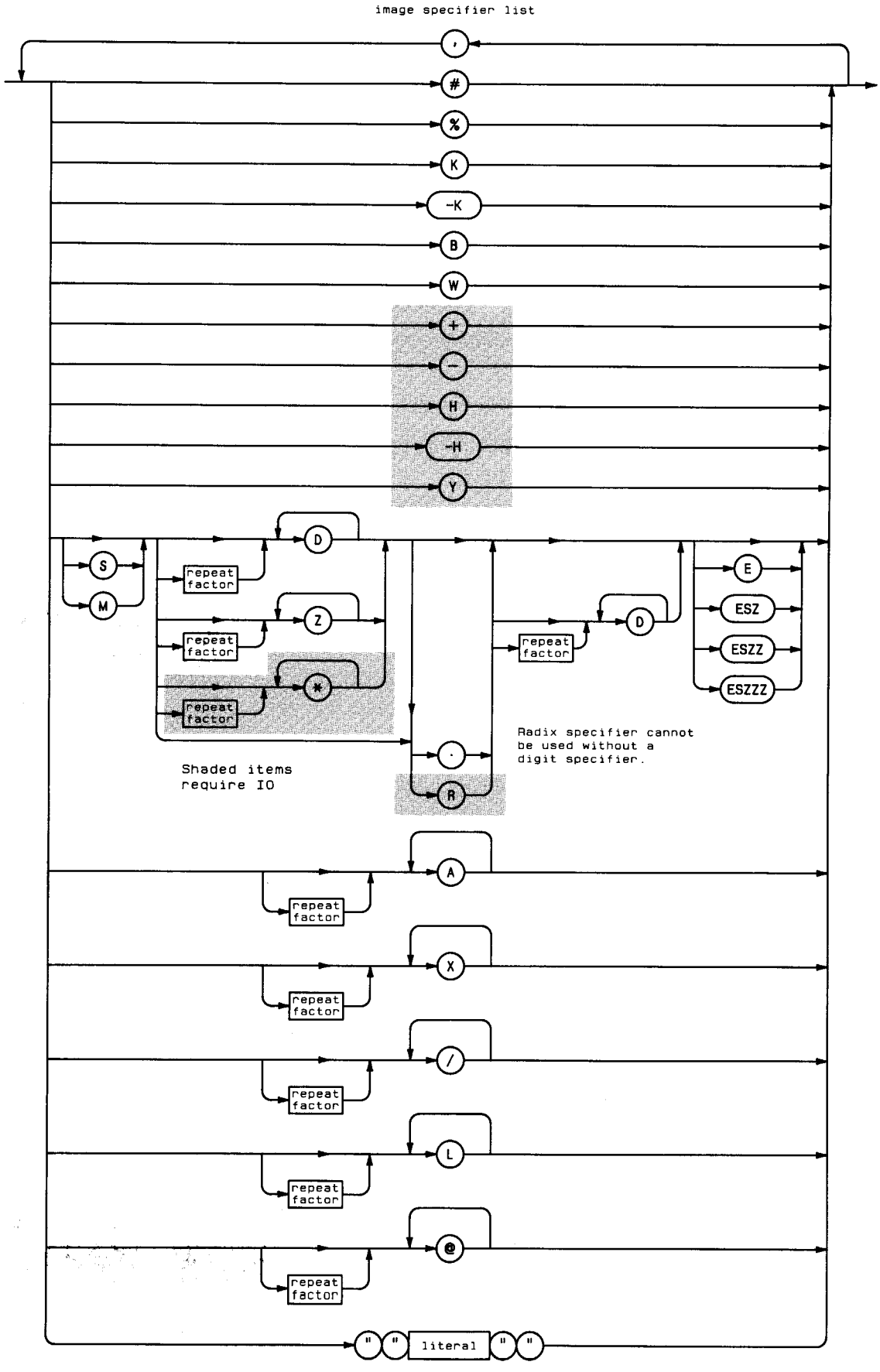
Expanded diagram:



literal form of image specifier:



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, mass storage file, or buffer	any valid name (see ASSIGN)
record number	numeric expression, rounded to an integer	1 thru $2^{31}-1$
device selector	numeric expression, rounded to an integer	(see Glossary)
source string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
image line number	integer constant identifying an IMAGE statement	1 thru 32 766
image line label	name identifying an IMAGE statement	any valid name
image specifier	string expression	(see drawing)
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed


image specifier list
,
#
%
K
-K
B
W
+
-
H
-H
Y
S
M
repeat factor
D
Z
*
E
ESZ
ESZZ
ESZZZ
.
R
Shaded items require IO
Radix specifier cannot be used without a digit specifier.
A
X
/
L
@
" " literal " "

Example Statements

```
ENTER 705;Number,String$
ENTER @File;Array(*)
ENTER @Source USING Fmt5;Item(1),Item(2),Item(3)
ENTER 12 USING "#,6A";A$[2;6]
```

Semantics

The Number Builder

If the data being received is ASCII and the associated variable is numeric, a number builder is used to create a numeric quantity from the ASCII representation. The number builder ignores all leading non-numeric characters, ignores all blanks, and terminates on the first non-numeric character, or the first character received with EOI true. (Numeric characters are 0 thru 9, +, -, decimal point, e, and E, in a meaningful numeric order.) If the number cannot be converted to the type of the associated variable, an error is generated. If more digits are received than can be stored in a variable of type REAL, the rightmost digits are lost, but any exponent will be built correctly. Overflow occurs only if the exponent overflows.

Arrays

Entire arrays may be entered by using the asterisk specifier. Each element in an array is treated as an item by the ENTER statement, as if the elements were listed separately. The array is filled in row major order (rightmost subscript varies fastest.)

Files as Source

If an I/O path has been assigned to a file, the file may be read with ENTER statements. The file must be an ASCII or BDAT file. The attributes specified in the ASSIGN statement are used only if the file is a BDAT file. Data read from an ASCII file is always in ASCII format. Data read from a BDAT file is considered to be in internal format if FORMAT is OFF, and is read as ASCII characters if FORMAT is ON.

Serial access is available for both ASCII and BDAT files. Random access is available for BDAT files. The file pointer is important to both serial and random access. The file pointer is set to the beginning of the file when the file is opened by an ASSIGN. The file pointer always points to the next byte available for ENTER operations.

Random access uses the record number parameter to read items from a specific location in a file. The record specified must be before the end-of-file. The ENTER begins at the beginning of the specified record.

It is recommended that random and serial access to the same file not be mixed. Also, data should be entered into variables of the same type as those used to output it (e.g. string for string, REAL for REAL, etc.).

Devices as Source

An I/O path name or a device selector may be used to ENTER from a device. If a device selector is used, the default system attributes are used (see ASSIGN). If an I/O path name is used, the ASSIGN statement determines the attributes used. If multiple devices were specified in the ASSIGN, the ENTER sets the first device to be talker, and the rest to be listeners.

If FORMAT ON is the current attribute, the items are read as ASCII. If FORMAT OFF is the current attribute, items are read from the device in the computer's internal format. Two bytes are read for each INTEGER, eight bytes for each REAL. Each string entered consists of a four byte header containing the length of the string, followed by the actual string characters. The string must contain an even number of characters; if the length is odd, an extra byte is entered to give alignment on the word boundary.

CRT as Source

If the device selector is 1, the ENTER is from the CRT. The ENTER reads characters from the CRT, beginnning at the current print position (print position may be modified by using TABXY in a PRINT statement.) The print position is updated as the ENTER progresses. After the last non-blank character in each line, a line-feed is sent with a simulated "EOI". After the last line is read, the print position is off the screen. If the print position is off screen when an ENTER is started, the off-screen text is first scrolled into the last line of the display.

Keyboard as Source

ENTER from device selector 2 may be used to read the keyboard. An entry can be terminated by pressing [ENTER], [EXECUTE], [RETURN], [CONTINUE], or [STEP]. Using [ENTER], [EXECUTE], [RETURN] or [STEP] causes a CR/LF to be appended to the entry. The [CONTINUE] key adds no characters to the entry and does not terminated the ENTER statement. If an ENTER is stepped into, it is stepped out of, even if the [CONTINUE] key is pressed. An HP-IB EOI may be simulated by pressing [CTRL] [E] before the character to be sent, if this feature has been enabled by an appropriate CONTROL statement to the keyboard (see the Control and Status Registers in the back of this book).

Strings as Source

If a string name is used as the source, the string is treated similarly to a file. However, there is no file pointer; each ENTER begins at the beginning of the string, and reads serially within the string.

Buffers as Source (Requires TRANS)

When entering from an I/O path assigned to a buffer, data is removed from the buffer beginning at the location indicated by the buffer's empty pointer. As data is received, the current number-of-bytes register and empty pointer are adjusted accordingly. Encountering the fill pointer (buffer empty) produces an error unless a continuous inbound TRANSFER is filling the buffer. In this case, the ENTER will wait until more data is placed in the buffer.

Since devices are logically bound to buffers, an ENTER statement cannot intercept data while it is traveling between the device and the buffer. If an I/O path is currently being used in an outbound TRANSFER, and an ENTER statement uses it as a source, execution of the ENTER is deferred until the completion of the TRANSFER. An ENTER can be concurrent with an inbound TRANSFER only if the source is the I/O path assigned to the buffer.

An ENTER from a string variable that is also a buffer will not update the buffer's pointers and may return meaningless data.

ENTER With USING

When the computer executes an ENTER USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If no variable is required for the field specifier, the field specifier is acted upon without referencing the enter items. When the field specifer references a variable, bytes are entered and used to create a value for the next item in the enter list. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching enter item. If the image specifiers are exhausted before the enter items, the specifiers are reused, starting at the beginning of the specifier list.

Entry into a string variable always terminates when the dimensioned length of the string is reached. If more variables remain in the enter list when this happens, the next character received is associated with the next item in the list.

When USING is specified, all data is interpreted as ASCII characters. FORMAT ON is always assumed with USING, regardless of any attempt to specify FORMAT OFF.

Effects of the image specifiers on the ENTER statement are shown in the following table:

Image Specifier	Meaning
K	Freefield Entry. Numeric: Entered characters are sent to the number builder. Leading non-numeric characters are ignored. All blanks are ignored. Trailing non-numeric characters and characters sent with EOI true are delimiters. Numeric characters include digits, decimal point, +, −, e, and E when their order is meaningful. String: Entered characters are placed in the string. Carriage-return not immediately followed by line-feed is entered into the string. Entry to a string terminates on CR/LF, LF, a character received with EOI true, or when the dimensioned length of the string is reached.
−K	Like K except that LF is entered into a string, and thus CR/LF and LF do not terminate the entry.
H	Like K, except that the European number format is used. Thus, a comma is the radix indicator and a period is a terminator for a numeric item. (Requires IO)
−H	Same as −K for strings; same as H for numbers. (Requires IO)
S	Same action as D.
M	Same action as D.
D	Demands a character. Non-numerics are accepted to fill the character count. Blanks are ignored, other non-numerics are delimiters.
Z	Same action as D.
*	Same action as D. (Requires IO)
.	Same action as D.
R	Like D, R demands a character. When R is used in a numeric image, it directs the number builder to use the European number format. Thus, a comma is the radix indicator and a period is a terminator for the numeric item. (Requires IO)
E	Same action as 4D.
ESZ	Same action as 3D.
ESZZ	Same action as 4D.

Image Specifier	Meaning
ESZZZ	Same action as 5D.
A	Demands a string character. Any character received is placed in the string.
X	Skips a character.
literal	Skips one character for each character in the literal.
B	Demands one byte. The byte becomes a numeric quantity.
W	Demands one 16-bit word, which is interpreted as a 16-bit, two's-complement integer. If either an I/O path name with the BYTE attribute or a device selector is used to access an 8-bit interface, two bytes will be entered; the most-significant byte is entered first. If an I/O path name with the BYTE attribute is used to access a 16-bit interface, the BYTE attribute is overridden and one word is entered in a single operation. If an I/O path name with the WORD attribute is used to access a 16-bit interface, one byte is entered and ignored when necessary to achieve alignment on a word boundary. If the source is a file, string variable, or buffer, the WORD attribute is ignored and all data are entered as bytes; however, one byte will be entered and ignored when necessary to achieve alignment on a word boundary.
Y	Like W, except that pad bytes are never entered to achieve word alignment. If an I/O path name with the BYTE is used to access a 16-bit interface, the BYTE attribute is not overridden (as with W specifier above). (Requires IO)
#	Statement is terminated when the last ENTER item is terminated. EOI and line-feed are item terminators, and early termination is not allowed.
%	Like #, except that an END indication (such as EOI or end-of-file) is an immediate statement terminator. Otherwise, no statement terminator is required. Early termination is allowed if the current item is satisfied.
+	Specifies that an END indication is required with the last character of the last item to terminate the ENTER statement. Line-feeds are not statement terminators. Line-feed is an item terminator unless that function is suppressed by −K or −H. (Requires IO)
−	Specifies that a line-feed terminator is required as the last character of the last item to terminate the statement. EOI is ignored, and other END indications, such as EOF or end-of-data, cause an error if encountered before the line-feed. (Requires IO)
/	Demands a new field; skips all characters to the next line-feed. EOI is ignored.
L	Ignored for ENTER.
@	Ignored for ENTER.

ENTER Statement Termination
A simple ENTER statement (one without USING) expects to give values to all the variables in the enter list and then receive a statement terminator. A statement terminator is an EOI, a line-feed received at the end of the last variable (or within 256 characters after the end of the last variable), an end-of-data indication, or an end-of-file. If a statement terminator is received before all the variables are satisfied, or no terminator is received within 256 bytes after the last variable is satisfied, an error occurs. The terminator requirements can be altered by using images.

An ENTER statement with USING, but without a % or # image specifier, is different from a simple ENTER in one respect. EOI is not treated as a statement terminator unless it occurs on or after the last variable. Thus, EOI is treated like a line-feed and can be used to terminate entry into each variable.

An ENTER statement with USING that specifies a # image requires no statement terminator other than a satisfied enter list. EOI and line feed end the entry into individual variables. The ENTER statement terminates when the variable list has been satisfied.

An ENTER statement with USING that specifies a % image allows EOI as a statement terminator. Like the # specifier, no special terminator is required. Unlike the # specifier, if an EOI is received, it is treated as an immediate statement terminator. If the EOI occurs at a normal boundary between items, the ENTER statement terminates without error and leaves the value of any remaining variables unchanged.

EOL

See the ASSIGN, PRINTALL IS, and PRINTER IS statements.

EOR

See the OFF EOR, ON EOR, and TRANSFER statements.

EOT

See the OFF EOT and ON EOT statements.

ERRDS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an INTEGER representing the device selector of the I/O resource involved in the most recent I/O error.



Example Statements

```
IF ERRDS=701 THEN GOSUB Printer_fault
IF ERRN=163 THEN Missing=ERRDS
```

Semantics

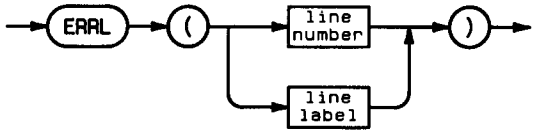
The device selector will include a primary address if the interface addressed allows it (i.e. HP-IB). If the resource is a file, the device specifier of the drive containing the file is returned. If the resource is not a device, 0 is returned. If no I/O error has occured in a running program since power-up, SCRATCH A, or pre-run, 0 is returned.

If an error occurs in a TRANSFER statement without WAIT, the error number is recorded in the assignment table associated with the I/O path name assigned to the non-buffer end of the transfer instead of being reported immediately. It is not reported until the next reference to the I/O path name, and ERRDS will not be updated until this time.

ERRL

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value of 1 if the most recent error occurred in the specified line; otherwise, a value of 0 is returned.



Item	Description/Default	Range Restrictions
line number	integer constant	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
IF ERRL(220) THEN Parse_error
IF NOT ERRL(Parameters) THEN Other
```

Semantics

The specified line must be in the same context as the ERRL function, or an error will occur.

If an error occurs in a TRANSFER statement without WAIT, the error number is recorded in the assignment table associated with the non-buffer end of the transfer instead of being reported immediately. It is not reported until the next reference to the I/O path name, and ERRL will not be updated until this time. Therefore, ERRL will actually refer to the line containing the new reference to the I/O path name, not the line containing the TRANSFER statement that caused the error.

Data Communications

This function returns 0 for all Data Communications errors.

ERRM$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the text of the error message associated with the most recent program execution error.



Example Statements

```
PRINT ERRM$
Em$=ERRM$
ENTER Em$;Error_number,Error_line
```

Semantics

If no error has occurred since power on, prerun, SCRATCH, SCRATCH A, LOAD, or GET, the null string will be returned. The line number and error number returned in the ERRM$ string are the same as those used by ERRN and ERRL, which may be surprising when a TRANSFER is in effect. For details on the interaction, see ERRL and ERRN.

ERRN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the number of the most recent program execution error. If no error has occurred, a value of 0 is returned.



Example Statements

```
IF ERRN=80 THEN Disc_out
DISP "Error Number";ERRN
```

Semantics

If an error occurs in a TRANSFER statement without WAIT, the error number is recorded in the assignment table associated with the non-buffer end of the transfer instead of being reported immediately. It is not reported until the next reference to the I/O path name, and ERRN will not be updated until this time.

ERROR

See the OFF ERROR and ON ERROR statements.

EXIT IF

See the LOOP statement.

EXOR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based on the logical exclusive-or of its arguments.



Example Statements

```
Ok=First_pass EXOR Old_data
IF A EXOR Flag THEN Exit
```

Semantics

A non-zero value (positive or negative) is treated as a logical 1; only a zero is treated as a logical 0.

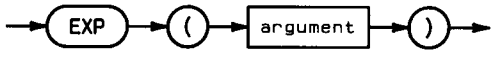
The EXOR function is summarized in this table.

A	B	A EXOR B
0	0	0
0	1	1
1	0	1
1	1	0

EXP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function raises e to the power of the argument. In the computer, Napierian e ≈ 2.718 281 828 459 05.



Item	Description/Default	Range Restrictions
argument	numeric expression	−708.396 418 532 264 thru +709.782 712 893 383 8

Example Statements

```
Y=EXP(-X^2/2)
PRINT "e to the";Z;"=";EXP(Z)
```

EXPANDED

See the DUMP DEVICE IS statement.

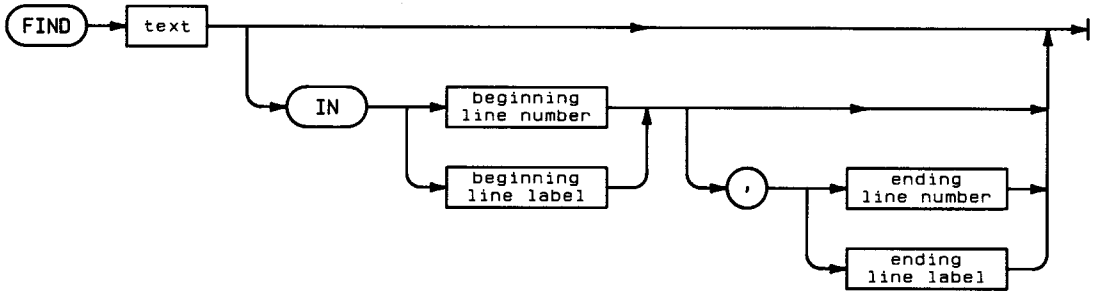
FILL

See the IPLOT, PLOT, POLYGON, RECTANGLE, RPLOT, and SYMBOL statements.

FIND

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command allows you to find a character sequence while editing a program.



Item	Description/Default	Range Restrictions
text	literal	—
beginning line number	integer constant identifying program line	1 to 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line	1 to 32 766
ending line label	name of a program line	any valid name

Example Statements

```
FIND "SUB Print"
FIND "Cost=" IN 250,Label1
FIND "Interval" IN 1550
```

Semantics

This command causes a search to be made through the program currently in memory. It compares the specified text to an internal "listing" of the program. Therefore, line numbers, keywords, variables, and constants can be found.

If an occurrence of the specified text is found, the line containing it is displayed with the cursor under the first character of that occurrence. The line can be modified or deleted if desired. If ENTER, RETURN or the delete line key is pressed, the search resumes with the next character. Alternately, the search is resumed without modifying the program when CONTINUE is pressed. Note that overlapping occurrences will not be detected; e.g, if you were looking for "issi", only one occurrence would be found in "Mississippi".

If the Beginning Line Number is given, the search commences at that line number. If the specified line number doesn't exist, the next line that does exist is used. If the Beginning Line Number is not specified, then the search begins at the line currently being edited; or, (if you're not in edit mode), with the first line of the program. If a specified label doesn't exist, an error occurs.

The search continues through the last character of the Ending Line; or (if that was not-specified) the end of the program. If you specify an Ending Line Number that does not exist, the highest-numbered line which occurs before that line number is used.

If there were no occurrences found, the cursor is left at the end of the first line searched. If one or more occurrences were found, the cursor is left at the end of the line containing the last occurrence.

A FIND command is cancelled by entering a line after having changed its line number. Other keys which will cancel a FIND are [EXECUTE], [CLR I/O], [BREAK], [↑], [↓], or [INS LN]. Any of the keys which cancel EDIT mode will also cancel a FIND.

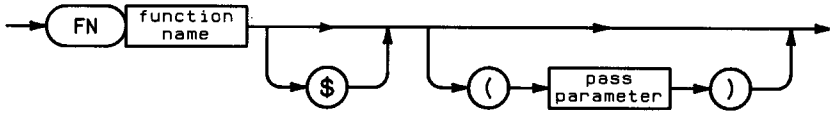
FIND is not allowed while a program is running; however, it may be executed while a program is paused. The program is continuable if it has not been altered by pressing [ENTER], [RETURN], [EXECUTE] or [DEL LN].

While in the FIND mode, keyboard execution is only possible with the [EXECUTE] key. Using [ENTER] or [RETURN] causes an error.

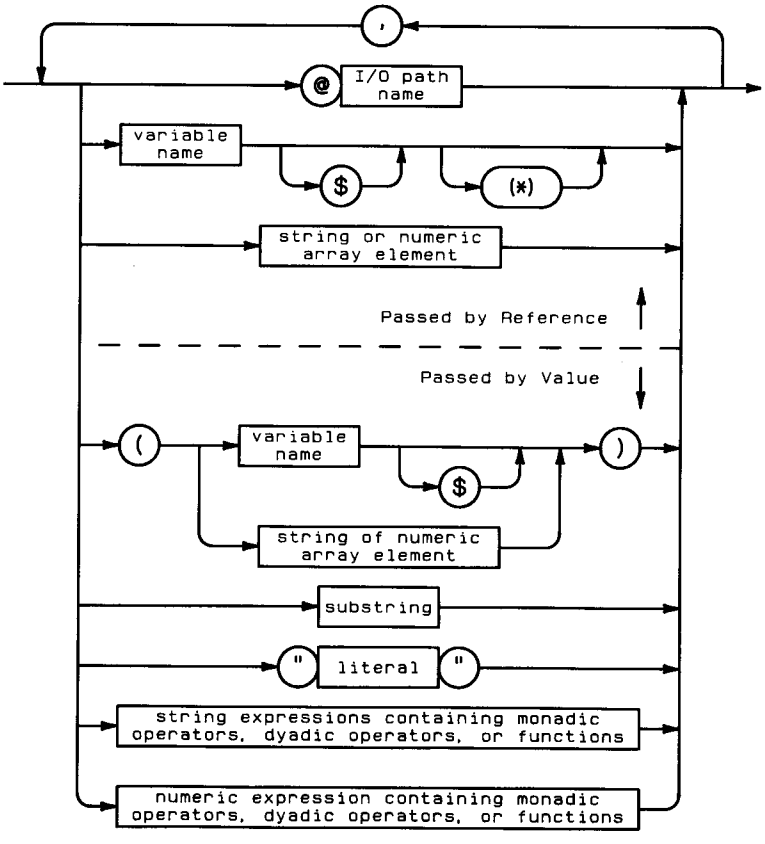
FN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This keyword transfers program execution to the specified user-defined function and may pass items to the function. The value returned by the function is used in place of the function call when evaluating the statement containing the function call.



pass parameters:



Item	Description/Default	Range Restrictions
function name	name of a user-defined function	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
variable name	name of a numeric or string variable	any valid name
substring	string expression containing substring notation	(see Glossary)
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—

Example Statements

```
PRINT X;FNChange(X)
Final$=FNTrim$(First$)
Result=FNPround(Item,Power)
```

Semantics

A user-defined function may be invoked as part of a stored program line or as part of a statement executed from the keyboard. If the function name is typed and then EXECUTE, ENTER or RETURN is pressed, the value returned by the function is displayed. The dollar sign suffix indicates that the returned value will be a string. User-defined functions are created with the DEF FN statement.

The pass parameters must be of the same type (numeric or string) as the corresponding parameters in the DEF FN statement. Numeric values passed by value are converted to the numeric type (REAL or INTEGER) of the corresponding formal parameter. Variables passed by reference must match the type of the corresponding parameter in the DEF FN statement exactly. An entire array may be passed by reference by using the asterisk specifier.

Invoking a user-defined function changes the program context. The functions may be invoked recursively.

If there is more than one user-defined function with the same name, the lowest numbered one is invoked by FN.

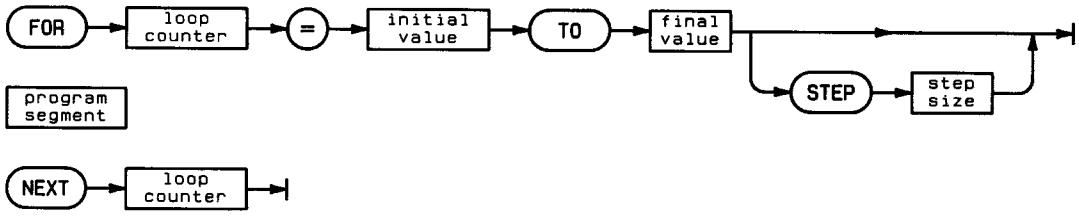
FNEND

See the DEF FN statement.

FOR...NEXT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the loop counter passes a specific value. The step size may be positive or negative.



Item	Description/Default	Range Restrictions
loop counter	name of a numeric variable	any valid name
initial value	numeric expression	—
final value	numeric expression	—
step size	numeric expression; Default = 1	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
100  FOR I=4 TO 0 STEP -.1
110    PRINT I;SQR(I)
120  NEXT I

1220  INTEGER Point
1230  FOR Point=1 TO LEN(A$)
1240    CALL Convert(A$[Point;1])
1250  NEXT Point
```

Semantics

The loop counter is set equal to the initial value when the loop is entered. Each time the corresponding NEXT statement is encountered, the step size (which defaults to 1) is added to the loop counter, and the new value is tested against the final value. If the final value has not been passed, the loop is executed again, beginning with the line immediately following the FOR statement. If the final value has been passed, program execution continues at the line following the NEXT statement. Note that the loop counter is not equal to the specified final value when the loop is exited.

The loop counter is also tested against the final value as soon as the values are assigned when the loop is first entered. If the loop counter has already passed the final value in the direction the step would be going, the loop is not executed at all. The loop may be exited arbitrarily (such as with a GOTO), in which case the loop counter has whatever value it had obtained at the time the loop was exited.

The initial, final and step size values are calculated when the loop is entered and are used while the loop is repeating. If a variable or expression is used for any of these values, its value may be changed after entering the loop without affecting how many times the loop is repeated. However, changing the value of the loop counter itself can affect how many times the loop is repeated.

The loop counter variable is allowed in expressions that determine the initial, final, or step size values. The previous value of the loop counter is not changed until after the initial, final, and step size values are calculated.

If the step value evaluates to 0, the loop repeats infinitely and no error is given.

Nesting Constructs Properly
Each FOR statement is allowed one and only one matching NEXT statement. The NEXT statement must be in the same context as the FOR statement. FOR...NEXT loops may be nested, and may be contained in other constructs, as long as the loops and constructs are properly nested and do not improperly overlap.

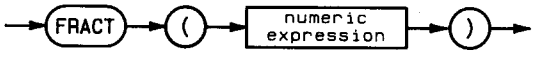
FORMAT

See the ASSIGN statement.

FRACT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a number greater than or equal to zero and less than 1, representing the "fractional part" of the value of its argument. For all X, X = INT(X) + FRACT(X).



Example Statements

```
PRINT FRACT(X)
Right_digits=FRACT(All_digits)
```

FRAME

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a frame around the current clipping area using the current pen number and line type. After drawing the frame, the current pen position coincides with the lower left corner of the frame, and the pen is up.



FRENCH

See the LEXICAL ORDER IS statement.

GCLEAR

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement clears the graphics display or sends a command to an external plotter to advance the paper. With bit-mapped displays, the memory is cleared and the alpha is restored.



Multi-Plane Bit-Mapped Displays

The GCLEAR statement clears all planes designated as graphics planes with the current graphics write-mask. This includes any planes which are both alpha and graphics planes. See the "Multi-Plane Bit-Mapped Displays" section in the *Graphics Techniques* manual for information on enabling and displaying specific frame buffer planes.

Note

If any planes in the frame buffer are enabled by *both* the alpha mask and the graphics mask, the common planes, as well as the graphics planes, will be cleared. Then, the alpha data will be redisplayed in the common planes. This may cause text which was previously hidden or overwritten by graphics to reappear.

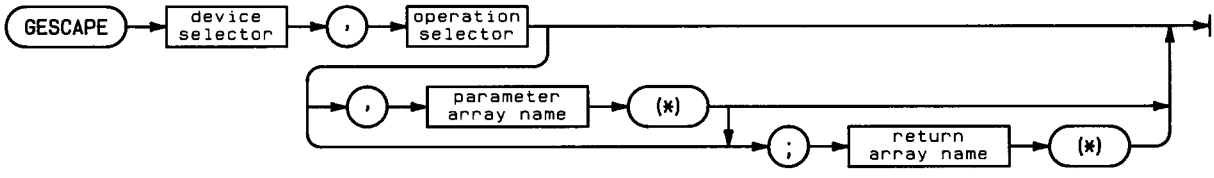
GERMAN

See the LEXICAL ORDER IS statement.

GESCAPE

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used for communicating device-dependent information.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
operation selector	numeric expression, rounded to an integer	(device dependent, see Semantics)
parameter array name	name of array which has a specific rank and size, containing parameters necessary for executing request	any valid name
return array name	name of array which has a specific rank and size into which the returned parameters are placed	any valid name

Example Statements

```
GESCAPE 28,5                (Selects alternate drawing mode)
GESCAPE 3,2;Color_map(*)    (Get the values in the color map)
```

Semantics

The parameter array and return array are for sending data to the device and getting data from the device, respectively. The use of the parameter array is currently unimplemented and is reserved for future use.

Color Map Information

The number of entries in the color map can be determined with a GESCAPE operation selector 1. The return array must be one-dimensional with at least one element.

The RGB values of the pens in the the color map can be obtained through GESCAPE operation selector 2. The return array must be a two-dimensional three-column array with at least one row. The values returned are in the range on 0 to 1 and are multiples of 1/15 (one fifteenth). The first row in the array always contains the values for PEN 0; if you want PEN 12, you must have at least thirteen rows in the array. Array filling occurs until either the array or the color map is exhausted.

Determining Hard Clip Limits and GSTORE Array Size

The hard clip limits of the current plotting device can be obtained through executing a GESCAPE with operation selector 3. The return array must be a one-dimensional INTEGER array with at least four elements. Values will be returned in the smallest resolvable units for that device. For a CRT, units are pixels.

Operation selector 3 also returns information useful for GSTORE and GLOAD files. The fifth and sixth elements returned give the two array dimensions to use (in conjunction with the ALLOCATE statement) to GSTORE the contents of the specified display. For example, on a HP 98544A display with all planes enabled for graphics, the dimensions returned would be 256 and 400–256 words for each of the 400 lines. That is, 1024 pixels wide, and four pixels' worth of information in each 16-bit word. This allows the user to programmatically determine the size of the integer array to allocate for storing an image and thus avoid machine-dependent code.

Drawing Mode Dominance

The normal drawing mode and the alternate drawing mode can be entered by using GESCAPE operation selectors 4 and 5, respectively. Drawing in normal mode "covers up" any previously-drawn image. Drawing in alternate mode with positive pen numbers causes the color-map entry number at each pixel to be inclusively-ORed with the pen value currently being drawn with. Drawing in alternate mode with negative pen numbers causes the color-map entry number at each pixel to be exclusively-ORed with the pen value currently being drawn with. Drawing in alternate mode with negative pen numbers causes the color-map entry number at each pixel to be exclusively-ORed with the pen value currently being drawn with.

Multi-Plane Bit-Mapped Displays

The Write-Enable and Display-Enable Masks

If you have a multi-plane frame buffer and display, there are two user-definable masks which control certain aspects of graphical operations. They are:

- The *write-enable mask*. This mask is an integer whose bits, from the least-significant bit end, designate those frame buffer planes which will be affected by graphics operations. Bit values of 1 denote enabled planes (planes to be written to), and bit values of 0 denote disabled planes (planes which will not be written to). For example, if you have a four-plane frame buffer, and you set the write-enable mask to 3 (binary 0011), only values in frame buffer planes 1 and 2 will be modified by graphical operations.
- The *display-enable mask*. This mask is an integer whose bits, from the least-significant bit end, designate those frame buffer planes which are to be displayed. These bits may or may not indicate the same planes as the write-enable mask indicates. That is, you can write to some planes, and display others. Bit values of 1 denote planes which are to be displayed, and bit values of 0 denote planes which are not to be displayed. For example, if you have a four-plane frame buffer, and you set the write-enable mask to 5 (binary 0101), only values in frame buffer planes 1 and 3 will be displayed.

NOTE

Both the write-enable mask and the display-enable mask are initialized to all planes that exist in the machine at power up and SCRATCH A time.

Operation selector 6, which works with all CRTs, allows the user to obtain the current graphics write-enable and display-enable values. The first element of the return array contains the write-enable mask; the second represents the display-enable mask. The return array must be a one-dimensional integer array with at least one element. Array filling occurs until either the array or the masks are exhausted.

Operation selector 7, which works only with multi-plane Series 300 CRTs, allows the user to set the graphics write-enable and display-enable values. The first element of the parameter array contains the write-enable mask; the second represents the display-enable mask. Again, the parameter array must be a one-dimensional integer array with one or more elements. If only one element exists, the write-enable mask is set as specified and the display-enable mask remains unchanged.

Legal values for both masks are:

- 0 through 15 for 4-plane systems,
- 0 through 255 for 8-plane systems.

Absolute Locator Hard Clip Limits

Operation selector 20 sets the hardclip limits for absolute HP-HIL locators. That is, it simulates, in software, the changing of the hardclip limits. These limits must be inside the largest X and largest Y, taken individually, for all absolute locators on the HP-HIL bus.

Operation selector 21 returns the current hardclip limits for absolute HP-HIL locators. These are the values used in `GRAPHICS INPUT IS` scaling. Operation selector 21 is different than operation selector 22 in that 22 always returns the values "hardwired" into the device(s) on the HP-HIL bus, whereas the values returned by operation selector 21 may have come from operation selector 20 or from the device on the bus.

Operation selector 22 returns the hardware-defined hardclip limits of all absolute locators on the HP-HIL bus.

For the three `GESCAPE` selectors above–20, 21, and 22–the parameter array must be a one-dimensional integer array. Only the first two entries will be used for 20 and 21: X2 and Y2. No space is taken for the X1 and Y1 values, since the coordinates of P1 (the lower, left-hand corner) cannot be changed on HP-HIL absolute locators; X1 and Y1 will always be zeroes. For operation selector 22, entries will be made until the array is full or all devices on the bus have been covered. If more array entries exist after the devices are all represented, a -1 will be put in what would be the X coordinate entry of the next device to indicate the end of valid data. (Hardclip limits for these devices are limited to the range 0 through 32 767.)

Unlike other `GESCAPE`s, selectors 20 through 22 do not require the device at the specified select code to be currently active. Indeed, to be effective, `GESCAPE 2,20`, which sets hard clip limits, must be done before doing the `GRAPHICS INPUT IS KBD, "TABLET"` statement. Operations 20 and 21 will give "`DEVICE NOT PRESENT`" errors if no tablet (or HP-HIL interface) exists, but 22 will return -1 for its first entry in that case. All will give a configuration error if the `KBD` binary is not present.

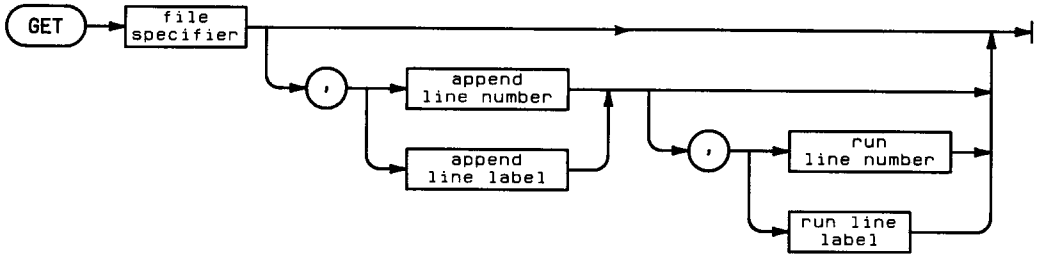
Functions Available Through GESCAPE

Operation Selector	Return Array	
1	A(0): Number of entries in the color map	Color-Mapped Graphic Devices
2	A(0,0): Pen 0 red color map value A(0,1): Pen 0 green color map value A(0,2): Pen 0 blue color map value : : A(15,0): Pen 15 red color map value A(15,1): Pen 15 green color map value A(15,2): Pen 15 blue color map value	Color-Mapped Graphic Devices
3	A(0): X minimum hard clip value A(1): Y minimum hard clip value A(2): X maximum hard clip value A(3): Y maximum hard clip value	All Graphics Devices
	A(4): Rows required for GSTORE integer array A(5): Columns required for GSTORE integer array	All CRTs
4	Set normal drawing mode	All Color CRT Graphics Devices
5	Set alternate drawing mode	All Color CRT Graphics Devices
6	A(0): Current graphics write-enable mask value A(1): Current graphics display-enable mask value	Series 300 Displays
7	A(0): Graphics write-enable mask value to be set A(1): Graphics display-enable mask value to be set	Series 300 Displays
20	A(0): X maximum hard clip value to be set A(1): Y maximum hard clip value to be set	HP-HIL Locators
21	A(0): Current X maximum hard clip value A(1): Current Y maximum hard clip value	HP-HIL Locators
22	A(0): X maximum hard clip value for first absolute locator A(1): Y maximum hard clip value for first absolute locator A(2): X maximum hard clip value for second absolute locator A(3): Y maximum hard clip value for second absolute locator : : A(*n*): A value of −1 indicates that there are no more absolute locators	HP-HIL Locators

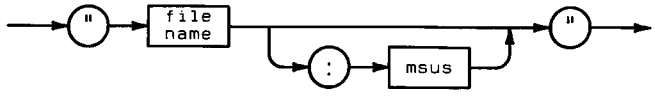
GET

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement reads the specified ASCII file and attempts to store the strings into memory as program lines. (If using GET with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
append line number	integer constant identifying a program line	1 thru 32 766
append line label	name of a program line	any valid name
run line number	integer constant identifying a program line	1 thru 32 766
run line label	name of a program line	any valid name
file name	literal	any valid file name
msus	literal	(see MASS STORAGE IS)

Example Statements

```
GET "George"
GET Next_prog$,180,10
```

Semantics

When GET is executed, the first line in the specified file is read and checked for a valid line number. If no valid line number is found, the current program stays in memory and error 68 is generated. If the GET was attempted from a running program, the program remains active and the error 68 can be trapped with ON ERROR. If there is no ON ERROR in effect, the program pauses.

If there is a valid line number at the start of the first line in the file, the GET operation proceeds. Values for all variables except those in COM are lost and the current program is deleted from the append line to the end. If no append line is specified, the entire current program is deleted.

As the file is brought in, each line is checked for proper syntax. The syntax checking during GET is the same as if the lines were being typed from the keyboard, and any errors that would occur during keyboard entry will also occur during GET. Any lines which contain syntax errors are listed on the PRINTER IS device. Those erroneous lines which have valid line numbers are converted into comments and syntax is checked again. If the GET encounters a line longer than 256 characters, the operation is terminated and error 128 is reported. If any line caused any other syntax error, an error 68 is reported at the completion of the GET operation. This error is not trappable because the old program was deleted and the new one is not running yet.

Any line in the main program or any subprogram may be used for the append location. If an append line number is specified, the lines from the file are renumbered by adding an offset to their line numbers. This offset is the difference between the append line number and the first line number in the file. This operation preserves the line-number intervals that exist in the file. When a line containing an error (or an invalid line number caused by renumbering) is printed on the PRINTER IS device, the line number shown is the one the line had in the file. Any programmed references to line numbers that would be renumbered by REN are also renumbered by GET. If no append line is specified, the lines from the file are entered without renumbering.

If a successful GET is executed from a program, execution resumes automatically after a prerun initialization (see RUN). If no run line is specified, execution resumes at the lowest-numbered line in the program. If a run line is specified, execution resumes at the specified line. The specified run line must be a line in the main program segment.

If a successful GET is executed from the keyboard **and** a run line is specified, a prerun is performed and program execution begins automatically at the specified line. If GET is executed from the keyboard with no run line specified, RUN must be executed to start the program. GET is not allowed from the keyboard while a program is running.

GINIT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement establishes a set of default values for variables affecting graphics operations.



Semantics

The following operations are performed when GINIT is executed:

```
AREA PEN 1
CLIP OFF
CSIZE 5,0.6
LDIR 0
LINE TYPE 1,5
LORG 1
MOVE 0,0
PDIR 0
PEN 1
PIVOT 0
GESCAPE CRT,4 (PEN MODE NORMAL)
VIEWPORT 0,RATIO*100,0,100
WINDOW 0,RATIO*100,0,100
```

In addition an active plotter or graphics input device is terminated. If the plotter is a file, the file is closed.

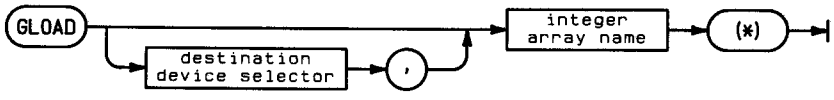
After a GINIT and before a PLOTTER IS statement is executed, the following statements select a default plotter.

AXES
DRAW
DUMP GRAPHICS
FRAME
GCLEAR
GLOAD
GRID
GSTORE
IDRAW
IMOVE
IPLOT
LABEL
MOVE
PLOT
POLYGON
POLYLINE
RECTANGLE
RPLOT
SET ECHO
SET PEN
SYMBOL

GLOAD

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement loads the contents of an INTEGER array into a frame buffer (the converse of GSTORE).



Item	Description/Default	Range Restrictions
destination device selector	numeric expression, rounded to an integer: Default = last CRT plotter	(see Glossary)
integer array name	name of an INTEGER array.	any valid name

Example Statements

```
GLOAD Picture(*)
IF Flag THEN GLOAD Array(*)
GLOAD CRT,Screen(*)
GLOAD 28,Screen(*)
```

Semantics

A frame buffer is an area of memory which contains the digital representation of a raster image. A monochromatic image has a frame buffer of one bit deep. The Model 236 color monitor has a four-bit frame buffer which allows sixteen colors. The HP 98627A external color interface has a three-bit frame buffer which allows eight colors. The 98543A and 98545A display boards have 4 planes, allowing 16 colors, and the 98700 has 4 or 8 planes, allowing 16 or 256 colors, respectively.

If a destination device is not explicitly specified, the array's contents are loaded into the current PLOTTER IS device (if it is a frame buffer) or into the last frame buffer device specified by a PLOTTER IS statement.

GLOAD operates on active plotting devices. A plotting device is active when it is specified in a PLOTTER IS statement. In addition, the internal CRT is also activated by any of the following operations: any pen movement; GCLEAR; GLOAD to the current default destination; GSTORE from the current default source; DUMP GRAPHICS from the current default source; and SET PEN. Plotters are de-activated by power-up, GINIT, SCRATCH A or (RESET).

The array's contents are loaded into the specified frame buffer if a currently active frame buffer (CRT) is explicitly specified as the destination. However, if the specified frame buffer is not activated, error 708 occurs.

The GLOAD is not performed if a non-frame buffer destination which is the current PLOTTER IS device is explicitly specified. However, if a non-frame buffer destination which is **not** the current PLOTTER IS device is specified, error 708 occurs.

Pixel Representation
A pixel is a picture element. Each pixel on a monochromatic display is represented by one bit in memory; a binary 1 represents a pixel that is on, while a binary 0 represents a pixel which is off. Each INTEGER array element represents 16 pixels on a monochromatic display.

Pixels on color displays have different representation. The Model 236 color display requires four bits to represent each pixel. The optional color monitor (HP 98627) requires three bits to represent each pixel.

The number of pixels on the horizontal and vertical axes and the number of INTEGER array elements necessary to represent the entire display is shown in the following table for each model and display.

Model	Horizontal Size	Vertical Size	INTEGER Elements
216 (HP 9816) (monochromatic)	400	300	7500
220 (HP 9920) (HP 98204A) (HP 98204B) (monochromatic)	400	300	7500
	512	390	12 480
226 (HP 9826) (monochromatic)	400	300	7500
HP 98627A (external color)	512	512	49 152
236 (HP 9836) (monochromatic)	512	390	12 480
236 (HP 9836C) (color)	512	390	49 920
237 (HP 9837) (HP 98781A) (bit-mapped, monochromatic)	1024	768	49 152
35731A (medium-resolution bit-mapped, monochromatic)	1024	400	25 600
35741A(medium-resolution bit-mapped, color, 4 planes)	1024	400	102 400
98781A (high-resolution bit-mapped, monochromatic)	1024	768	49152
98782A (high-resolution bit-mapped, color, 4 planes)	1024	768	19 6608
98700 (high-resolution bit-mapped, color, 8 planes)	1024	768	393 216

The declared array size can be larger or smaller than the graphics memory size; the operation stops when either graphics memory or the array is exhausted.

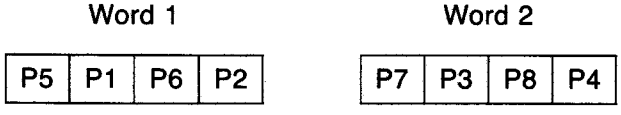
Since any one dimension of an array cannot be more than 32 767 elements, for an array to be large enough to hold the entire graphics representation, the array may have to be multi-dimensional. For example,

```
INTEGER Screen(1:390,1:64,1:2)  !for Model 236 Color
INTEGER Screen(1:512,1:32,1:3)  !for HP 98627A Color
```

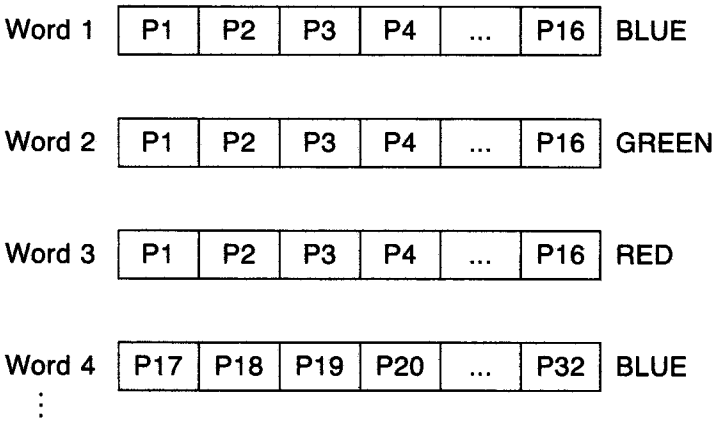
Storage Format
The pixel representation on a monochromatic display are stored sequentially in the array using GSTORE.

The pixel representation for color displays are stored in different formats using GSTORE.

Model 236 color display: Consecutive pairs of 16-bit words are used, regardless of the array structure. P in the diagram is the 4-bit representation of the pixel.



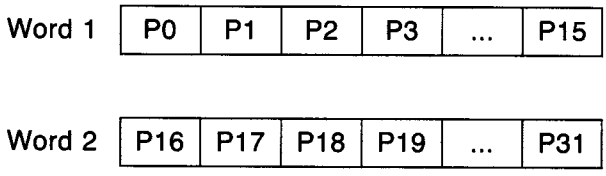
HP 98627A color display: Each word contains the blue, green or red representation for 16 pixels. P in the diagram is the 1-bit color representation of the pixel.



Storage Format on Multi-Plane Bit-Mapped Displays
GLOAD loads information from an array into the *graphics planes* in the frame buffer. "Graphics planes" means those planes which have been write-enabled for graphics operations via power-up, SCRATCH A, or GESCAPE. You can change the graphics write mask with GESCAPE.

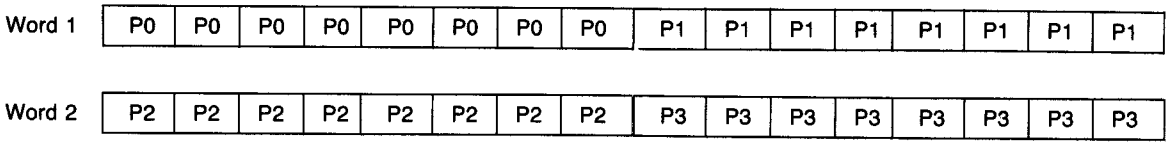
In the following paragraphs, reference is made to the "highest graphics plane." The "highest graphics plane" is that plane in the frame buffer whose corresponding bit in the graphics write-enable mask has the highest number. For example, the highest graphics plane with a write mask of binary 1000 is 4. Also note that although bits in a byte are numbered from 0 through 7 (right to left), planes in the frame buffer are numbered 1 through 8.

If the highest graphics plane currently enabled is 1 (or none), act like there is 1. The storage format is:



If the highest graphics plane currently enabled is between 2 and 4, inclusive, act like there are 4. The storage format is the same as the Model 236C format, described above.

If the highest graphics plane currently enabled is between 5 and 8, inclusive, act like there are 8. The storage format is:



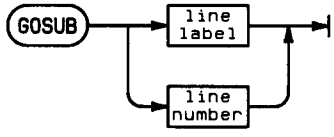
Images should be GLOADed on the same display and with the same write-enable mask that was used when the image was GSTOREd. If these guidelines are not observed, the GLOADed image may bear no resemblance to the GSTOREd image.

To determine the number of elements needed in an integer array the right size to hold an image, use the GESCAPE operation selector 3.

GOSUB

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement transfers program execution to the subroutine at the specified line. The specified line must be in the current context. The current program line is remembered in anticipation of returning (see RETURN). (Also see the ON... statements.)



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766

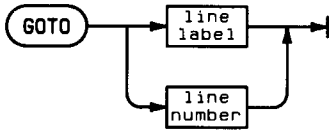
Example Statements

```
GOSUB 120
IF Numbers THEN GOSUB Process
```

GOTO

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to the specified line. The specified line must be in the current context. (Also see the ON... statements.)



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766

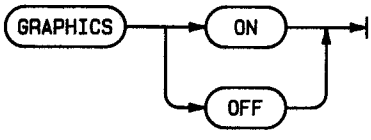
Example Statements

```
GOTO 550
GOTO Loop_start
IF Full THEN Exit
```

GRAPHICS

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement turns the graphics display on or off. This statement has no effect on the contents of the graphics memory, it just controls whether it is displayed or not. At power-on or after SCRATCH A, the graphics display is off. (Also see DUMP.)



Example Statements

```
GRAPHICS ON
IF Flag THEN GRAPHICS OFF
```

Semantics

Multi-Plane Bit-Mapped Displays

If you do not understand the concept of write-enable masks or display-enable masks, see GCLEAR before reading the following paragraph.

GRAPHICS ON/OFF applies only to the graphics display which also is the alpha display. For example, suppose your configuration consists of a display which has both alpha *and* graphics, and another display which has only graphics. In this case, there would be no way, with the GRAPHICS statement, to turn graphics on or off on the display which has graphics exclusively.

With default alpha and graphics write-masks, the GRAPHICS ON and GRAPHICS OFF statements have no effect on bit-mapped displays. If designated alpha and graphics write masks do not overlap, then the statements will enable/disable graphics planes for displaying as with non-bit-mapped systems. When the write masks overlap, planes that are used *only* for graphics (not alpha) are enabled/disabled. For example, if the alpha write-enable mask is binary 1110 and the graphics write-enable mask is binary 0011, GRAPHICS ON and GRAPHICS OFF would only affect plane 1. Plane 2 is not affected because it is indicated by *both* the alpha and graphics write-enable masks, and planes 3 and 4 are not affected because they are not indicated by the graphics write-enable mask.

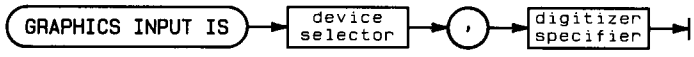
Note

Mixing ALPHA/GRAPHICS ON/OFF with explicit definition of the display-enable mask may cause the (ALPHA) and/or (GRAPHICS) keys to have unexpected results. The reason for this is that explicit setting of the display mask is, in a manner of speaking, working "behind the back" of the operating system. Thus, you could turn off graphics by modifying the display-enable mask, and the internal variables which keep track of (ALPHA) and (GRAPHICS) keypresses would not–indeed, *could* not–have been updated. The reason these variables cannot be updated is that you can set the display mask to a state in which "alpha on" is only partially true; some alpha planes are on, and some aren't. The same goes for graphics.

GRAPHICS INPUT IS

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement defines which device is to be used for graphics input in subsequent DIGITIZE, SET LOCATOR, TRACK IS...ON/OFF, and READ LOCATOR statements.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
digitizer specifier	string expression	(see semantics)

Example Statements

```
GRAPHICS INPUT IS 706,"HPGL"
GRAPHICS INPUT IS Ds,Hp$
GRAPHICS INPUT IS KBD,"KBD"
GRAPHICS INPUT IS KBD,"TABLET"
```

Semantics

The specified device is defined to be the graphics input device for subsequent graphics input statements (DIGITIZE, READ LOCATOR, SET LOCATOR, and TRACK...IS ON). This input device becomes undefined when a power-up, (RESET), GINIT, or SCRATCH A is executed. The default input device is KBD, "KBD".

The operating system attempts to use the current VIEWPORT and WINDOW (or SHOW) parameters for both the current PLOTTER IS device and the specified GRAPHICS INPUT IS device, so that the usable areas of the input and output devices correspond in a 1-to-1 mapping. If the aspect ratios of the input and output devices are different, the input device limits are truncated to match the output device's aspect ratio.

If the VIEWPORT statement specifies an area that does not exist on the input device, error 705 will be reported.

If you specify the keyboard device selector, there are two possibilities for the digitizer specifier. To specify relative pointing devices (e.g., the cursor keys, knob, or mouse), use "KBD" or "ARROW KEYS". For a port path to the Series 500, use the string "ARROW KEYS". To specify absolute pointing devices (e.g., HP-HIL tablets or the TouchScreen), use the string "TABLET". "HPGL" must be specified if the device selector is anything other than the keyboard select code.

When doing a DIGITIZE, the arrow keys and the knob move the graphics cursor. Otherwise, *in addition to moving the graphics cursor*, they perform their normal "alpha" functions: scrolling text on the screen, and moving the alpha cursor within the keyboard entry line.

HP-HIL Absolute Locators

This statement can specify HP-HIL absolute locators, which include graphics tablets as well as the Touchscreen. As with relative locators, all devices of this type are lumped together and processed as if they were a single device. This could lead to interference if two or more of these devices were connected to the HP-HIL bus. The intent is to support *one* active absolute locator on the HP-HIL bus, although careful use will allow more than one. In particular, the GESCAPE values of 20, 21, and 22 allow use of the HP-HIL Touchscreen on the same bus as a Tablet, provided the stylus is removed from the Tablet when the Touchscreen is in use.

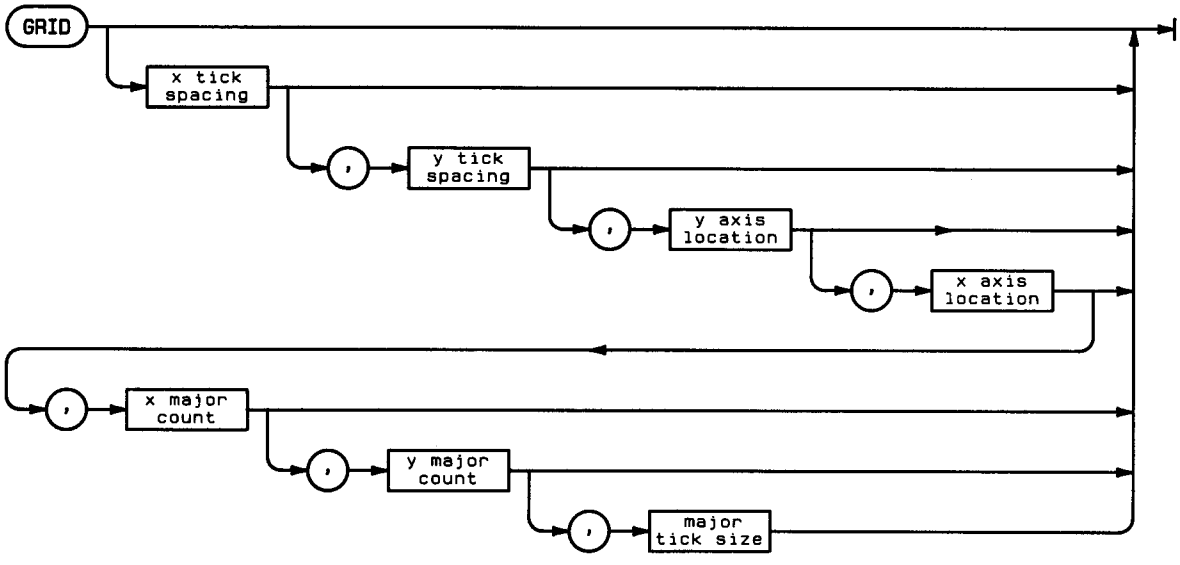
Absolute Locator Hard Clip Limits

You can set the position of P2–the upper right corner of the digitizing area–on HP-HIL tablets by using GESCAPE with operation selectors 20 through 22. This is conceptually similar to setting the P2 point with HPGL commands on HPGL tablets. See GESCAPEfor further information.

GRID

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a full grid pattern. The pen is left at the intersection of the X and Y axes.



Item	Description/Default	Range Restrictions
x tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y axis location	numeric expression specifying the location of the y axis in x-axis units; Default = 0	—
x axis location	numeric expression specifying the location of the x axis in y-axis units; Default = 0	—
x major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
y major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
major tick size	numeric expression in graphic display units; Default = 2	—

Example Statements

```
GRID 10,10,0,0
GRID Xmin,Ymin,Xintercept,Yintercept,5,5
```

Semantics

Grids are drawn with the current line type and pen number. Major tick marks are drawn as lines across the entire soft clipping area. A cross tick is drawn at the intersection of minor tick marks.

The X and Y tick spacing must not generate more than 32 768 grid marks in the clip area, or error 20 will be generated. Only the grid marks within the current clip area are drawn.

Applicable Graphics Transformations

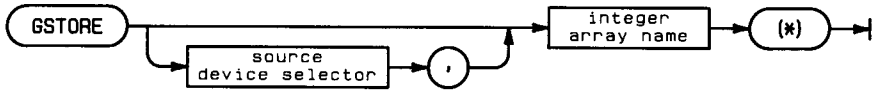
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

GSTORE

Option Required	GRAPH
Keyboard Executabel	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement stores the contents of the frame buffer into an INTEGER array (the converse of GLOAD).



Item	Description/Default	Range Restrictions
source device selector	numeric expression, rounded to an integer; Default = last CRT plotter	(see Glossary)
integer array name	name of an INTEGER array)	any valid name

Example Statements

```
GSTORE Screen(*)
IF Done THEN GSTORE 28,Picture(*)
```

Semantics

A frame buffer is an area of memory which contains the digital representation of a raster image. A monochromatic image has a frame buffer of one bit deep. The Model 236 color monitor has a four-bit frame buffer which allows sixteen colors. The HP 98627A external color monitor has a three-bit frame buffer which allows eight colors. The 98543A and 98545A display boards have 4 planes, allowing 16 colors, and the 98700 has 4 or 8 planes, allowing 16 or 256 colors, respectively.

If a source device is not explicitly specified, the array's contents are loaded from the current PLOTTER IS device (if it is a frame buffer) or from the last frame buffer device specified by a PLOTTER IS statement.

GSTORE operates on active plotting devices. A plotting device is active when it is specified in a PLOTTER IS statement. In addition, the internal CRT is also activated by any of the following operations: any pen movement; GCLEAR; GLOAD to the current default destination; GSTORE from the current default source; DUMP GRAPHICS from the current default source; and SET PEN. Plotters are de-activated by power-up, GINIT, SCRATCH A or (RESET).

The frame buffer's contents are loaded into the specified array if a currently active frame buffer (CRT) is explicitly specified as the source. However, if the specified frame buffer is not activated, error 708 occurs.

The GSTORE is not performed if a non-frame buffer source which is the current PLOTTER IS device is explicitly specified. However, if a non-frame buffer source which is **not** the current PLOTTER IS device is specified, error 708 occurs.

Pixel Representation
A pixel is a picture element. Each pixel on a monochromatic display is represented by one bit in memory; a binary 1 represents a pixel that is on, while a binary 0 represents a pixel which is off. Each INTEGER array element represents 16 pixels on a monochromatic display.

Pixels on color displays have different representation. The Model 236 color display requires four bits to represent each pixel. The optional color monitor (HP 98627) requires three bits to represent each pixel.

The number of pixels on the horizontal and vertical axes and the number of INTEGER array elements necessary to represent the entire display is shown in the following table for each model and display.

Model	Horizontal Size	Vertical Size	INTEGER Elements
216 (HP 9816) (monochromatic)	400	300	7500
220 (HP 9920) (HP 98204A) (HP 98204B) (monochromatic)	400	300	7500
	512	390	12 480
226 (HP 9826) (monochromatic)	400	300	7500
HP 98627A (external color)	512	512	49 152
236 (HP 9836) (monochromatic)	512	390	12 480
236 (HP 9836C) (color)	512	390	49 920
237 (HP 9837) (HP 98781A) (bit-mapped, monochromatic)	1024	768	49 152
35731A (medium-resolution bit-mapped, monochromatic)	1024	400	25 600
35741A(medium-resolution bit-mapped, color, 4 planes)	1024	400	102 400
98781A (high-resolution bit-mapped, monochromatic)	1024	768	49152
98782A (high-resolution bit-mapped, color, 4 planes)	1024	768	19 6608
98700 (high-resolution bit-mapped, color, 8 planes)	1024	768	393 216

The declared array size can be larger or smaller than the graphics memory size; the operation stops when either graphics memory or the array is exhaused.

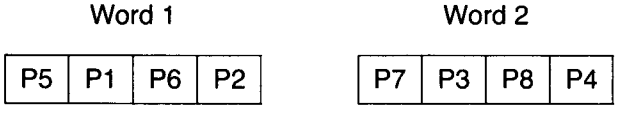
Since any one dimension of an array cannot be more than 32 767 elements, for an array to be large enough to hold the entire graphics representation, the array may have to be multi-dimensional. For example,

```
INTEGER Screen(1:340,1:64,1:2)  !for Model 236 Color
INTEGER Screen(1:512,1:32,1:3)  !for HP 98627A Color
```

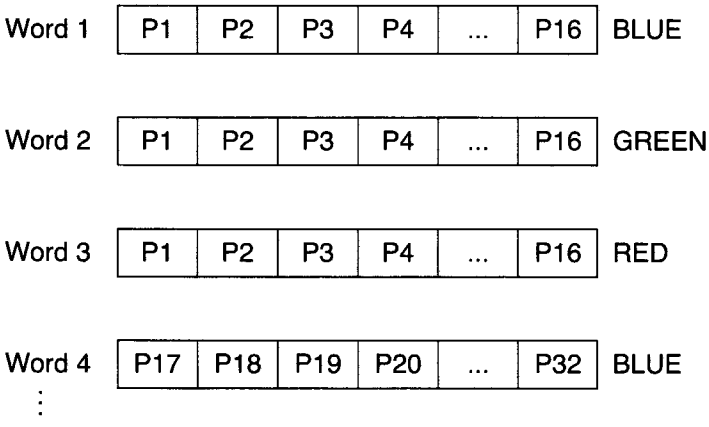
Storage Format
The pixel representation on a monochromatic display are stored sequentially in the array using GSTORE.

The pixel representation for color displays are stored in different formats using GSTORE.

Model 236 color display: Consecutive pairs of 16-bit words are used, regardless of the array structure. P in the diagram is the 4-bit representation of the pixel.



HP 98627A color display: Each word contains the blue, green or red representation for 16 pixels. P in the diagram is the 1-bit color representation of the pixel.

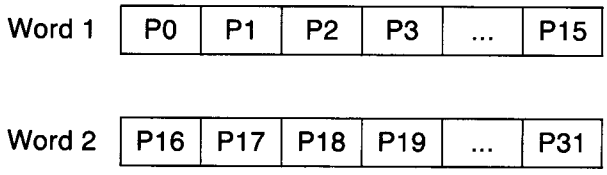


Multi-Plane Bit-Mapped Displays

GSTORE stores information from the *graphics planes* in the frame buffer into an array. "Graphics planes" means those planes which have been write-enabled for graphics via powerup, SCRATCH A, or GESCAPE.

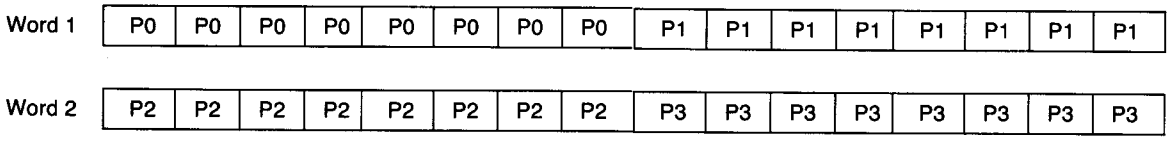
In the following paragraphs, reference is made to the "highest graphics plane." The "highest graphics plane" is that plane in the frame buffer whose corresponding bit in the graphics write-enable mask has the highest number. For example, the highest graphics plane with a write mask of binary 1000 is 4. Also note that although bits in a byte are numbered from 0 through 7 (right to left), planes in the frame buffer are numbered 1 through 8.

If the highest graphics plane currently enabled is 1 (or none), act like there is 1. The storage format is:



If the highest graphics plane currently enabled is between 2 and 4, inclusive, act like there are 4. The storage format is the same as the Model 236C format, described above.

If the highest graphics plane currently enabled is between 5 and 8, inclusive, act like there are 8. The storage format is:



Images should be GLOADed on the same display and with the same write-enable mask that was used when the image was GSTOREd. If these guidelines are not observed, the GLOADed image may bear no resemblance to the GSTOREd image.

To determine the number of elements needed in an integer array the right size to hold an image, use the GESCAPE operation selector 3.

When using graphics and alpha write masks, you may prefer not to overlap the masks; that is, have any planes which are simultaneously indicated by both masks. If planes enabled for alpha overlap those enabled for graphics, some alpha information will be stored along with the graphics information.

You can conserve space if you are using fewer than the maximum number of planes. For example, on a 98700 with eight planes, if pens 0 through 15 *only* are being used, the graphics write mask could be set to 15 (binary 00001111) rather than the default of 255 (binary 11111111). In this way, only half the memory would be required to GSTORE the image. You can change the graphics write mask with GESCAPE.

Non-Square Pixel Displays

With nonsquare-pixel displays, GSTORE will store all pixels (e.g., all 1024×400 pixels), thus requiring over twice the amount of memory as with a Model 236C. This is to insure that any image GSTOREd will appear exactly the same when GLOADed back into the frame buffer. Since alpha uses the nonsquare pixels as separate elements–not as pairs as in graphics–it is possible to have pixel pairs with different values in each pixel. If pixel *pairs* were stored, images with mixed alpha and graphics could appear blurred when reloaded.

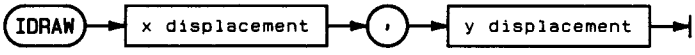
IDN

See the MAT statement.

IDRAW

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a line from the current pen position to a position calculated by adding the X and Y displacements to the current pen position.



Item	Description/Default	Range Restrictions
x displacement	numeric expression in current units	—
y displacement	numeric expression in current units	—

Example Statements

```
IDRAW X+50,0
IDRAW Delta_x,Delta_y
```

Semantics

The X and Y displacement information is interpreted according to the current unit-of-measure.

The line is clipped at the current clipping boundary.

An `IDRAW 0,0` generates a point. IDRAW updates the logical pen position at the completion of the IDRAW statement, and leaves the pen down on an external plotter. IDRAW is affected by the PIVOT transformations.

If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

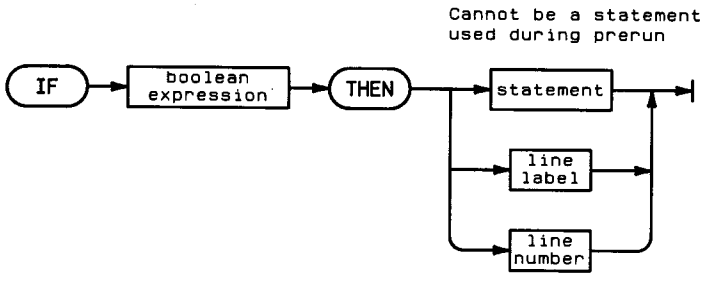
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

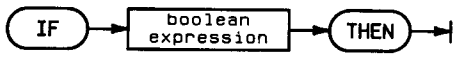
Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

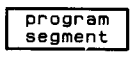
IF...THEN

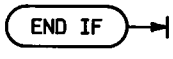
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

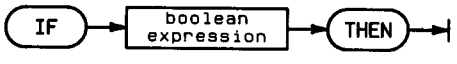
This statement provides conditional branching.

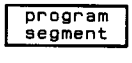


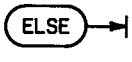


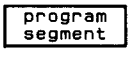














Item	Description/Default	Range Restrictions
boolean expresion	numeric expression; evaluated as true if non-zero and false if zero	—
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
statement	a programmable statement	(see following list)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Program Segments

```
150   IF Flag THEN Next_file
160   IF Pointer<1 THEN Pointer=1

580   IF First_pass THEN
590     Flag=0
600     INPUT "Command?",Cmd$
610     IF LEN(Cmd$) THEN GOSUB Parse
620   END IF

1000   IF X<0 THEN
1010     BEEP
1020     DISP "Improper Argument"
1030   ELSE
1040     Root=SQR(X)
1050   END IF
```

Semantics

If the boolean expression evaluates to 0, it is considered false; if the evaluation is non-zero, it is considered true. Note that a boolean expression can be constructed with numeric or string expressions separated by relational operators, as well as with a numeric expression.

Single Line IF...THEN

If the conditional statement is a GOTO, execution is transferred to the specified line. The specified line must exist in the current context. A line number or line label by itself is considered an implied GOTO. For any other statement, the statement is executed, then program execution resumes at the line following the IF...THEN statement. If the tested condition is false, program execution resumes at the line following the IF...THEN statement, and the conditional statement is not executed.

Prohibited Statements

The following statements must be identified at prerun time or are not executed during normal program flow. Therefore, they are not allowed as the statement in a single line IF...THEN construct.

CASE	END IF	IF	REM
CASE ELSE	END LOOP	IMAGE	REPEAT
COM	END SELECT	INTEGER	SELECT
DATA	END WHILE	LOOP	SUB
DEF FN	EXIT IF	NEXT	SUBEND
DIM	FNEND	OPTION BASE	UNTIL
ELSE	FOR	REAL	WHILE
END			

When ELSE is specified, only one of the program segments will be executed. When the condition is true, the segment between IF...THEN and ELSE is executed. When the condition is false, the segment between ELSE and END IF is executed. In either case, when the construct is exited, program execution continues with the statement after the END IF.

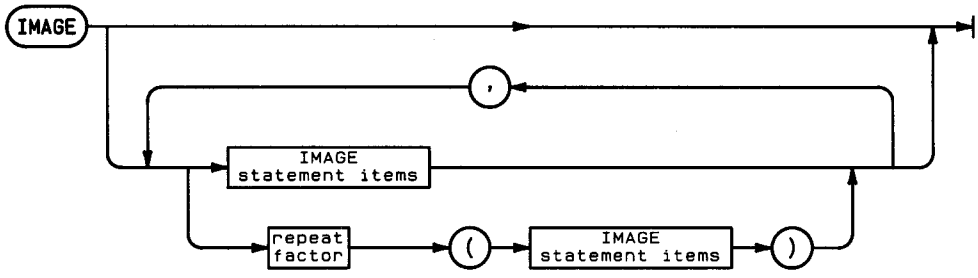
Branching into an IF...THEN construct (such as with a GOTO) results in a branch to the program line following the END IF when the ELSE statement is executed.

The prohibited statements listed above are allowed in multiple-line IF...THEN constructs. However, these statements are not executed conditionally. The exceptions are other IF...THEN statements or constructs such as FOR...NEXT, REPEAT...UNTIL, etc. These are executed conditionally, but need to be properly nested. To be properly nested, the entire construct must be contained in one program segment (see drawing).

IMAGE

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

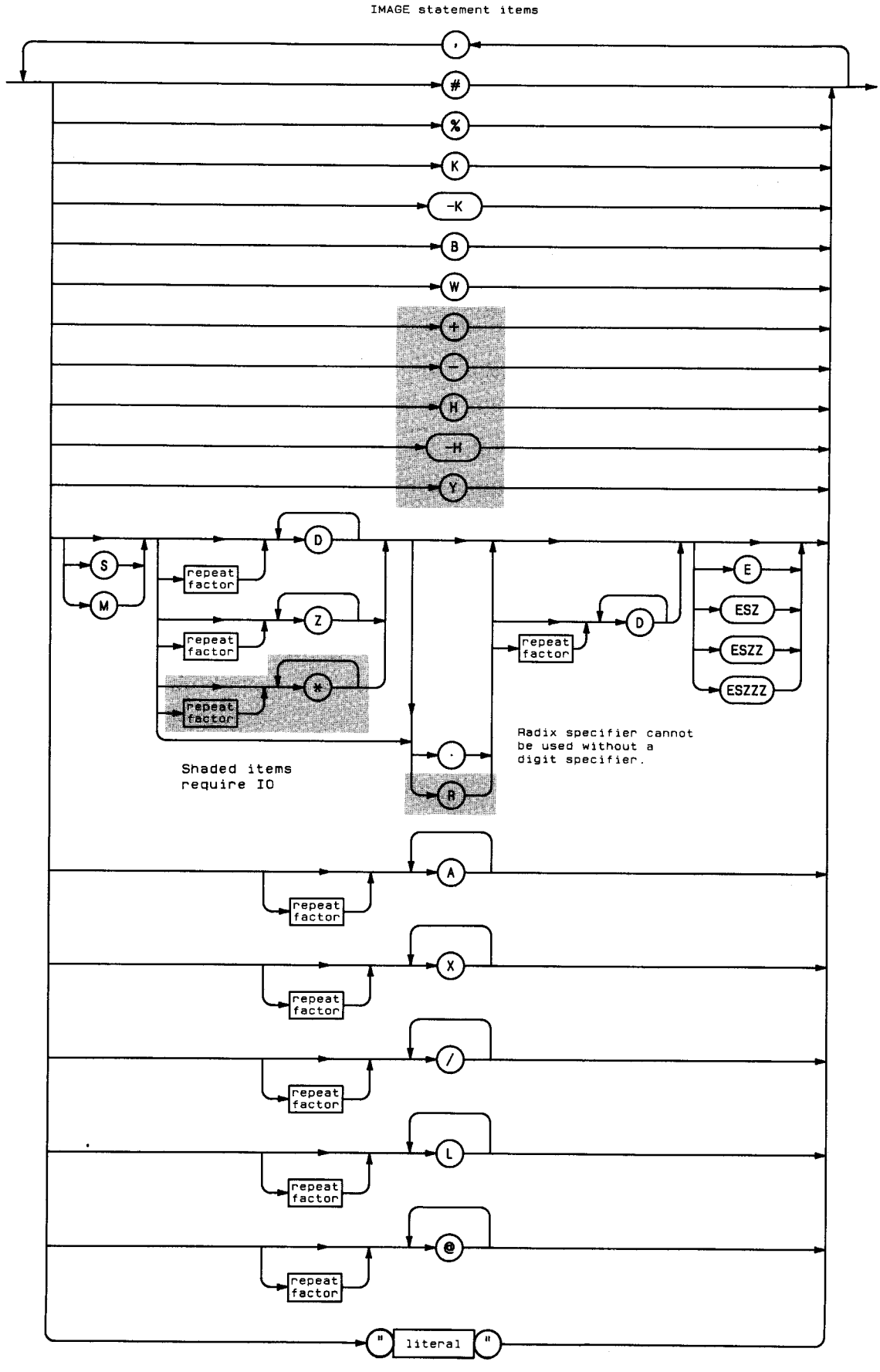
This statement provides image specifiers for the ENTER, OUTPUT, DISP, LABEL, and PRINT statements. Refer to the appropriate statement for details on the effect of the various image specifiers.



Item	Description/Default	Range Restrictions
IMAGE statement items	literal	(see drawing)
repeat factor	integer constant	1 thru 32 767
literal	string composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

Example Statements

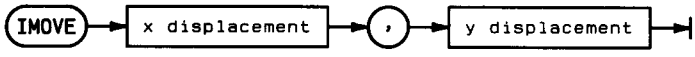
```
IMAGE 4Z,DD,3X,K,/
IMAGE "Result = ",SDDDE,3(XX,ZZ)
IMAGE #,B
```


IMAGE statement items
,
#
%
K
-K
B
W
+
-
H
-H
Y
S
M
repeat factor
D
Z
*
E
ESZ
ESZZ
ESZZZ
Shaded items require IO
Radix specifier cannot be used without a digit specifier.
.
R
A
X
/
L
@
" literal "

IMOVE

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lifts the pen and moves it from the current pen position to a position calculated by adding the specified X and Y displacements to the current pen position.



Item	Description/Default	Range Restrictions
x displacement	numeric expression in current units	—
y displacement	numeric expression in current units	—

Example Statements

```
IMOVE X+50,0
IMOVE Delta_x,Delta_y
```

Semantics

The X and Y displacements are interpreted according to the current unit-of-measure. IMOVE is affected by the PIVOT transformation.

If both current physical pen position and specified pen position are outside current clip limits, no physical pen movement is made; however, the logical pen is moved the specified displacement.

Applicable Graphics Transformations

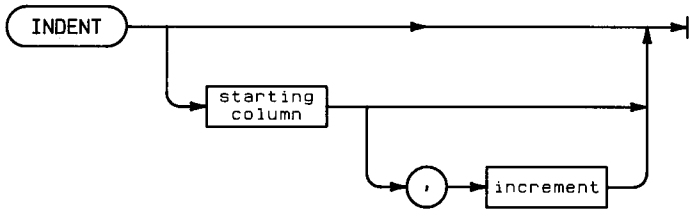
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

INDENT

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This commands indents your program to reflect the structure that results from its constructs.



Item	Description/Default	Range Restrictions
starting column	integer constant; Default = 6	0 thru Screen Width – 8
increment	integer constant; Default = 2	0 thru Screen Width – 8

Example Statements

```
INDENT
INDENT 8,4
```

Semantics

The starting column specifies the column in which the first character of the first statement of each context appears. The increment specifies the number of spaces that the beginning of the lines move to the left or right when the nesting level of the program changes. Note that a line label may override the indentation computed for a particular line. The INDENT command does not move comments which start with an exclamation point, but it does move comments starting with REM. However, if a BASIC program line is moved to the right a comment after it may have to be moved to make room for it. In both of these cases (line labels and comments), the text moves only as far as is necessary; no extra blanks are generated.

Indenting a program may cause the length of some of its lines to become longer than the machine can list. This condition is indicated by the presence of an asterisk after the line numbers of the lines which are overlength. If this occurs, the program will run properly, STORE properly and LOAD properly. However, you cannot do a SAVE, then a GET. Doing an INDENT with smaller values will alleviate this problem.

Indentation occurs after the following statements:

FOR	REPEAT
LOOP	WHILE
SUB	SELECT
IF...THEN[1]	DEF FN

1 This is only true for IF...THEN statements where the THEN is followed by an end-of-line or an exclamation point.

The following statements cause a one-line indentation reversal; that is, indentation is reversed for these statements but re-indented immediately after them:

CASE	EXIT IF
CASE ELSE	FNEND
ELSE	SUBEND

Indentation is reversed before the following statements:

END IF	END WHILE
END LOOP	NEXT
END SELECT	UNTIL

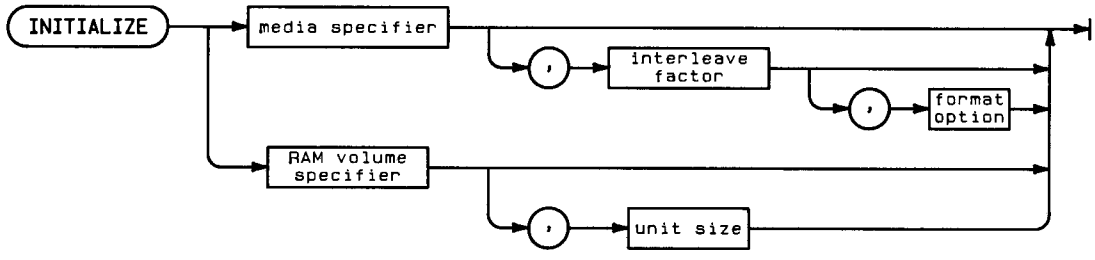
Indentation remains the same from line to line for all other statements.

Improperly matched nesting will cause improper indentation. Deeply nested constructs may cause indentation to exceed Screen Width – 8. However, visible indentation is bounded by Starting Column and Screen Width – 8. If a large Increment is used, indentation may attempt to go beyond Screen Width – 8. This will not be allowed to occur, but an internal indentation counter is maintained, so construct-forming statements will have matching indentation.

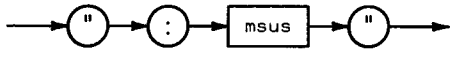
INITIALIZE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

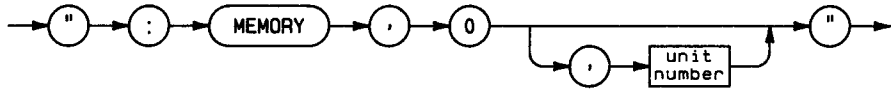
This statement prepares mass storage media for use by the computer. When INITIALIZE is executed, **any data on the media is lost.** (If using INITIALIZE with SRM, also refer to the "SRM" section of this manual.)



literal form of media specifier:



literal form of RAM volume specifier:



Item	Description/Default	Range Restrictions	Recommended Range
media specifier	string expression	(see drawing)	—
interleave factor	numeric expression, rounded to an integer; Default = device dependent (see table)	−32 768 thru +32 767	1 thru 15
format option	numeric expression Default = 0	device dependent	—
RAM volume specifier	string expression	(see drawing)	—
unit size	numeric expression, rounded to an integer; Specifies 256-byte sectors. Default = 1056	4 thru 32 767	memory dependent
msus	literal	(see MASS STORAGE IS)	—

Example Statements

```
INITIALIZE ":INTERNAL"
INITIALIZE Disc$,2
INITIALIZE ":MEMORY,0",Sectors
INITIALIZE ":HP,701",0,4
```

Semantics

Any media used by the computer must be initialized before its **first** use. Initialization rewrites the directory, eliminating any access to old data. The media is partitioned into physical records. The quality of the media is checked during initialization. Defective tracks are "spared" (marked so that they will not be used).

The device type of the internal 5.25 inch disc drive is INTERNAL; the interface select code is 4; the unit number of the right-hand drive is 0; the left-hand drive is 1.

The interleave factor establishes the distance in physical records between consecutively numbered records. The interleave factor is ignored if the mass storage device is not a disc. If you specify 0 for the interleave factor, the default for the device is used.

Device Type	Default Interleave
INTERNAL	1
CS80	see Note
HP9121	2
HP913X (floppy)	4
HP913X (hard)	9
HP9885	1
HP9895	3
HP8290X	4

Note

CS80 discs use the current interleave as the default. If the disc is uninitialized, the interleave recommended for that disc is used. Factory-shipped interleave is 1 for the HP 7908, HP 7911, HP 7912 and HP 7914 discs. An uninitialized HP 9122 disc has a default interleave of 2.

Some mass storage devices allow you to select the format to which the disc is initialized. Omitting the format option or specifying a format option of 0 initializes the disc to its default format. Refer to the disc drive manual for format options available with your disc drive. For example, when initializing a single sided flexible disc on the HP 9122 double sided flexible disc drive use format option 4.

Initializing EPROM (Requires EPROM)

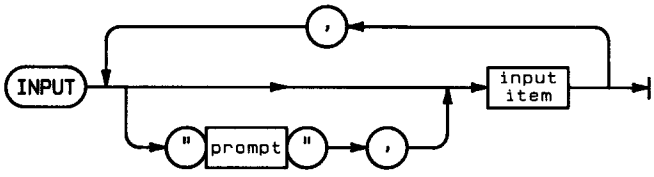
In order to initialize an EPROM unit, it must be completely erased. The select code specified in the INITIALIZE statement must be the select code of the EPROM Programmer card currently connected to the EPROM memory card; if not, error 72 is reported.

The unit number must be one greater than the greatest unit number of any initialized EPROM unit currently in the system. For example, if the greatest unit number of an EPROM unit in the system is 3, then the unit to be initialized must be unit number 4.

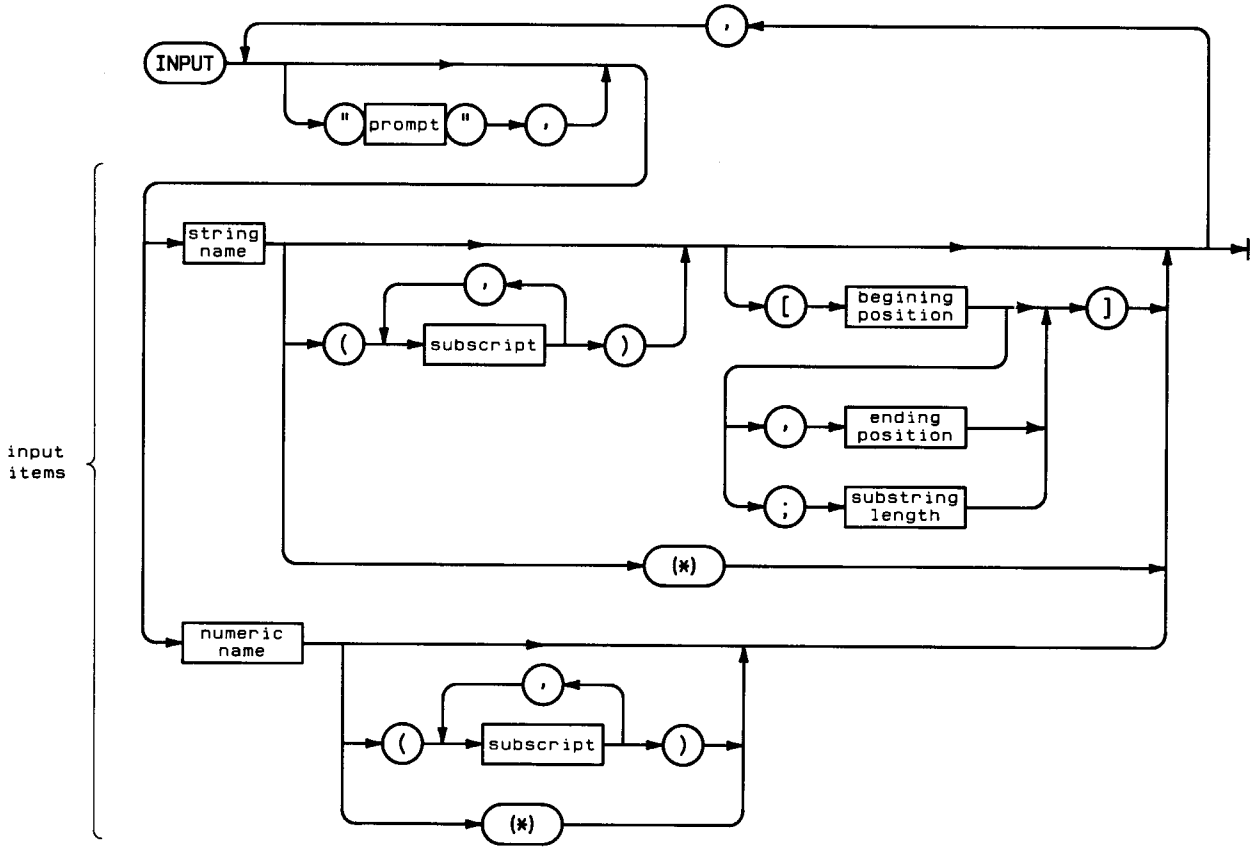
INPUT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement is used to assign keyboard input to program variables.



Expanded diagram:



Item	Description/Default	Range Restrictions
prompt	a literal composed of characters from the keyboard, including those generated using the ANY CHAR key; Default = question mark	—
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
numeric name	name of a numeric variable	any valid name

Example Statements

```
INPUT "Name?",N$,"ID Number?",Id
INPUT Array(*)
```

Semantics

Values can be assigned through the keyboard for any numeric or string variable, substring, array, or array element.

A prompt, which is allowed for each item in the input list, appears on the CRT display line. If the last DISP or DISP USING statement suppressed its EOL sequence, the prompt is appended to the current display line contents. If the last DISP or DISP USING did not suppress the EOL sequence, the prompt replaces the current display line contents.

Not specifying a prompt results in a question mark being used as the prompt. Specifying the null string (" ") for the prompt suppresses the question mark.

To respond to the prompt, the operator enters a number or a string. Leading and trailing blank characters are deleted. Unquoted strings may not contain commas or quote marks. Placing quotes around an input string allows any characters to be used as input. If " is intended to be a character in a quoted string, use " ".

Multiple values can be entered individually or separated by commas. Press the CONTINUE, RETURN, EXECUTE, ENTER or STEP after the final input response. Two consecutive commas cause the corresponding variable to retain its original value. Terminating an input line with a comma retains the old values for all remaining variables in the list.

The assignment of a value to a variable in the INPUT list is done as soon as the terminator (comma or key) is encountered. Not entering data and pressing [CONTINUE], [ENTER], [EXECUTE], [RETURN], or [STEP] retains the old values for all remaining variables in the list.

If [CONTINUE], [ENTER], [EXECUTE], or [RETURN] is pressed to end the data input, program execution continues at the next program line. If [STEP] is pressed, the program execution continues at the next program line in single step mode. (If the INPUT was stepped into, it is stepped out of, even if [CONTINUE], [ENTER], [EXECUTE], or [RETURN] is pressed.)

If too many values are supplied for an INPUT list, the extra values are ignored.

An entire array may be specified by the asterisk specifier. Inputs for the array are accepted in row major (right most subscript varies most rapidly).

Live keyboard operations are not allowed while an INPUT is awaiting data entry. [PAUSE] or [STOP] on an HP 46020A keyboard can be pressed so live keyboard operations can be performed. The INPUT statement is re-executed, beginning with the first item, when [CONTINUE] or [STEP] is pressed. All values for that particular INPUT statement must be re-entered.

ON KBD, ON KEY and ON KNOB events are deactivated during an INPUT statement. Errors do not cause an ON ERROR branch. If an input response results in an error, re-entry begins with the variable which would have received the erroneous response.

INT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the greatest integer which is less than or equal to the expression. The result will be of the same type (REAL or INTEGER) as the argument.



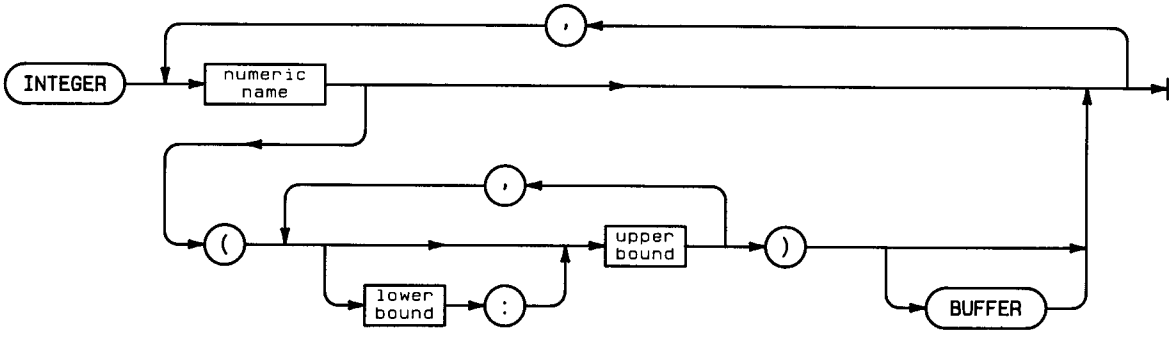
Example Statements

```
Whole=INT(Number)
IF X/2=INT(X/2) THEN Even
```

INTEGER

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement declares INTEGER variables, dimensions INTEGER arrays, and reserves memory for them. (For information about INTEGER as a secondary keyword, see the ALLOCATE, COM, DEF FN, or SUB statements.)



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or 1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)

Example Statements

```
INTEGER I,J,K
INTEGER Array(-128:255)
INTEGER A(4096) BUFFER
```

Semantics

An INTEGER variable (or an element of an INTEGER array) uses two bytes of storage space. An INTEGER array can have a maximum of six dimensions. The maximum number of elements is a function of your computer's memory size, but no single dimension can have more than 32 767 total elements.

Declaring Buffers

To declare INTEGER variables to be buffers, each variable's name must be followed by the keyword BUFFER; the designation BUFFER applies only to the variable which it follows.

INTENSITY

See the AREA and SET PEN statements.

INTERACTIVE

See the RESUME INTERACTIVE and SUSPEND INTERACTIVE statements.

INTR

See the OFF INTR and ON INTR statements.

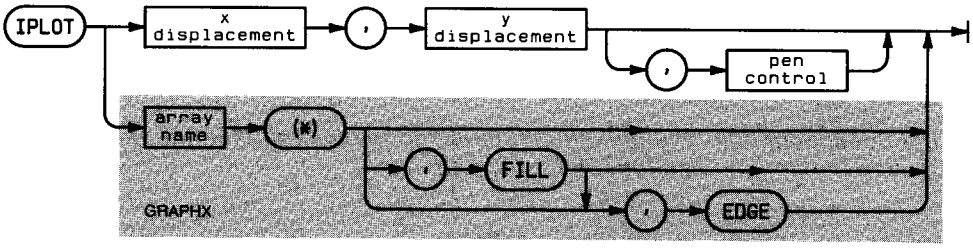
INV

See the MAT statement.

IPLOT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement moves the pen from the current pen position to the point specified by adding the specified X and Y displacements to the current pen position. It can be used to move without drawing a line, or to draw a line, depending on the pen control parameter.



Item	Description/Default	Range Restrictions
x displacement	numeric expression, in current units	—
y displacement	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move)	−32 768 thru +32 767
array name	name of two-dimensional, two-column or three-column numeric array. Requires GRAPHX.	any valid name

Example Statements

```
IPLOT X,Y,Pen
IPLOT -5,12
IPLOT Shape(*),FILL,EDGE
```

Semantics

Non-Array Parameters

The specified X and Y displacement information is interpreted according to the current unit-of-measure. Lines are drawn using the current pen color and line type.

The line is clipped at the current clipping boundary. IPLOT is affected by PIVOT and PDIR transformations.

If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

The optional pen control parameter specifies the following plotting actions; the default value is +1 (down after move).

Pen Control Parameter

Pen Control	Resultant Action
−Even	Pen up before move
−Odd	Pen down before move
+Even	Pen up after move
+Odd	Pen down after move

That is, even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Array Parameters

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the ranges 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the IPLOT statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the IPLOT statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the IPLOT statement, FILL must occur first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

When using an IPLOT statement with an array, the following table of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth. See the list below.

Column 1	Column 2	Operation Selector	Meaning
X	Y	−2	Pen up before moving
X	Y	−1	Pen down before moving
X	Y	0	Pen up after moving (Same as +2)
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	ignored	3	Select pen
line type	repeat value	4	Select line type
color	ignored	5	Color value
ignored	ignored	6	Start polygon mode with FILL
ignored	ignored	7	End polygon mode
ignored	ignored	8	End of data for array
ignored	ignored	9	NOP (no operation)
ignored	ignored	10	Start polygon mode with EDGE
ignored	ignored	11	Start polygon mode with FILL and EDGE
ignored	ignored	12	Draw a FRAME
pen number	ignored	13	Area pen value
red value	green value	14	} Color
blue value	ignored	15	} Value
ignored	ignored	>15	Ignored

Moving and Drawing

If the operation selector is less than or equal to two, it is interpreted in exactly the same manner as the third parameter in a non-array IPLOT statement. As mentioned above, even means lift the pen up, odd means put the pen down, positive means act after pen motion, negative means act before pen motion. Zero is considered positive.

Selecting Pens

The operation selector of 3 is used to select pens. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types

The operation selector of 4 is used to select line types. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color

Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation Selector 14 is used in conjunction with Operation Selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessarily immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation Selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word, that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities for red, green, and blue ranging from zero to one in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map identical to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons (series of draws separated by moves) can be filled without terminating the mode with an operation selector 7. The first and second columns are ignored; therefore they should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored, and the X and Y values of the last data point should not be in them. Edging and/or filling will begin immediately upon encountering this operation selector.

Doing a FRAME

Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the IPLOT statement, so one probably would not have more than one operation selector 12 in an array to IPLOT, since the last FRAME will overwrite all the previous ones.

Premature Termination

Operation selector 8 causes the IPLOT statement to be terminated. The IPLOT statement will successfully terminate if the actual end of the array has been reached, so the use of operation selector 8 is optional.

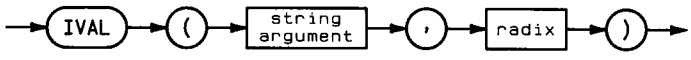
Ignoring Selected Rows in the Array

Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

IVAL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a binary, octal, decimal, or hexadecimal string expression into an INTEGER.



Item	Description/Default	Range Restrictions
string argument	string expression, containing digits valid for the specified base	(see table)
radix	numeric expression, rounded to an integer	2, 8, 10, or 16

Example Statements

```
Number=IVAL("FDF0",16)
I=IVAL("1111111111111110",2)
DISP IVAL(Octal$,8)
```

Semantics

The radix is a numeric expression that will be rounded to an integer and must evaluate to 2, 8, 10, or 16.

The string expression must contain only the characters allowed for the particular number base indicated by the radix. ASCII spaces are not allowed.

Binary strings are presumed to be in two's-complement form. If all 16 digits are specified and the leading digit is a 1, the returned value is negative.

Octal strings are presumed to be in the octal representation of two's-complement form. If all 6 digits are specified, and the leading digit is a 1, the returned value is negative.

Decimal strings containing a leading minus sign will return a negative value.

Hex strings are presumed to be in the hex representation of the two's-complement binary form. The letters A through F may be specified in either upper or lower case. If all 4 digits are specified and the leading digit is 8 through F the returned value is negative.

Radix	Base	String Range	String Length
2	binary	0 thru 1111111111111111	1 to 16 characters
8	octal	0 thru 177777	1 to 6 characters
10	decimal	−32768 thru +32768	1 to 6 characters
16	hexadecimal	0 thru FFFF	1 to 4 characters

Radix	Legal Characters	Comments
2	+,0,1	—
8	+,0,1,2,3,4,5,6,7	Range restricts the leading character. Sign must be a leading character.
10	+,−,0,1,2,3,4, 5, 6,7,8,9	Signs must be a leading character.
16	+,0,1,2,3,4,5,6,7, 8,9,A,B,C,D,E,F, a,b,c,d,e,f	A/a = 10, B/b = 11, C/c = 12, D/d = 13 E/e = 14, F/f = 15

IVAL$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts an INTEGER into a binary, octal, decimal, or hexadecimal string.



Item	Description/Default	Range Restrictions
"16-bit" argument	numeric expression, rounded to an integer	(see table)
radix	numeric expression, rounded to an integer	2, 8, 10, or 16

Example Statements

```
F$=IVAL$(-1,16)
Binary$=IVAL$(Count DIV 256,2)
```

Semantics

The rounded argument must be a value that can be expressed (in binary) using 16 bits or less.

The radix must evaluate to be 2, 8, 10, or 16; representing binary, octal, decimal, or hexadecimal notation.

If the radix is 2, the returned string is in two's-complement form and contains 16 characters. If the numeric expression is negative, the leading digit will be 1. If the value is zero or positive there will be leading zeros.

If the radix is 8, the returned string is the octal representation of the two's-complement binary form and contains 6 digits. Negative values return a leading digit of 1.

If the radix is 10, the returned string contains 6 characters. Leading zeros are added to the string if necessary. Negative values have a leading minus sign.

If the radix is 16, the returned string is the hexadecimal representation of the two's-complement binary form and contains 4 characters. Negative values return a leading digit in the range 8 thru F.

Radix	Base	Range of Returned String	String Length
2	binary	0000000000000000 thru 1111111111111111	16 characters
8	octal	000000 thru 177777	6 characters
10	decimal	−32768 thru 032768	6 characters
16	hexadecimal	0000 thru FFFF	4 characters

KBD

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This INTEGER function returns a 2, the select code of the keyboard.



Example Statements

```
STATUS KBD;Kbd_status
OUTPUT KBD;Clear$;
```

KBD$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the contents of the buffer established by ON KBD.



Example Statements

```
Keys$=KBD$
IF Active THEN Command$=Command$&KBD$
```

Semantics

When an ON KBD branch is in effect, all subsequent keystrokes are trapped and held in a special "keyboard" buffer. The KBD$ function returns the contents of this buffer and then clears it. A null string is returned if the buffer is empty or no ON KBD branch is active.

Non-ASCII keys are stored in the buffer as two bytes; the first has a decimal value of 255, and the second specifies the key. Pressing CTRL and a non-ASCII key simultaneously generates three bytes; the first two have a decimal value of 255, and the third specifies the key. See the *Second Byte of Non-ASCII Key Sequences* table in the "Useful Tables" section for a list of these keycodes.

The buffer can hold 256 characters. Further keystrokes are not saved and produce beeps. An overflow flag is set after the buffer is full. This flag can be checked by reading keyboard status register 5 and is cleared by reading the status register SCRATCH A, and a RESET operation.

The buffer is cleared by KBD$, OFF KBD, SCRATCH, SCRATCH A, INPUT, LINPUT, ENTER 2, and a RESET operation.

KEY

See the OFF KEY and ON KEY statements.

KNOB

See the OFF KNOB and the ON KNOB statements.

KNOBX

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the net number of horizontal knob pulses counted since the last time the KNOBX counter was zeroed.



Example Statements

```
Position=KNOBX
IF KNOBX<0 THEN Backwards
```

Semantics

Sampling occurs during the time interval established by the ON KNOB statement. The counter is zeroed when the KNOBX function is called and at the times specified in the Reset Table at the back of this manual. Clockwise rotation gives positive counts; counter-clockwise rotation gives negative counts. There are 120 counts for one revolution of the knob. If there is no active ON KNOB definition, KNOBX returns zero.

Counts are accumulated by the KNOBX function during each ON KNOB sampling interval. The pulse count during each sampling interval is limited to −127 thru +128. The limits of the KNOBX function are −32 768 thru +32 767.

You can use a relative pointing device, such as the HP 46060A on an HP 46020A keyboard, if the KBD BIN is loaded.

Note

KNOBX functions differently if BIN file KNB2_0 is loaded. Refer to the Knob section in Chapter 15 of the *BASIC Programming Techniques* manual for more information.

KNOBY

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the net number of vertical knob pulses counted since the last time the KNOBY counter was zeroed.



Example Statements

```
Position=KNOBY
IF KNOBY<0 THEN Backwards
```

Semantics

Sampling occurs during the time interval established by the ON KNOB statement. The counter is zeroed when the KNOBY function is called and at the times specified in the Reset Table at the back of this manual. Clockwise rotation gives positive counts; counter-clockwise rotation gives negative counts. There are 120 counts for one revolution of the knob. If there is no active ON KNOB definition, KNOBY returns zero.

Counts are accumulated by the KNOBY function during each ON KNOB sampling interval. The pulse count during each sampling interval is limited to −127 thru +128. The limits of the KNOBY function are −32 768 thru +32 767.

You can use a relative pointing device, such as the HP 46060A on an HP 46020A keyboard, if the KBD BIN is loaded.

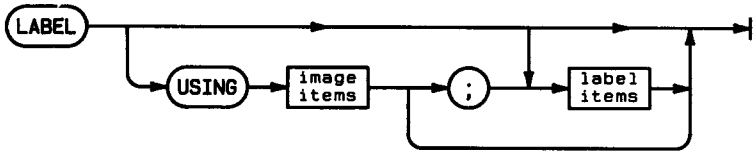
Note

KNOBY functions differently if BIN file KNB2_0 is loaded. Refer to the Knob section in Chapter 15 of the *BASIC Programming Techniques* manual for more information.

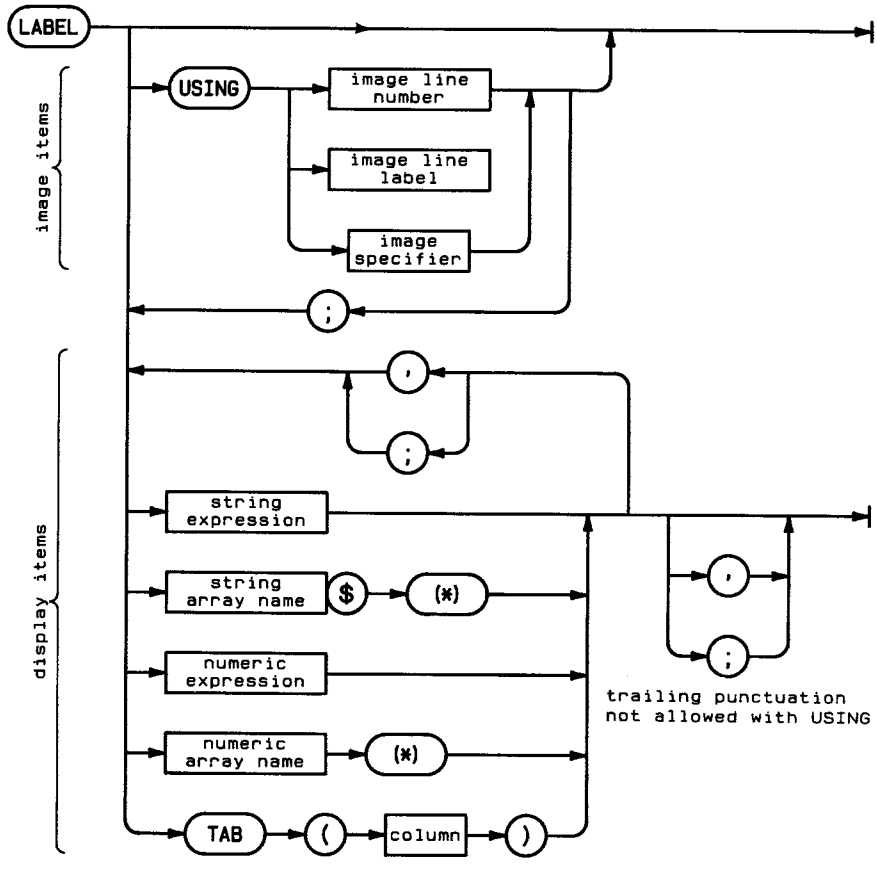
LABEL

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

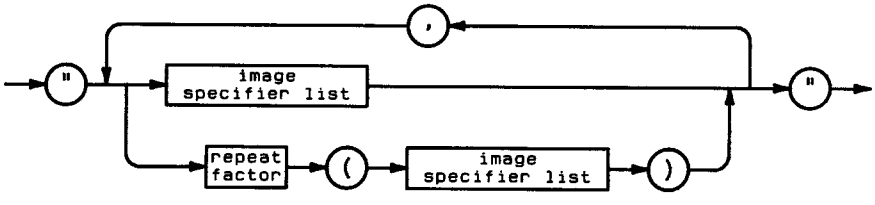
This statement produces alphanumeric labels on graphic devices. (For information about LABEL as a secondary keyword, *see* the ON KEY statement.)

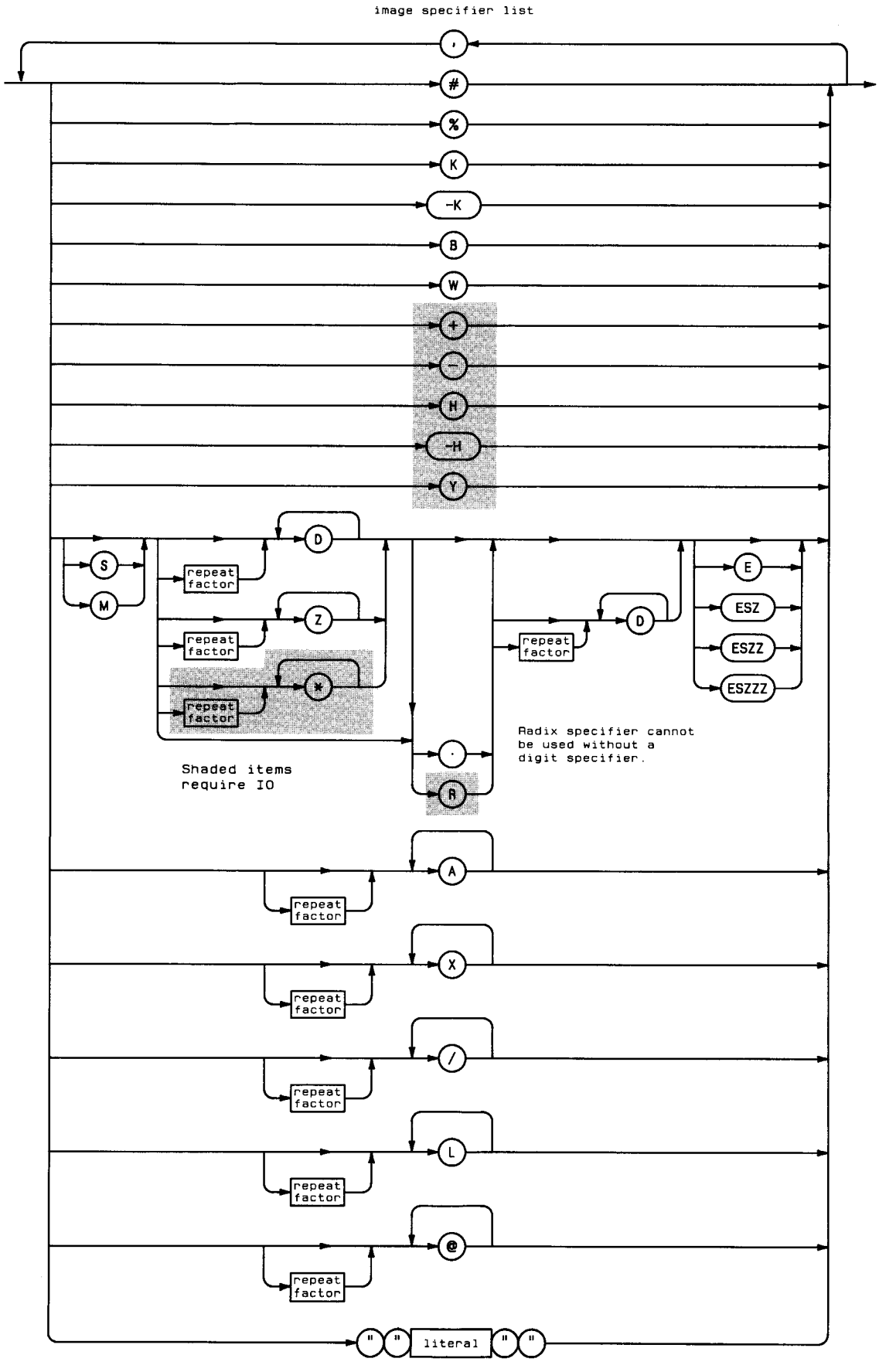


Expanded diagram:



literal form of image specifier:




image specifier list
,
#
%
K
-K
B
W
+
-
H
-H
Y
S
M
repeat factor
D
Z
X
E
ESZ
ESZZ
ESZZZ
.
R
Shaded items require IO
Radix specifier cannot be used without a digit specifier.
A
/
L
@
" " literal " "

Item	Description/Default	Range Restrictions
image line number	integer constant identifying an IMAGE statement	1 thru 32 766
image line label	name identifying an IMAGE statement	any valid name
image specifier	string expression	(see drawing)
string array name	name of a string array	any valid name
numeric array name	name of a numeric array	any valid name
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

Example Statements

```
LABEL Number,String$
LABEL USING "5Z.DD";Money
```

Semantics

The label begins at the current logical pen position, with the current pen. Labels are clipped at the current clip boundary. Other statements which affect label generation are PEN, LINE TYPE, PIVOT, CSIZE, LORG, and LDIR. The current pen position is updated at the end of the label operation.

Standard Numeric Format

The standard numeric format depends on the value of the number being output. If the absolute value of the number is greater than or equal to 1E−4 and less than 1E+6, it is rounded to 12 digits and output in floating point notation. If it is not within these limits, it is output in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the label list is exhausted, an End-of-Line (EOL) sequence is sent to the logical pen, unless it is suppressed by trailing punctuation or a pound-sign image specifier. The EOL sequence is also sent after every 256 characters. This "plotter buffer exceeded" EOL is not suppressed by trailing punctuation, but is suppressed by the pound-sign specifier.

Control Codes

Some ASCII control codes have a special effect in LABEL statements.

Character	Keystroke	Name	Action
CHR$(8)	CTRL-H	backspace	Back up the width of one character cell.
CHR$(10)	CTRL-J	linefeed	Move down the height of one character cell.
CHR$(13)	CTRL-M	carriage return	Move back the length of the label just completed.

Any control character that the LABEL statement does not recognize is treated as an ASCII blank [CHR$(32)].

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

Arrays

Arrays may be output as labels by using the asterisk specifier. They are output in row-major order (right-most subscript varies most rapidly) and their format depends on the label mode selected.

LABEL Without Using

If LABEL is used without USING, the punctuation following an item determines the width of the item's label field; a semicolon selects the compact field, and a comma selects the default label field. When the label item is an array with the asterisk array specifier, each array element is considered a separate label item. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the label field to be used for the label item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are output with one trailing blank. String items are output with no leading or trailing blanks.

The default label field labels items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is output with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

LABEL With Using

When the computer executes a LABEL USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the label items, the field specifier is acted upon without accessing the label list. When the field specifer requires characters, it accesses the next item in the label list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching label item. If the image specifiers are exhausted before the label items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than are specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

Effects of the image specifiers on the LABEL statement are shown in the following table:

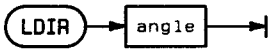
Image Specifier	Meaning
K	Compact field. Outputs a number or string as a label in standard form with no leading or trailing blanks.
−K	Same as K.
H	Similar to K, except the number is output using the European number format (comma radix). (Requires IO)
−H	Same as H. (Requires IO)
S	Outputs the number's sign (+ or −) as a label.
M	Outputs the number's sign as a label if negative, a blank if positive.
D	Outputs one digit character as a label. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is output, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are output.
*	Same as Z, except that asterisks are output instead of leading zeros. (Requires IO)
.	Outputs a decimal-point radix indicator as a label.
R	Outputs a comma radix indicator as a label (European radix). (Requires IO)
E	Outputs as a label: an E, a sign, and a two-digit exponent.
ESZ	Outputs as a label: an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Outputs as a label: an E, a sign, and a three-digit exponent.

Image Specifier	Meaning
A	Outputs a string character as a label. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Outputs a blank as a label.
literal	Outputs as a label the characters contained in the literal.
B	Outputs as a label the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.
W	Outputs as a label two characters represented by the two bytes of a 16-bit, two's-complement integer. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is used; if it is less than −32 768, then −32 768 is used. The most-significant byte is sent first.
Y	Same as W. (Requires IO)
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence following the last label item.
%	Ignored in LABEL images.
+	Changes the automatic EOL sequence that normally follows the last label item to a single carriage-return. (Requires IO.)
−	Changes the automatic EOL sequence that normally follows follows the last label item to a single line-feed. (Requires IO)
/	Sends a carriage-return and a line-feed to the PLOTTER IS device.
L	Same as /.
@	Sends a form-feed to the PLOTTER IS device; produces a blank.

LDIR

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines the angle at which a label or symbol is drawn. The angle is interpreted as counterclockwise, from horizontal. The current angle mode is used.



Item	Description/Default	Range Restrictions
angle	numeric expression in current units of angle; Default = 0	(same as COS)

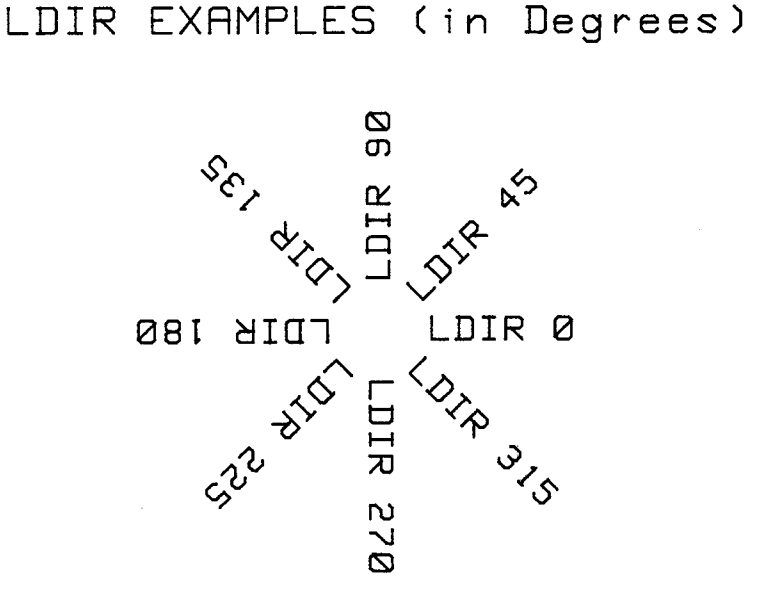
Example Statements

```
LDIR 90
LDIR ACS(Side)
```

Semantics

LDIR affects the appearance of LABEL, LABEL USING and SYMBOL output.

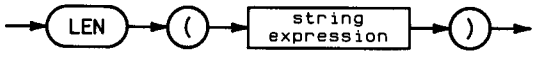
The angle is interpreted as shown below.



LEN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the current number of characters in the argument. The length of the null string ("") is 0.



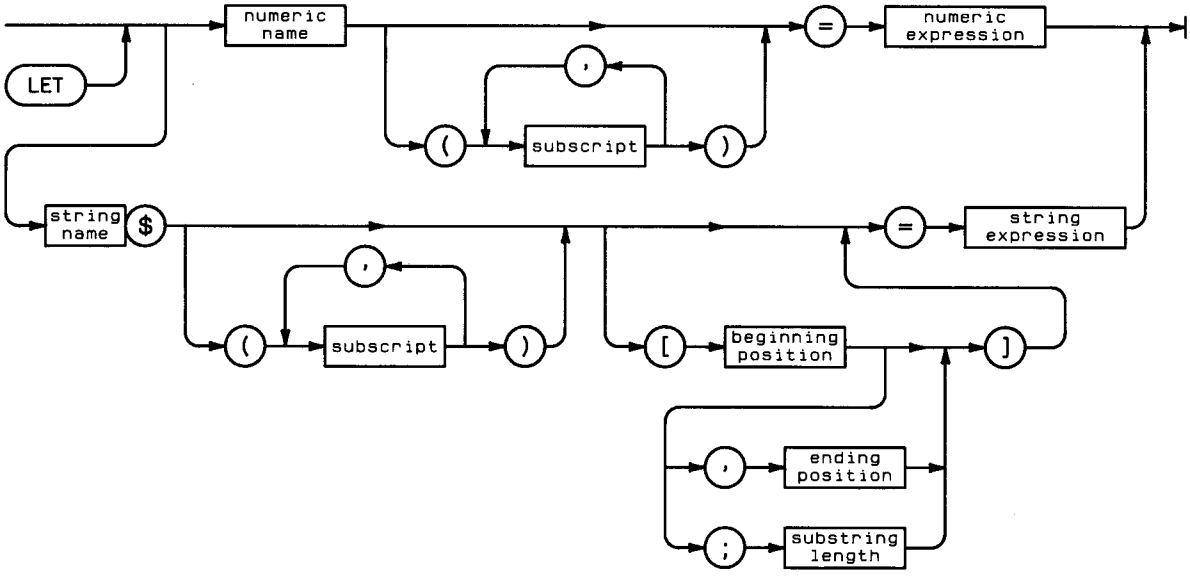
Example Statements

```
Last=LEN(String$)
IF NOT LEN(A$) THEN Empty
```

LET

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This is the assignment statement, which is used to assign values to variables.



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
LET Number=33
Array(I+1)=Array(I)/2
String$="Hello Scott"
A$(7)[1;2]=CHR$(27)&"Z"
```

Semantics

The assigment is done to the variable which is to the left of the equals sign. Only one assignment may be performed in a LET statement; any other equal signs are considered relational operators, and must be enclosed in a parenthetical expression (i.e. `A=A+(B=1)+5`). A variable can occur on both sides of the assignment operator (i.e. `I=I+1` or `Source$=Source$&Temp$`).

A real expression will be rounded when assigned to an INTEGER variable, if it is within the INTEGER range. Out-of-range assignments to an INTEGER give an error.

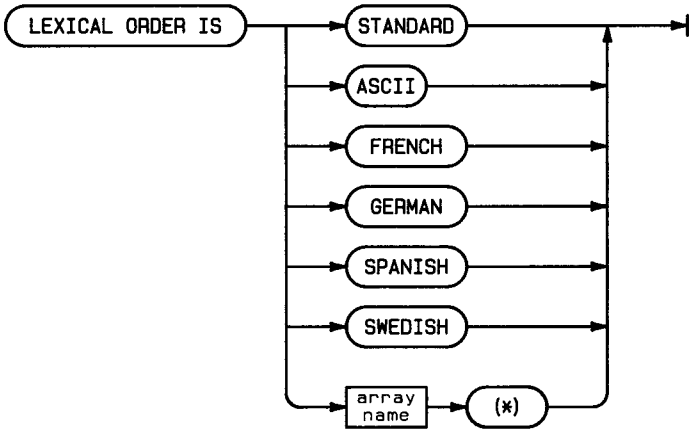
The length of the string expression must be less than or equal to the dimensioned length of the string it is being assigned to. Assignments may be made into substrings, using the normal rules for substring definition. The string expression will be truncated or blank-filled on the right (if necessary) to fit the destination substring when the substring has an explicitly stated length. If only the beginning position of the substring is specified, the substring will be replaced by the string expression and the length of the recipient string variable will be adjusted accordingly; however, error 18 is reported if the expression overflows the recipient string variable.

If the name of the variable to the left of the equal sign begins with AND, DIV, EXOR, MOD or OR (the name of an operator) and the keyword LET is omitted, the prefix must have at least one uppercase letter and one lowercase letter in it. Otherwise, a live keyboard execution is attempted and fails, even though the line number is present.

LEXICAL ORDER IS

Option Required	LEX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines the collating sequence for all string relational operators and operations.



Item	Description/Default	Range Restrictions
array name	the name of a one-dimensional INTEGER array, with at least 257 elements	any valid name

Examples

```
LEXICAL ORDER IS FRENCH
LEXICAL ORDER IS Lex_table(*)
```

Semantics

The STANDARD lexical order is determined by the internal keyboard jumper preset to match the language on the keyboard. For example, with an English language or Katakana keyboard, the STANDARD lexical order is the same as the ASCII lexical order.

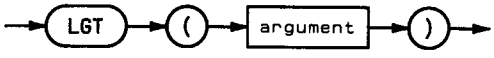
The default lexical order is STANDARD. This is also true after a SCRATCH A. The most recent LEXICAL ORDER IS statement overrides any previous definition and affects all contexts.

Lexical order allows languages to be properly collated. This includes such treatments as ignoring characters, dealing with accents, and character replacements. See *BASIC Programming Techniques* manual for the details of pre-defined and user-defined lexical order tables.

LGT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the logarithm (base 10) of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	greater than 0

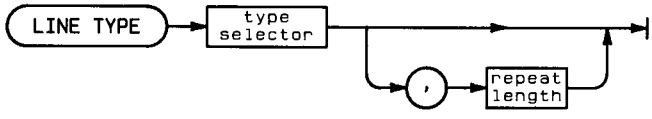
Example Statements

```
Decibel=20*LGT(Volts)
PRINT "Log of";X;"=";LGT(X)
```

LINE TYPE

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a line type and repeat length for lines, labels, frames, axes and grids.



Item	Description/Default	Range Restrictions	Recommended Range
type selector	numeric expression, rounded to an integer; Default = 1	1 thru 10	—
repeat length	numeric expression, rounded to an integer; Default = 5	−32 768 thru +32 767	greater than 0

Example Statements

```
LINE TYPE 1
LINE TYPE Select,20
```

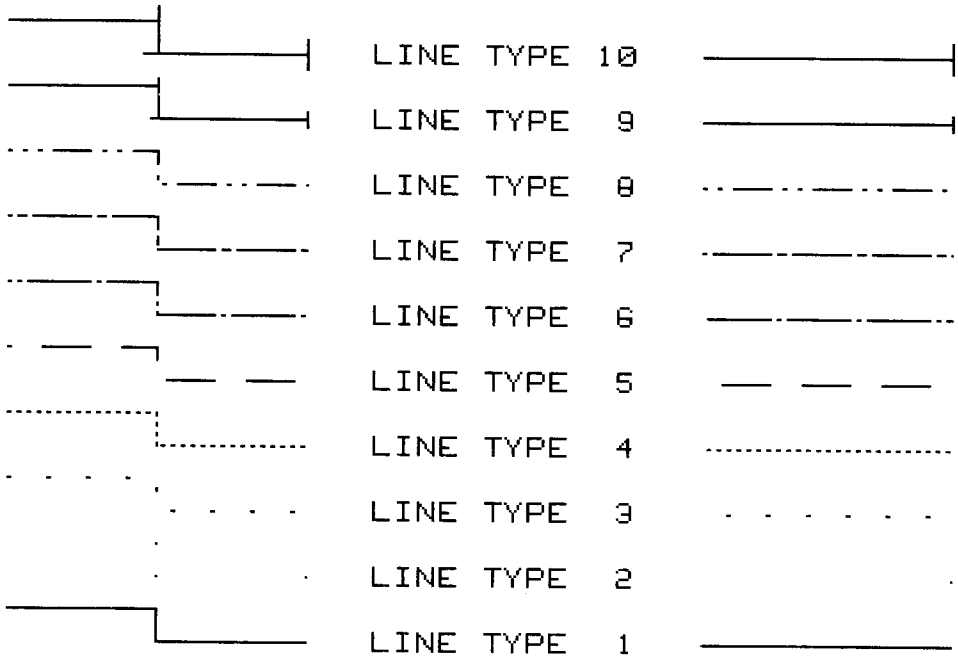
Semantics

At power-up the default line type is a solid line (type 1), and the default repeat length is 5 GDUs.

The repeat length establishes the number of GDUs required to contain an arbitrary seqment of the line pattern. When the plotter is the internal CRT, the repeat length is evaluated and taken as the next lower multiple of 5, with a minimum value of 5.

When the plotter is an external plotter, the line produced by the line identifier is device dependent. Refer to your plotter's documentation for further information.

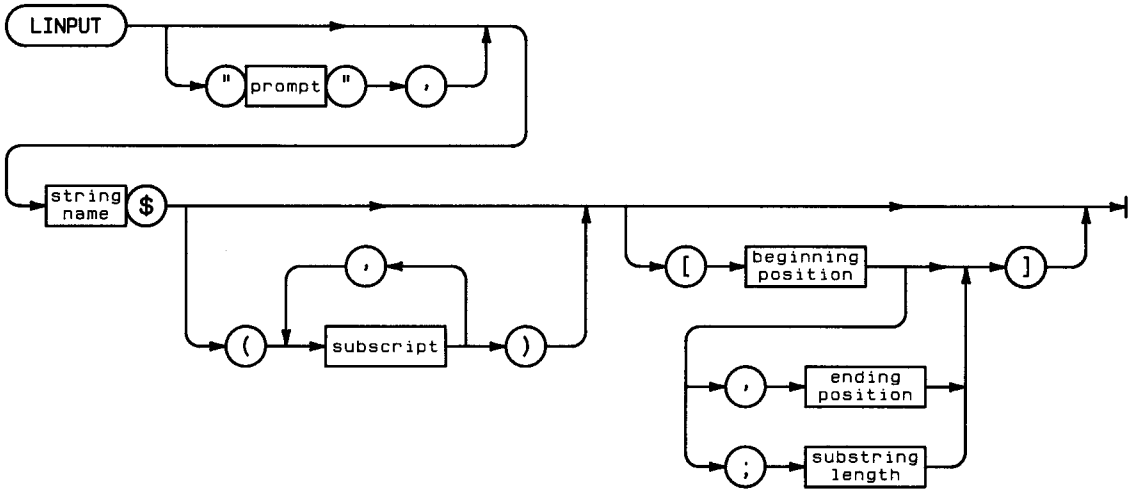
The available CRT line types are shown here.



LINPUT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement accepts alphanumeric input from the keyboard for assignment to a string variable. The LINPUT statement allows commas or quotation marks to be included in the value of the string, and leading or trailing blanks are not deleted.



Item	Description/Default	Range Restrictions
prompt	a literal composed of characters from the keyboard, including those generated using the ANY CHAR key; Default = question mark	—
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
LINPUT "Next Command?",Response$
LINPUT Array$(I)[3]
```

Semantics

A prompt, which remains until the LINPUT item is satisfied, appears on the CRT display line. If the last DISP statement suppressed its CR/LF, the prompt is appended onto the current display line contents. If the last DISP did not suppress the CR/LF, the prompt replaces the current display line contents. Not specifying a prompt results in the question mark being used as the prompt. Specifying the null string (" ") for the prompt suppresses the question mark.

CONTINUE, ENTER, EXECUTE, RETURN, or STEP must be pressed to indicate that the entry is complete. If no value is provided from the keyboard, the null string is used. If CONTINUE, ENTER, EXECUTE, or RETURN is pressed to end the data input, program execution continues at the next program line. If STEP is pressed, the program execution continues at the next program line in single step mode. (If the LINPUT was stepped into, it is stepped out of, even if CONTINUE, ENTER, EXECUTE, or RETURN is pressed.)

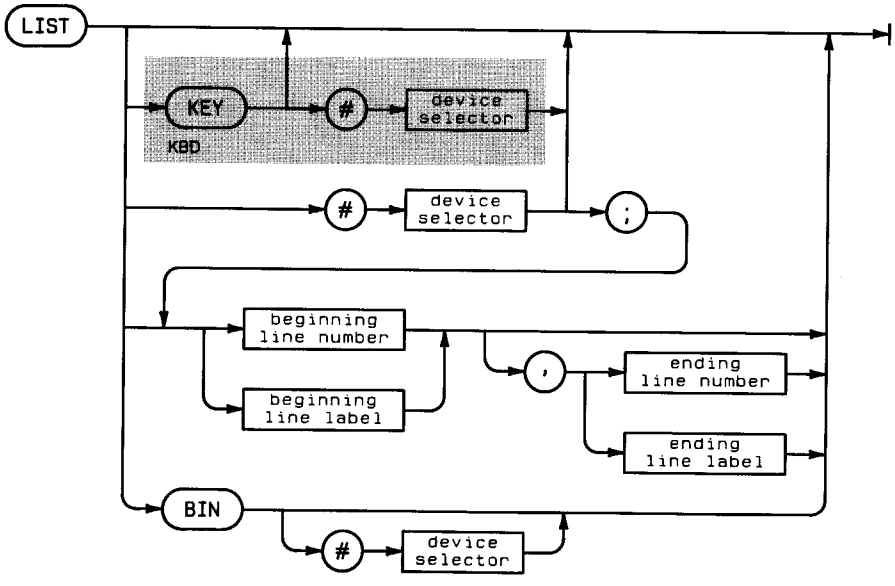
Live keyboard operations are not allowed while a LINPUT is waiting for data entry. PAUSE (or STOP on an HP 46020A keyboard) can be pressed so live keyboard operations can be performed. The LINPUT statement is re-executed from the beginning when CONTINUE or STEP is pressed.

ON KBD, ON KEY and ON KNOB events are deactivated during an LINPUT statement. Errors do not cause an ON ERROR branch. If an input response results in an error, the LINPUT statement is re-executed.

LIST

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows you to list the program or the key definitions currently in memory.



Item	Description/Default	Range Restrictions
device selector	numeric expression; is rounded to an integer. Default is PRINTER IS device.	(see Glossary)
beginning line number	integer constant identifying program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
LIST
LIST #701
LIST 100,Label1
LIST KEY
```

Semantics

LIST

When a label is used as a line identifier, the lowest-numbered line in memory having that label is used. When a number is used as a line identifier, the lowest-numbered line in memory having a number equal to or greater than the specified line is used. An error occurs if the ending line identifier occurs before the beginning line identifier or if a specified line label does not exist in the program.

Executing a LIST from the keyboard while a program is running causes the program to pause at the end of the current line. The listing is sent to the selected device, and program execution resumes.

After the listing is finished, the amount of available memory, in bytes, is displayed on the CRT.

LIST KEY (Requires KBD)

The LIST KEY statement lists the current typing-aid key definitions (**not** the labels of ON KEY definitions) to the specified device. If a key does not currently have a definition, it will not be listed.

LIST BIN

The LIST BIN statement lists the BINs currently loaded in memory. The name, version and brief description of the BIN is listed. For example:

NAME	VERSION	DESCRIPTION
GRAPH	4.0	Graphics
MAT	4.0	Matrix Statements

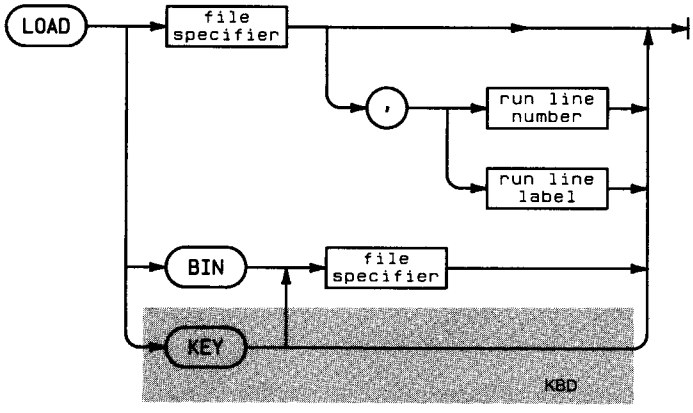
LISTEN

See the SEND statement.

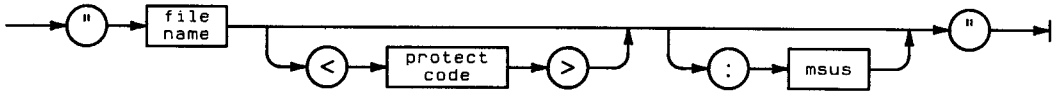
LOAD

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows you to load programs, BIN files or typing-aid key definitions into the computer. (If using LOAD with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)
run line number	integer constant identifying program line	1 thru 32 765
run line label	name of a program line	any valid name

Example Statements

```
LOAD Filename$&Msus$
LOAD "UTIL",120
LOAD BIN "MAT"
LOAD KEY "KEYS:INTERNAL,4,1"
```

Semantics

LOAD

The BASIC program and all variables not in COM are lost when a LOAD is executed. Every COM block in the newly-loaded program is compared with the COM blocks of the program in memory. If a COM area of the newly-loaded program does not match an existent COM area, the values in the old COM area are lost. Thus, some COM areas may be retained while others are lost. If a PROG file contains a binary program that is compatible with BASIC 4.0, the binary is skipped and the program is loaded.

LOAD is allowed from the keyboard if a program is not running. If no run line is specified, [RUN] must be pressed to initiate program execution, and execution will begin on the first line in the program. If a run line is specified, prerun initialization (see RUN) is performed, and program execution begins at the line specified. The line on which execution begins must exist in the main program context of the newly-loaded program. If you specify a line *number* and it doesn't exist, execution begins with the next higher-numbered line, provided that line is in the main program context.

Executing LOAD from a program causes the new program file to be loaded, prerun, and program execution to resume. Execution begins at the line specified, or the lowest-numbered program line if a run line is not specified.

BASIC automatically loads and runs a file called AUTOST if the file exists on the same mass storage device as the system. If the system is loaded from SRM, the autostart file is /SYSTEMS/AUTOSTnn, where nn is the node number. If this file does not exist on the SRM, BASIC looks at the root directory for a file called AUTOST.

LOAD BIN

LOAD BIN is used to load system BIN files such as MAT. Executing a LOAD BIN does not affect the currently loaded BASIC program or any variables.

A BIN file contains either language extensions (such as MAT or GRAPH) or drivers (such as DISC). A BIN file may contain more than one of the extensions or drivers; if so, when loaded, only the entensions or drivers not already present in memory are loaded.

BIN files can be loaded from an external device (or SRM) even though the BIN to access that device is absent.

LOAD KEY (Requires KBD)

LOAD KEY sets the typing-aid definitions of the softkeys according to the contents of the specified BDAT file. If the file is not in the proper format, an error occurs. The file containing the key definitions may be created by a user program. See the STORE KEY statement for file format.

All existing key definitions are cleared before the file's key definitions are loaded.

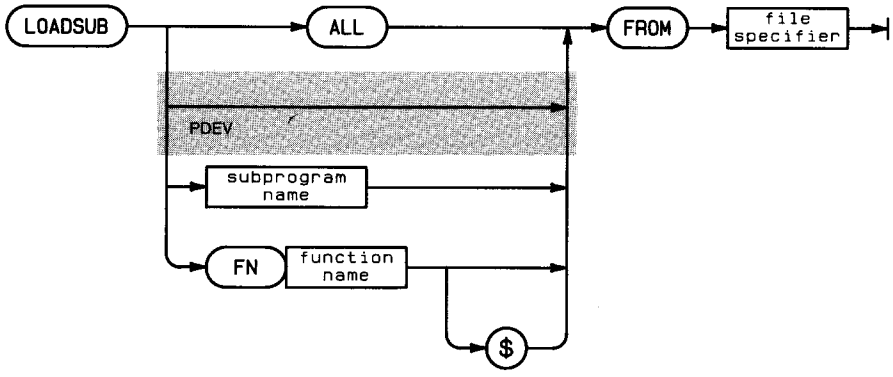
If LOAD KEY is executed without a file specifier, the keys are reset to their power-on values.

ON KEY definitions are not affected by LOAD KEY.

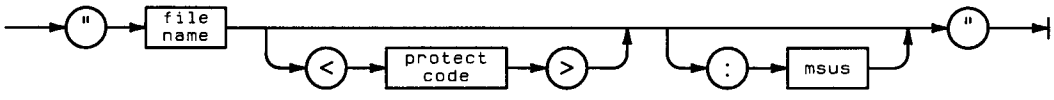
LOADSUB

Option Required	None
Keyboard Executable	Yes
Programmable	Yes (See Semantics)
In an IF...THEN...	Yes (See Semantics)

This statement allows you to load subprograms from a PROG file into the computer. (If using LOADSUB with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
subprogram name	name of a SUB or CSUB subprogram	any valid name
function name	name of a user-defined function	any valid name
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
LOADSUB FROM "APSUBS"                    (Not Programmable)
LOADSUB FNReplace$ FROM "SUBFILE"        (Programmable)
LOADSUB ALL FROM Subfile$                (Programmable)
```

Semantics

LOADSUB FROM (Requires PDEV)

The command LOADSUB FROM (without a subprogram name) **is not programmable**; it is used before a program is run. It looks through the program and notices all the subprogram references which are unsatisfied. Unsatisfied references are statements which reference subprograms that don't yet exist in memory. It then accesses the specified file (which must be a PROG file), and loads all the needed subprograms, appending them to the end of the current program, renumbering as necessary. It also looks through the subprograms it just loaded to see if *they* call anything which is not yet in memory. If so, those references will be satisfied. This process repeats for each set of subprograms loaded until all the routines that are referenced are loaded or until it is determined they are not on the specified file. At the end of the LOADSUB FROM command, if there are still unsatisfied references, an error message and a list of the subprograms names still needed is displayed.

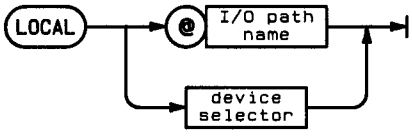
LOADSUB ALL FROM
LOADSUB <subprogram name> FROM

LOADSUB, when a subprogram name or ALL is included, loads the specified subprogram(s) from the specified file. This form **is programmable**. If either the file name or the subprogram name specified is not found, or the file name is not a PROG file, an error will occur. As the subprogram is loaded, it will be renumbered to fit at the end of the program. LOADSUB does not cause the program or any data currently in memory to be lost.

LOCAL

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement returns all specified devices to their local state.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
LOCAL @Dvm
LOCAL 7
```

Semantics

If only an interface select code is specified by the I/O path name or device selector, all devices on the bus are returned to their local state by setting REN false. Any existing LOCAL LOCKOUT is cancelled.

If a primary address is included, the GTL message (Go To Local) is sent to all listeners. LOCAL LOCKOUT is not cancelled.

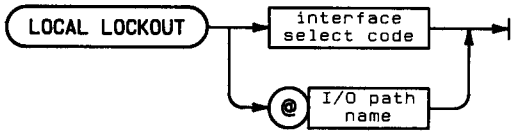
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{REN}$ $\overline{ATN}$	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	$\overline{REN}$	Error	Error	

LOCAL LOCKOUT

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This HP-IB statement sends the LLO (local lockout) message, preventing an operator from returning the specified device to local (front panel) control.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
LOCAL LOCKOUT 7
LOCAL LOCKOUT @Hpib
```

Semantics

The computer must be the active controller to execute LOCAL LOCKOUT.

If a device is in the LOCAL state when this message is sent, it does not take effect on that device until the device receives a REMOTE message and becomes addressed to listen.

LOCAL LOCKOUT does not cause bus reconfiguration, but issues a universal bus command received by all devices on the interface whether addressed or not. The command sequence is ATN and LLO.

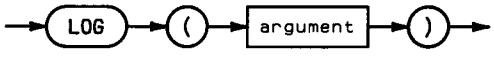
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error			

LOG

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the natural logarithm (base e) of the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	greater than 0

Example Statements

```
Time=-1*Rc*LOG(Volts/Emf)
PRINT "Natural log of";Y;"=";LOG(Y)
```

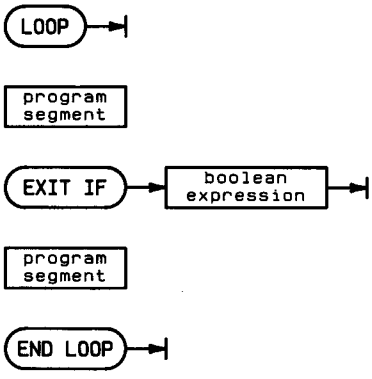
LOCATOR

See the READ LOCATOR and SET LOCATOR statements.

LOOP

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the boolean expression in an EXIT IF statement evaluates to be logically true (evaluates to a non-zero value).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if 0	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
460   LOOP
470   EXIT IF LEN(A$)<2
480     P=POS(A$,Delim$)
490   EXIT IF NOT P
500     A$=A$[1,P-1]&A$[P+2]
510   END LOOP

1200  LOOP  ! Until an EOF branch
1210    ENTER @File;Text$
1220    PRINT Text$
1230  END LOOP
```

Semantics

The LOOP...END LOOP construct allows continuous looping with conditional exits which depend on the outcome of relational tests placed within the program segments. The program segments to be repeated start with the LOOP statement and end with END LOOP. Reaching the END LOOP statement will result in a branch to the first program line after the LOOP statement.

Any number of EXIT IF statements may be placed within the construct to escape from the loop. The only restriction upon the placement of the EXIT IF statements is that they must not be part of any other construct which is nested within the LOOP...END LOOP construct.

If the specified conditional test is true, a branch to the first program line following the END LOOP statement is performed. If the test is false, execution continues with the next program line within the construct.

Branching into a LOOP...END LOOP construct (via a GOTO) results in normal execution from the point of entry. Any EXIT IF statement encountered will be executed. If execution reaches END LOOP, a branch is made back to the LOOP statement, and execution continues as if the construct had been entered normally.

Nesting Constructs Properly

LOOP...END LOOP may be placed within other constructs, provided it begins and ends before the outer construct can end.

LORG

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the relative origin of a label or symbol with respect to the current pen position.



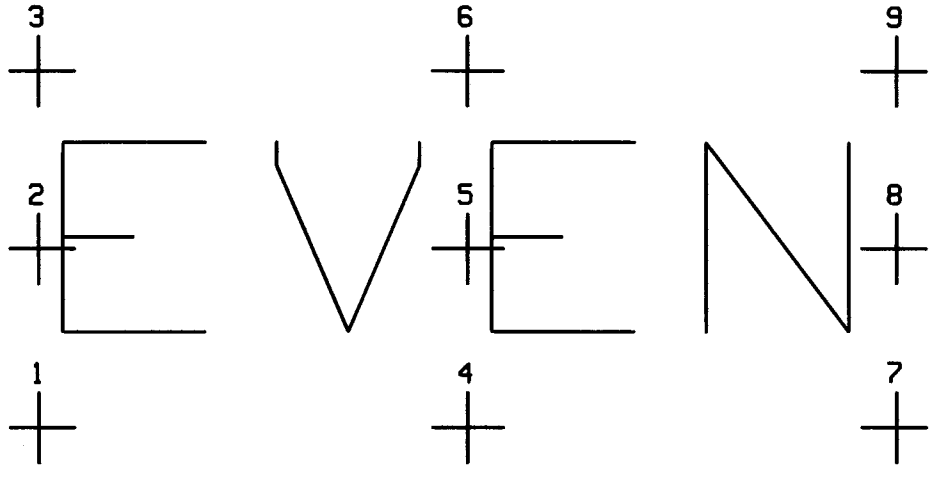
Item	Description/Default	Range Restrictions
label origin position	numeric expression, rounded to an integer; Default = 1	1 thru 9

Example Statements

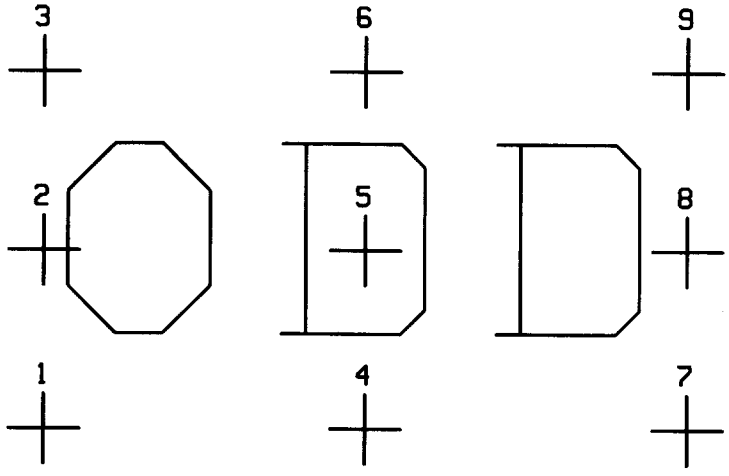
```
LORG 4
IF Y>Limit THEN LORG 3
```

Semantics

The following drawings show the relationship between a label and the logical pen position. The pen position before the label is drawn is represented by a cross marked with the appropriate LORG number.



Label Origins for Labels with an Even Number of Characters

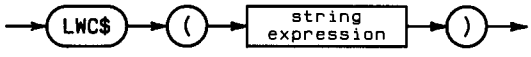


Label Origins for Labels with an Odd Number of Characters

LWC$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function replaces any uppercase characters with their corresponding lowercase characters.



Example Statements

```
Lower$=LWC$("UPPER")
IF LWC$(Yes$)="y" THEN True_test
```

Semantics

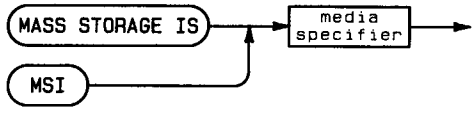
The LWC$ function converts only uppercase alphabetic characters to their corresponding lowercase characters and will not alter numerals or special characters.

The corresponding characters for the Roman Extension alphabetic characters are determined by the current lexical order. When the lexical order is a user-defined table, the correspondence is determined by the STANDARD lexical order.

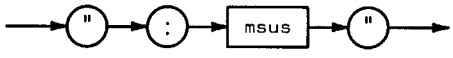
MASS STORAGE IS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

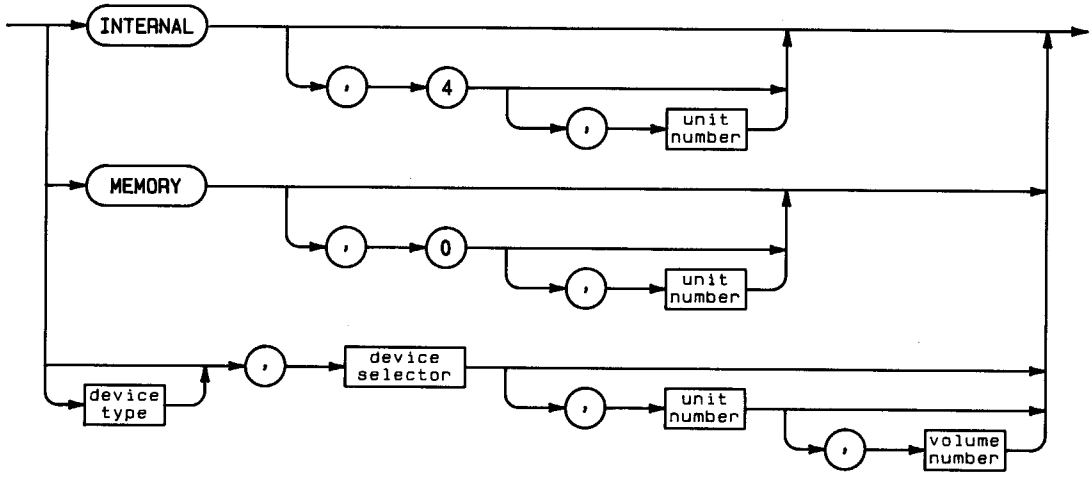
This statement specifies the system mass storage device. (If using MASS STORAGE IS with SRM, also refer to the "SRM" section of this manual.)



literal form of media specifier:



msus:



Item	Description/Default	Range Restrictions
media specifier	string expression	—
msus	literal	(see drawing)
device selector	integer constant	(see Glossary)
device type	literal	(see Sematics)
unit number	integer constant; Default = 0	0 thru 255 (device dependent)
volume number	integer constant; Default = 0	(device dependent)

Example Statements

```
MASS STORAGE IS ":INTERNAL,4,1"
MASS STORAGE IS ":X,12"
MASS STORAGE IS Msus$
```

Sematics

All mass storage operations which do not specify a source or destination by either an I/O path name or msus in the file specifier use the current system mass storage device.

MASS STORAGE IS can be abbreviated as MSI when entering a program line, but a program listing always shows the unabbreviated keywords.

Device Type

The following table shows the **valid** device types. Most device types require an option BIN for the statement to execute.

BIN Required	Device Type
none	INTERNAL MEMORY
DISC & HPIB or FHPIB	HP 9895 HP 9121 HP 9133 HP 9134 HP 9135 (5¼ inch uses HPIB not FHPIB) HP 913X
DISC & HPIB	HP 82901 HP 82902 HP 8290X
CS80 & HPIB or FHPIB	CS80 (7908, 7911, 7912, 7914, 9122...)
HP9885 & GPIO	HP 9885
SRM & DCOMM	REMOTE
BUBBLE	BUBBLE
EPROM	EPROM

Note the 98625 Card (which requires the FHPIB BIN) cannot be used with external 5¼ inch discs.

If the device type specified is not valid, the system tests the device to determine its type. There are two exceptions to this.

1. If the device selector is 0 and the device type is invalid, the device type is assummed to be MEMORY
2. If the device type is valid and the driver BIN for the device is not loaded, the system considers the device an invalid device type.

If a valid device type is specified and the system finds a different device at the device selector, error 72 occurs.

Non-Disc Mass Storage
Memory volumes are created by the INITIALIZE statement. They are removed by SCRATCH A or by turning off the power. The unit number for a MEMORY volume may be 0 thru 31.

A bubble memory card may have an select code of 8 thru 31. (Use of this card requires the BUBBLE BIN.) A bubble memory card is always unit number 0. It is recommended that these cards be given a high hardware-interrupt level to avoid error 314 in overlapped applications.

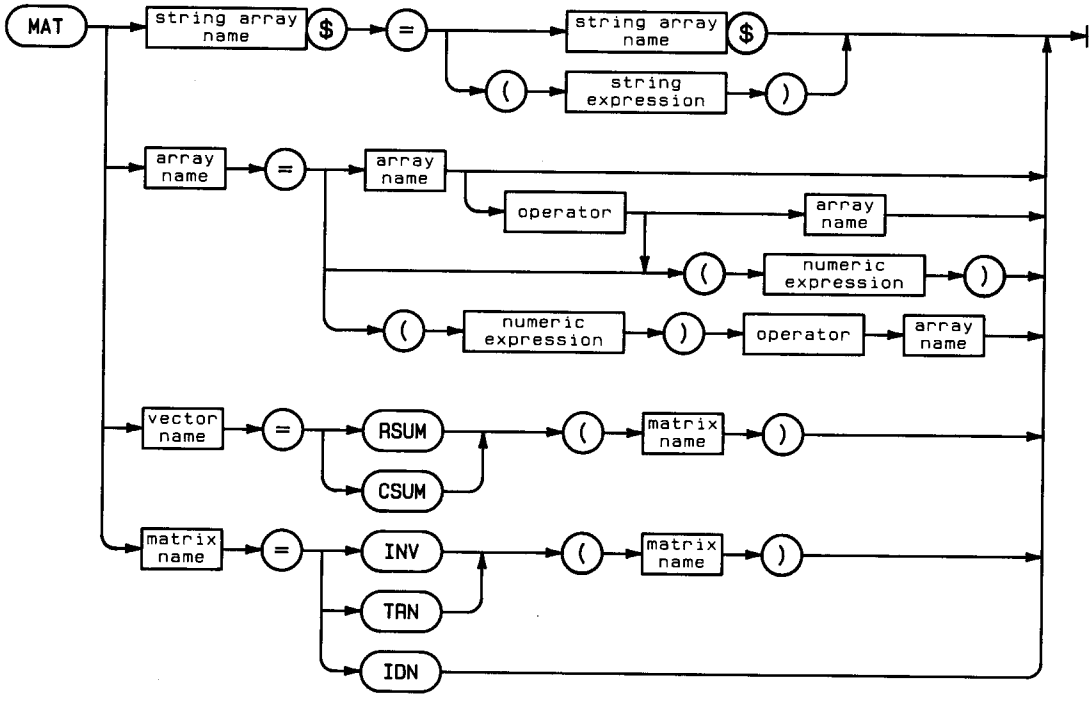
When writing data into EPROM (requires the EPROM BIN), specify the select code of the EPROM Programmer card that is connected to the desired EPROM memory card. When reading data from EPROM, specify a select code of 0 or use the select code of the currently-connected EPROM Programmer card. If the programmer card at the specified select code is not connected to the specified EPROM memory card, an error is reported. If the select code of 0 is used, you must specify "EPROM" in the mass storage unit specifier; otherwise, the system assumes MEMORY.

The unit numbers are given to the EPROM memory cards at power-up according to relative memory addresses. The card with the lowest address is given unit number 0, the card with the next greater address is given unit number 1, and so forth.

MAT

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

MAT can be used to initialize string and numeric arrays to constant values and copy string and numeric arrays. It can also be used to perform arithmetic operations on numeric arrays and, through the use of secondary keywords, can be used to perform special functions on numeric arrays.



Item	Description/Default	Range Restrictions
string array name	name of a string array	any valid name
array name	name of a numeric array	any valid name
operator	Any of the following: + - * / < <= = <> >= > *	* can only appear between two arrays
vector name	name of a one-dimensional numeric array	any valid name
matrix name	name of a two dimensional numeric array	any valid name

Example Statements

```
MAT A= A*(Ref+1/3)
MAT A= A+B
MAT A= B<(1)
MAT A= B<>C
MAT V= CSUM(A)
MAT I= IDN
MAT B= INV(A)
```

Semantics

The MAT statement allows you to:

- Copy a string expression into a string array or copy the contents of one string array into another string array.
- Copy a numeric expression into an array or the contents of one array into another array.
- Add an array and a numeric expression, or two arrays.
- Subtract a numeric expression from an array, an array from a numeric expression, or an array from an array.
- Multiply an array by a numeric expression or another array.
- Divide a numeric expression by an array, an array by a numeric expression, or an array by an array.
- Compare an array to a numeric expression or to another array.
- Calculate the Identity, Inverse, Transpose, Sum of rows and Sum of columns of a matrix.

Note

If an error occurs during the calculations involved in a MAT assignment the result array will contain only a partial result. Since you will have no idea which entries are valid, the contents of the array should be considered invalid.

Numeric Operations

In the case of operators, the specified operation is generally performed on every array element, and the results are placed in corresponding locations of the result array (the exception is the * operator, which is discussed under Matrix Multiplication, below.) This means that the result array must have the same size and shape (though not necessarily the same subscript ranges) as the operand array(s). If necessary, the system will redimension the result array to make it the proper size. The redimensioning can only take place, however, if the dimensioned size of the result array has at least as many elements as the current size of the operand array(s).

When two arrays are operated on, they must be exactly the same size and shape. If not, the computer returns an error. The specified operation is performed on corresponding elements in each operand array and the result is placed in the corresponding location of the result array. Multiplication of the elements of two arrays is performed with a period rather than an asterisk. The asterisk is reserved for matrix multiplication described below.

Relational Operators
Relational operations are performed on each element of the operand array(s). If the relation is TRUE, a 1 is placed in the corresponding location of the result array. If the relation is FALSE, a 0 is recorded. The result array, therefore, consists of all 0's and 1's.

Matrix Multiplication
The asterisk is used for two operations. If it is between an array and a numeric expression, each element in the array is multiplied by the numeric expression. If it is between two matrixes, it results in matrix multiplication. If **A** and **B** are the two operand matrices, and **C** is the result matrix, the matrix multiplication is defined by:

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

where n is the number of elements in a column in the matrix A. (This formula assumes that the array subscripts run from 1 through N; in actuality, the computer only requires that the two arrays be the correct size and shape, the actual values of the subscripts are unimportant.)

Note that the subscript values of the result array correspond to the rows of the first operand matrix and the columns of the second operand matrix. Note also that the column subscript of the first operand array is equal to the row subscript of the second operand array. We can summarize these observations in two general rules:

- The result matrix will have the same number of rows as the first operand matrix and the same number of columns as the second operand matrix.
- Matrix multiplication is legal if, and only if, the column size of the first operand matrix is equal to the row size of the second operand matrix.

A third rule of matrix multiplication is:

- The result matrix cannot be the same as either operand matrix.

The calculation is done in REAL math unless both operands are INTEGER, in which case the computation is also INTEGER. If the result matrix and the operand matrixes are different types (i.e., one is REAL and the others are INTEGER), the computer makes the conversion necessary for the assignment. However, the conversion is made **after** the multiplication is calculated, so even if the matrix receiving the result is REAL, the multiplication can generate an INTEGER overflow when the operands are INTEGER matrixes.

The computer allows you to do matrix multiplication on vectors by treating the vectors as if they were matrices. If the first operand is a vector, it is treated as a 1-by-N matrix. If the second operand is a vector, it is treated as an N-by-1 matrix

CSUM

This secondary keyword computes the sum of each column in a matrix and places the results in a vector. The result vector must have at least as many elements as the matrix has columns. If the vector is too large or its current size is too small (and there are enough elements in its original declaration to allow redimensioning), the computer redimensions it. If the result vector and the argument array are different types (i.e., one is REAL and the other is INTEGER), the computer makes the necessary conversion. However, the conversion is made **after** the column sums are calculated, so even if the vector receiving the result is REAL, CSUM can generate an INTEGER overflow when the argument is an INTEGER array.

IDN

This secondary keyword turns a square matrix into an identity matrix. An identity matrix has 1's along the main diagonal and 0's everywhere else. The matrix **must** be square.

INV

This secondary keyword finds the inverse of a square matrix. A matrix multiplied by its inverse produces an identity matrix. The inverse is found by using the pivot-point method. If the value of the determinant (see DET) is 0 after an INV, then the matrix has no inverse — whatever inverse the computer came up with is invalid. If the value of the determinant is very small compared with the elements in the argument matrix, then the inverse may be invalid and should be checked.

If the result matrix is not the same size and shape as the argument matrix, the computer will attempt to redimension it. If it is too large, or its current size is too small (and there are enough elements in its original declaration to allow redimensioning) the computer redimensions it. An error is returned if the computer cannot redimension the result array.

RSUM

This secondary keyword computes the sum of each row in a matrix and places the values in a vector. The result vector must be large enough to hold the sums of each row. If it is too large, or its current size is too small (and there are enough elements in its original declaration to allow redimensioning) the computer redimensions it. If the result vector and the argument array are different types (i.e., one is REAL and the other is INTEGER), the computer makes the necessary conversion. However, the conversion is made **after** the row sums are calculated, so even if the vector receiving the result is REAL, RSUM can generate an INTEGER overflow when the argument is an INTEGER array.

TRN

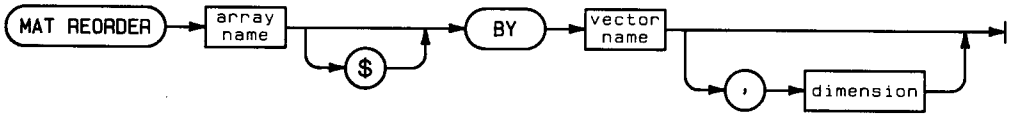
This secondary keyword produces the transpose of a matrix. The transpose is produced by exchanging rows for columns and columns for rows. The result matrix must be dimensioned to be at least as large as the current size of the argument matrix. If it's the wrong shape, the computer redimensions it. The result and argument matrices cannot be the same.

The transpose of an N-by-M matrix is an M-by-N matrix, and each element is defined by switching the subscripts. That is, **A**(m,n) in the argument matrix equals **B**(n,m) in the result matrix. (This description assumes that the array subscripts run from 1 through M and 1 through N; in actuality, the computer only requires that the array be the correct size and shape, the actual values of the subscripts are unimportant.)

MAT REORDER

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement reorders elements in an array according to the subscript list in a vector.



Item	Description/Default	Range Restrictions
array name	name of an array	any valid name
vector name	name of a one-dimensional numeric array	any valid name
dimension	numeric expression, rounded to an integer; Default = 1	1 thru 6; ≤ the RANK of the array

Example Statements

```
MAT REORDER A BY B
MAT REORDER A BY B,2
```

Semantics

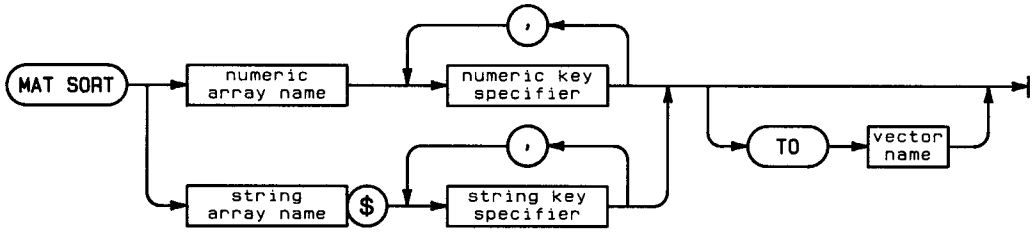
The *dimension* parameter is used to specify which dimension in a multidimensional array is to be reordered. If no dimension is specified, the computer defaults to dimension 1. The vector must be the same size as the specified dimension and it should contain integers corresponding to the subscript range of that dimension (no duplicate numbers, or numbers out of range).

Vectors generated by a MAT SORT TO statement are of the proper form for reordering (see MAT SORT).

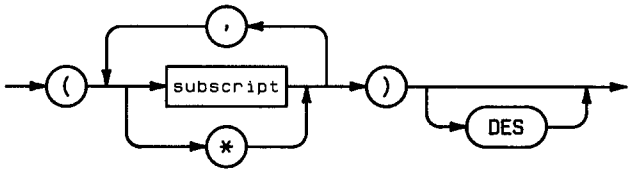
MAT SORT

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sorts an array along one dimension according to lexical or numeric order. In a string array, the current LEXICAL ORDER IS table is used for the sorting comparisons.

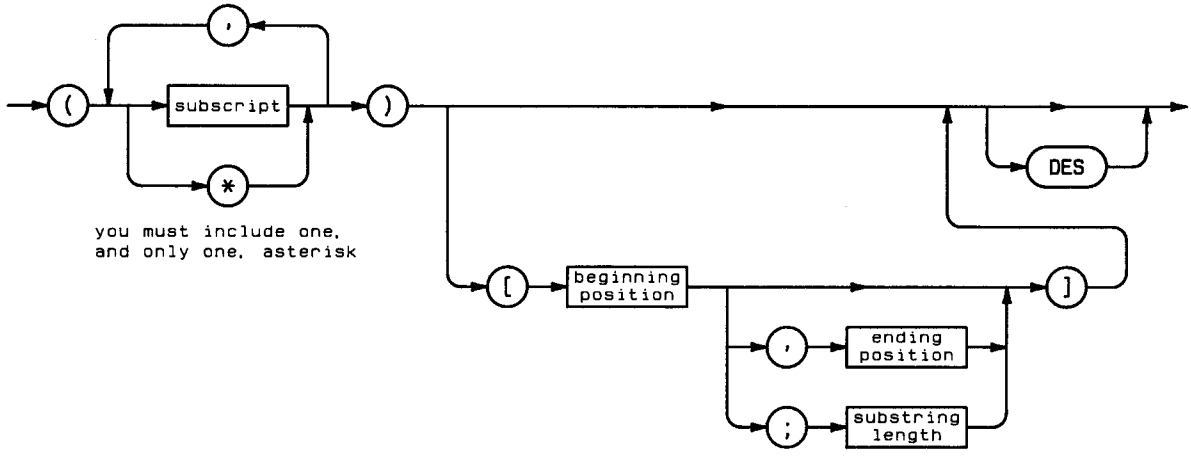


numeric key specifier:



you must include one, and only one, asterisk

string key specifier:



you must include one, and only one, asterisk

Item	Description/Default	Range Restrictions
numeric array name	name of a numeric array	any valid name
string array name	name of a string array	any valid name
vector name	name of a one-dimensional numeric array	any valid name
subscript	numeric expression, rounded to an integer	−32 768 thru +32 767 (see "array" in glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
MAT SORT A(1,*,3)
MAT SORT A(1,*,3),(2,*,5) DES
MAT SORT B(*) TO V
MAT SORT A$(3,*)[1;2] TO V
MAT SORT A$(*,2) DES,(*,3)[4,7]
```

Semantics

The elements to be compared are defined by a key specifier. The dimension to be sorted is marked with an asterisk, and the subscript values in the key specifier define which elements in that dimension should be used as the sorting values. Once `(*)` or `(*) DES` appears in the list following the array name, no other items can be added.

In the case of ties, the computer leaves the elements in their current order. However, you can define additional key specifiers to be used for ties. Whenever the computer encounters a tie, it will look to the next (moving from left to right) key specifier to break the tie. It will look at as many key specifiers as necessary to resolve the tie. In theory, there is no limit to the number of key specifiers you can have in one MAT SORT statement. In practice, it is limited by the length of a stored line on the computer you are dealing with. Each key must have an asterisk marking the same dimension.

Normally, the system sorts in ascending order. You can sort in descending order by using the secondary keyword DES. DES applies only to the key specifier which it follows. All others use the default ascending order.

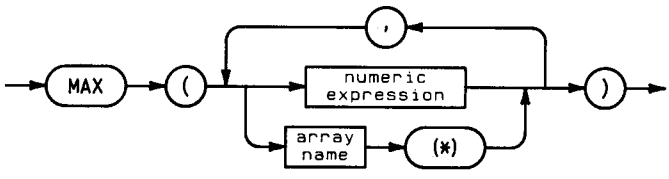
MAT SORT of string arrays allows you not only to define the elements to be sorted, but also to define substrings within each element. Substring specifiers refer only to the key specifier that immediately precedes them. Substrings may lie anywhere within the dimensioned size of the string. If a substring lies outside the current string length, the null string is used as the sorting key.

In addition to actually sorting an array, you can use MAT SORT...TO to store the new order in a vector and leave the original array intact. If the vector is too large, or its current size is too small (and there are enough elements in its original declaration to allow redimensioning) the computer redimensions it. After a MAT SORT TO statement, the array will be unchanged. The vector will contain the subscript values of the sorted dimension in their new order. You can then order the array or other parallel arrays using the REORDER statement. You can also use the contents of the vector to access the original array indirectly.

MAX

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value equal to the largest value in the list of arguments provided. If an array is specified as part of the list of arguments, it is equivalent to listing all the values in the array. An INTEGER is returned if and only if all arguments in the list are INTEGER.



Item	Description/Default	Range Restrictions
array name	name of a numeric array	any valid name

Example Statements

```
X=MAX(A(*))
X=MAX(A,3,B)
X=MAX(Floor,MIN(Ceiling,Argument))
```

Note

It is possible for the space needed for MAX to exceed the temporary storage allocated for expression evaluation. If the machine is close to overflowing memory this can be a **fatal** error and can crash the machine. It is recommended that statements including MAX not contain more than 20 variables and constants. An array is counted as one variable.

MAXREAL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the largest positive REAL number available in the range of the machine.



Example Statements

```
A=MAXREAL
IF A*B<MAXREAL THEN GOTO 100
```

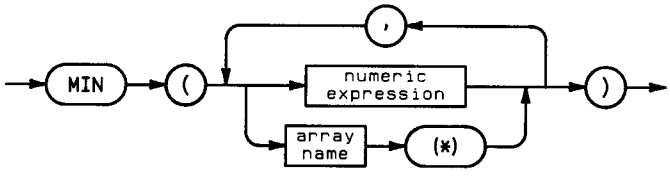
Semantics

The value of MAXREAL is approximately 1.797 693 134 862 32 E+308.

MIN

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value equal to the smallest value in the list of arguments provided. If an array is specified as part of the list of arguments, it is equivalent to listing all the values in the array. An INTEGER is returned if and only if all arguments in the list are INTEGER.



Item	Description/Default	Range Restrictions
array name	name of a numeric array	any valid name

Example Statements

```
X=MIN(A(*))
X=MIN(A,3,B)
X=MIN(Ceiling,MAX(Floor,Argument))
```

Note

It is possible for the space needed for MIN to exceed the temporary storage allocated for expression evaluation. If the machine is close to overflowing memory this can be a **fatal** error and can crash the machine. It is recommended that statements including MIN not contain more than 20 variables and constants. An array is counted as one variable.

MINREAL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the smallest positive REAL number available in the range of the computer.



Example Statements

```
A=MINREAL
IF A-B>MINREAL THEN GOTO 100
```

Semantics

The value of MINREAL is approximately 2.225 073 858 507 2 4E – 308

MLA

See the SEND statement.

MOD

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns the remainder of a division.



Item	Description/Default	Range Restrictions
dividend	numeric expression	—
divisor	numeric expression	not equal to 0

Example Statements

```
Remainder=Dividend MOD Divisor
PRINT "Seconds =";Time MOD 60
```

Semantics

MOD returns an INTEGER value if both arguments are INTEGER. Otherwise the returned value is REAL.

For INTEGERs, MOD is equivalent to X − Y × (X DIV Y).This may return a different result from the modulus function on other computers when negative numbers are involved.

MODULO

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns the integer remainder resulting from a division.



Item	Description/Default	Range Restrictions
dividend	numeric expression	range of REAL
modulus	numeric expression	range of REAL, ≠0

Example Statements

```
Remainder=Dividend MODULO Modulus
A=B MODULO C
```

Semantics

X MODULO Y is equivalent to $X - Y \times INT(X/Y)$.

The result satisfies:

$0 <= (X \text{ MODULO } Y) < Y$ if $Y>0$
$Y < (X \text{ MODULO } Y) <= 0$ if $Y<0$

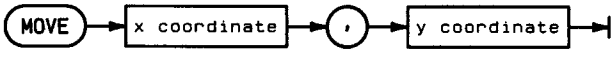
The type of the result is the higher of the types of the two operands. If the modulus is zero error 31 occurs.

MODULO returns the remainder of a division.

MOVE

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement moves both the logical and physical pens from the current pen position to the specified X and Y coordinates.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression in current units	—
y coordinate	numeric expression in current units	—

Example Statements

```
MOVE 10,75
MOVE Next_x,Next_y
```

Semantics

The X and Y coordinates are interpreted according to the current unit-of-measure. MOVE is affected by the PIVOT transformation.

If both current physical pen position and specified pen position are outside current clip limits, no physical pen movement is made; however, the logical pen position is moved to the specified coordinates.

Applicable Graphics Transformations

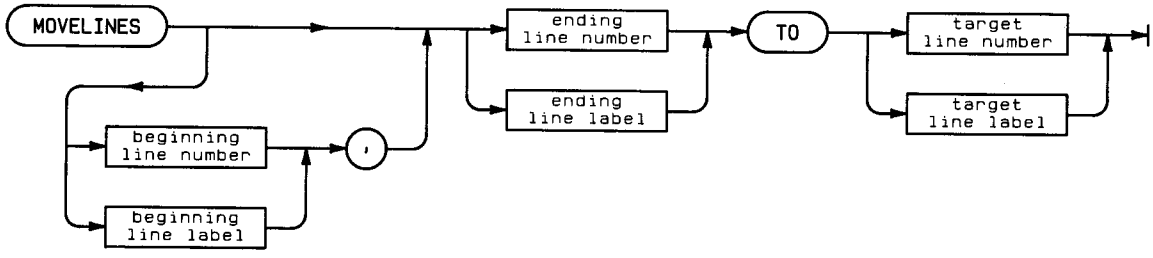
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

MOVELINES

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN	No

This command allows you to move one or more program lines to another place while editing a program.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying program line	1 to 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line	1 to 32 766
ending line label	name of a program line	any valid name
target line number	integer constant identifying program line	1 to 32 766
target line label	name of a program line	any valid name

Example Statements

```
MOVELINES 1200 TO 2350
MOVELINES 100,230 TO Label1
MOVELINES Util_start,Util_end TO 16340
```

Semantics

If the ending line identifier is not specified, only one line is moved.

The target line identifier will be the line number of the first line of the moved program segment. Moved lines are renumbered if necessary. The code (if any) which is "pushed down" to make room for the moved code is renumbered if necessary.

Line number references to the moved code are updated as they would be by a REN command (except external references to non-existent lines are renumbered).

If there are any DEF FN or SUB statements in the moved code, the target line number must be greater than any existing line number.

If you try to move a program segment to a line number **contained** in the segment, an error will result and no moving will occur.

If the starting line number does not exist, the next line is used. If the ending line number does not exist, the previous line is used. If a line label doesn't exist, an error occurs and no moving takes place.

If an error occurs *during* a MOVELINES (for example, a memory overflow), the move is terminated and the program is left partially modified.

MSI

See the MASS STORAGE IS statement.

MTA

See the SEND statement.

NEXT

See the FOR...NEXT construct.

NOT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns 1 if its argument equals 0. Otherwise, 0 is returned.



Example Statements

```
Invert_flag=NOT Std_device
IF NOT Pointer THEN Next_op
```

Semantics

When evaluating the argument, a non-zero value (positive or negative) is treated as a logical 1; only zero is treated as a logical 0.

The logical complement is shown below:

A	NOT A
0	1
1	0

NPAR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the number of parameters passed to the current subprogram. If execution is currently in the main program, NPAR returns 0.



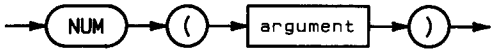
Example Statements

```
IF NPAR>3 THEN Extra
Factors=NPAR-2
```

NUM

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the decimal value of the ASCII code of the first character in the argument. The range of returned values is 0 thru 255.



Item	Description/Default	Range Restrictions
argument	string expression	not a null string

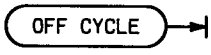
Example Statements

```
Letter=NUM(String$)
A$[I;1]=CHR$(NUM(A$[I])+32)
```

OFF CYCLE

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON CYCLE statement.



Example Statements

```
OFF CYCLE
IF Kick_stand THEN OFF CYCLE
```

Semantics

OFF CYCLE destroys the log of any CYCLE event which has already occurred but which has not been serviced.

If OFF CYCLE is executed in a subprogram such that it cancels an ON CYCLE in the calling context, the ON CYCLE definition is restored upon returning to the calling context.

OFF DELAY

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON DELAY statement.



Example Statements

```
OFF DELAY
IF Ready THEN OFF DELAY
```

Semantics

OFF DELAY destroys the log of any DELAY event which has already occurred but which has not been serviced.

If OFF DELAY is executed in a subprogram such that it cancels an ON DELAY in the calling context, the ON DELAY definition is restored upon returning to the calling context.

OFF END

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously enabled and defined by an ON END statement.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)

Example Statements

```
OFF END @File
IF Special THEN OFF END @Source
```

Semantics

If OFF END is executed in a subprogram and cancels an ON END in the context which called the subprogram, the ON END definitions are restored when the calling context is restored.

If there is no ON END definition in a context, end-of-file and end-of-record are reported as errors.

OFF EOR

Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON EOR statement.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name

Example Statements

```
OFF EOR @File
OFF EOR @Device_selector
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file; however, if the I/O path is assigned to a BUFFER, an error is reported when the OFF EOR statement is executed.

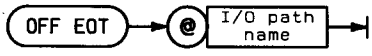
OFF EOR destroys the log of any EOR event which has already occurred but which has not been serviced.

If OFF EOR is executed in a subprogram such that it cancels an ON EOR in the calling context, the ON EOR definition is restored upon returning to the calling context.

OFF EOT

Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON EOT statement.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name

Example Statements

```
OFF EOT @File
IF Done_flag THEN OFF EOT @Info
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file; however, if the I/O path is assigned to a BUFFER, an error is reported when the OFF EOT statement is executed.

OFF EOT destroys the log of any EOT event which has already occurred but which has not been serviced.

If OFF EOT is executed in a subprogram such that it cancels an ON EOT in the calling context, the ON EOT definition is restored upon returning to the calling context.

OFF ERROR

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

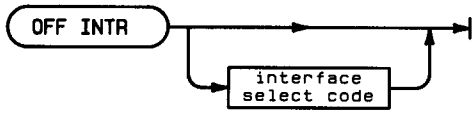
This statement cancels event-initiated branches previously defined and enabled by an ON ERROR statement. Further errors are reported to the user in the usual fashion.



OFF INTR

Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined by an ON INTR statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer; Default = all interfaces	5, and 7 thru 31

Example Statements

```
OFF INTR
OFF INTR Hpib
```

Semantics

Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the OFF INTR to apply to the event-initiated log entry for the specified interface only.

Any pending ON INTR branches for the effected interfaces are lost and further interrupts are ignored.

OFF KBD

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels the event-initiated branch previously defined by an ON KBD statement.



Example Statements

```
OFF KBD
IF NOT Process_Keys THEN OFF KBD
```

Semantics

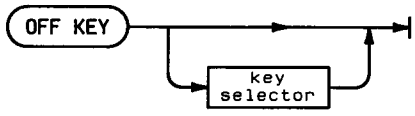
When this statement is executed, any pending ON KBD branch is cancelled, and the keyboard buffer is cleared.

If OFF KBD is executed in a subprogram such that it cancels an ON KBD in the calling context, the cancelled ON KBD definition is restored when the calling context is restored. However, the keyboard buffer's contents are not restored with the calling context, because the buffer was cleared with the OFF KBD.

OFF KEY

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON KEY statement.



Item	Description/Default	Range Restrictions
key selector	numeric expression, rounded to an integer; Default = all keys	0 thru 19

Example Statements

```
OFF KEY
OFF KEY 4
```

Semantics

Not specifying a softkey number disables the event-initiated branches for all softkeys. Specifying a softkey number causes the OFF KEY to apply to the specified softkey only. If OFF KEY is executed in a subprogram and cancels an ON KEY in the context which called the subprogram, the ON KEY definitions are restored when the calling context is restored.

Any pending ON KEY branches for the effected softkeys are lost. Pressing an undefined softkey generates a beep.

OFF KNOB

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

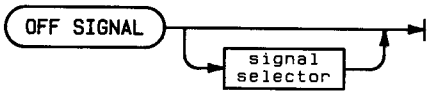
This statement cancels event-initiated branches previously defined and enabled by the ON KNOB statement. Any pending ON KNOB branches are lost. Further use of the knob will result in normal scrolling or cursor movement.



OFF SIGNAL

Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

OFF SIGNAL cancels the ON SIGNAL definition with the same signal selector. If no signal selector is provided, all ON SIGNAL definitions are cancelled. OFF SIGNAL only applies to the current context.



Item	Description/Default	Range Restrictions
signal selector	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
OFF SIGNAL
OFF SIGNAL 15
```

OFF TIME

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON TIME statement.



Example Statements

```
OFF TIME
IF Attended THEN OFF TIME
```

Semantics

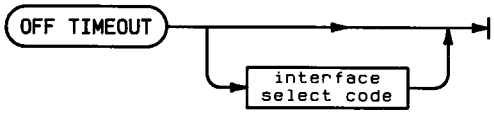
OFF TIME destroys the log of any TIME event which has already occurred but which has not been serviced.

If OFF TIME is executed in a subprogram such that it cancels an ON TIME in the calling context, the ON TIME definition is restored upon returning to the calling context.

OFF TIMEOUT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON TIMEOUT statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer; Default = all interfaces	7 thru 31

Example Statements

```
OFF TIMEOUT
OFF TIMEOUT Isc
```

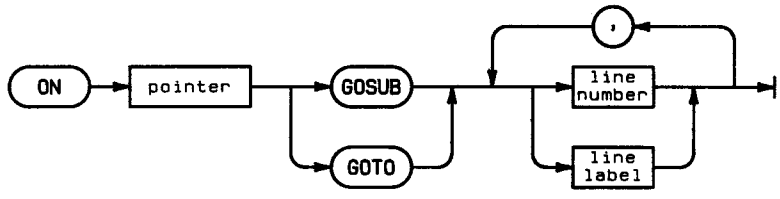
Semantics

Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the ON TIMEOUT to apply to the event-initiated branches for the specified interface only. When OFF TIMEOUT is executed, no more timeouts can occur on the effected interfaces.

ON

Option Required	None
Keyboard Executable	No
Programmabel	Yes
In an IF...THEN...	Yes

This statement transfers program execution to one of several destinations selected by the value of the pointer.



Item	Description/Default	Range Restrictions
pointer	numeric expression, rounded to an integer	1 thru 74
line number	integer constant identifying a program line	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
ON X1 GOTO 100,150,170
IF Point THEN ON Point GOSUB First,Second,Third,Last
```

Semantics

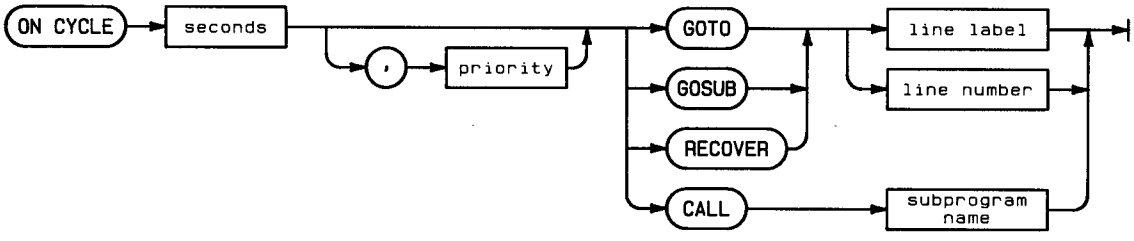
If the pointer is 1, the first line number or label is used. If the pointer is 2, the second line identifier is used, and so on. If GOSUB is used, the RETURN is to the line following the ON...GOSUB statement.

If the pointer is less than 1 or greater than the number of line labels or numbers, error 19 is generated. The specified line numbers or line labels must be in the same context as the ON statement.

ON CYCLE

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken each time the specified number of seconds has elapsed.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest 0.01 second	0.01 thru 167 772.16
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON CYCLE 1 GOSUB One_second
ON CYCLE 3600,12 CALL Chime
```

Semantics

The most recent ON CYCLE (or OFF CYCLE) definition overrides any previous ON CYCLE definition. If the overriding ON CYCLE definition occurs in a context different from the one in which the overridden ON CYCLE occurs, the overridden ON CYCLE is restored when the calling context is restored, but the time value of the more recent ON CYCLE remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON CYCLE can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON CYCLE priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON CYCLE statement. CALL and GOSUB will return to the next line that would have been executed if the CYCLE event had not been serviced, and the system priority is restored to that which existed before the ON CYCLE branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON CYCLE statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

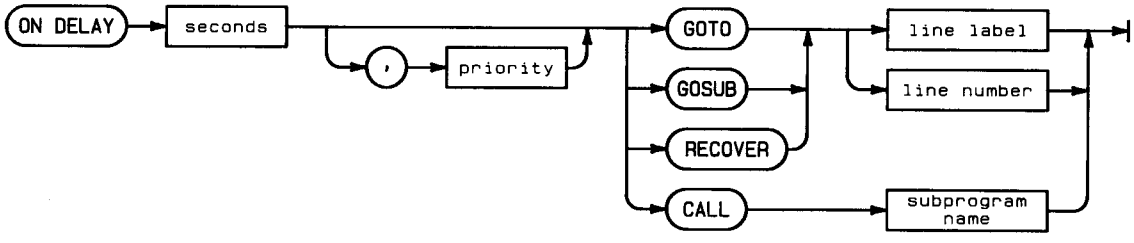
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON CYCLE is disabled by DISABLE and deactivated by OFF CYCLE. If the cycle value is short enough that the computer cannot service it, the interrupt will be lost.

ON DELAY

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken after the specified number of seconds has elapsed.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest 0.01 second	0.01 thru 167 772.16
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Examples

```
ON DELAY 10 GOTO Default
ON DELAY 3,2 GOSUB Low_level
```

Semantics

The most recent ON DELAY (or OFF DELAY) definition overrides any previous ON DELAY definition. If the overriding ON DELAY definition occurs in a context different from the one in which the overridden ON DELAY occurs, the overridden ON DELAY is restored when the calling context is restored, but the time value of the more recent ON DELAY remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON DELAY can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON DELAY priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON DELAY statement. CALL and GOSUB will return to the next line that would have been executed if the DELAY event had not been serviced, and the system priority is restored to that which existed before the ON DELAY branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON DELAY statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

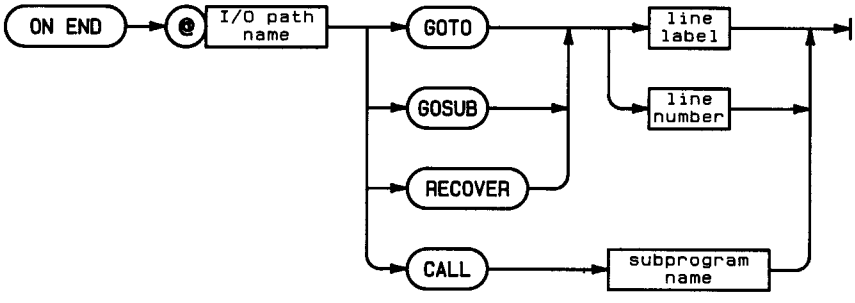
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON DELAY is disabled by DISABLE and deactivated by OFF DELAY.

ON END

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when end-of-file is reached on the mass storage file associated with the specified I/O path.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON END @Source GOTO Next_file
ON END @Dest CALL Expand
```

Semantics

The ON END branch is triggered by any of the following events:

- When the physical end-of-file is encountered.
- When an ENTER statement reads the byte at EOF or beyond.
- When a random access OUTPUT requires more than one defined record.
- When a random access OUTPUT is attempted beyond the next available record. (If EOF is the first byte of a record, then that record is the next available record. If EOF is not at the first byte of a record, the following record is the next available record.)

The priority associated with ON END is higher than priority 15. ON TIMEOUT and ON ERROR have the same priority as ON END, and can interrupt an ON END service routine.

Any specified line label or line number must be in the same context as the ON END statement. CALL and GOSUB will return to the line immediately following the one during which the end-of-file occurred, and the system priority is restored to that which existed before the ON END branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON END statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context was exited.

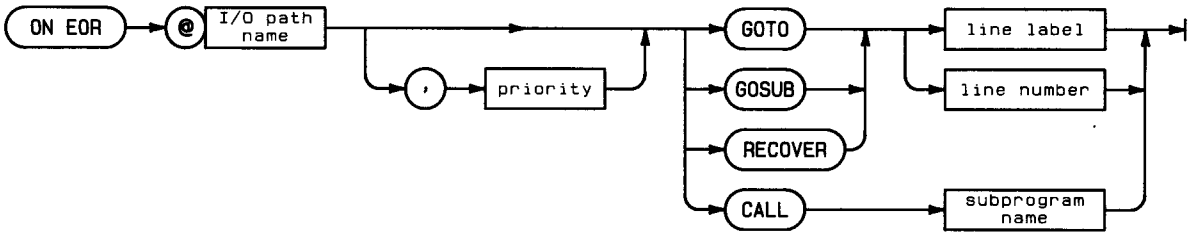
CALL and RECOVER remain active when the context changes to a subprogram, if the I/O path name is known in the new context. CALL and RECOVER do not remain active if the context changes as a result of a keyboard-originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram.

The end-of-record error (error 60) or the end-of-file error (error 59) can be trapped by ON ERROR if ON END is not active. ON END is deactivated by OFF END. DISABLE does not affect ON END.

ON EOR

Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when an end-of-record is encountered during a TRANSFER.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON EOR @Gpio GOSUB Gpio_eor
ON EOR @Hpib,9 CALL Eor_sensed
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the ON EOR statement is executed.

If a TRANSFER statement uses an I/O path name which is local to a subprogram and the TRANSFER has not completed by the time the context is exited, returning to the original context will be deferred until the end of the TRANSFER; at that time the ON EOR event cannot be serviced. To ensure that the event will be serviced, a statement that cannot be executed in overlap with the TRANSFER must be executed before the context is exited. A WAIT FOR EOR @Non_buf statement is used for this purpose.

End-of-record delimiters are defined by the EOR parameters of the TRANSFER statement (i.e., DELIM, COUNT, or END). An EOR event occurs when any of the specified end-of-record delimiters is encountered during a TRANSFER. The event's occurrence is logged, and the specified branch is taken when system priority permits.

The most recent ON EOR (or OFF EOR) definition for a given I/O path name overrides any previous ON EOR definition. If the overriding ON EOR definition occurs in a context different from the one in which the overridden ON EOR occurs, the overridden ON EOR is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON EOR can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON EOR priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON EOR statement. CALL and GOSUB will return to the next line that would have been executed if the EOR event had not been serviced, and the system priority is restored to that which existed before the ON EOR branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON EOR statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

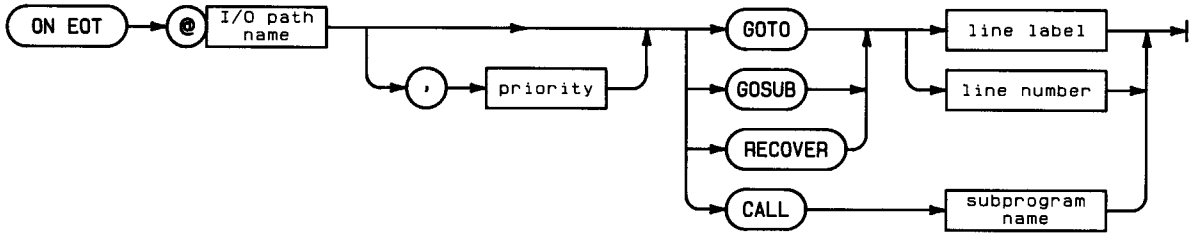
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EOR is disabled by DISABLE and deactivated by OFF EOR.

ON EOT

Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when the last byte is transferred by a TRANSFER statement.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON EOT @File GOTO Finished
ON EOT @Hpib,5 CALL More
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the ON EOT statement is executed.

If a TRANSFER statement uses an I/O path name which is local to a subprogram and the TRANSFER has not completed by the time the context is exited, returning to the original context will be deferred until the end of the TRANSFER; at that time the ON EOT event cannot be serviced. To ensure that the event will be serviced, a statement that cannot be executed in overlap with the TRANSFER must be executed before leaving the context. A WAIT FOR EOT @Non_buf statement is used for this purpose.

The most recent ON EOT (or OFF EOT) definition for a given path name overrides any previous ON EOT definition. If the overriding ON EOT definition occurs in a context different from the one in which the overridden ON EOT occurs, the overridden ON EOT is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON EOT can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON EOT priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON EOT statement. CALL and GOSUB will return to the next line that would have been executed if the EOT event had not been serviced, and the system priority is restored to that which existed before the ON EOT branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON EOT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

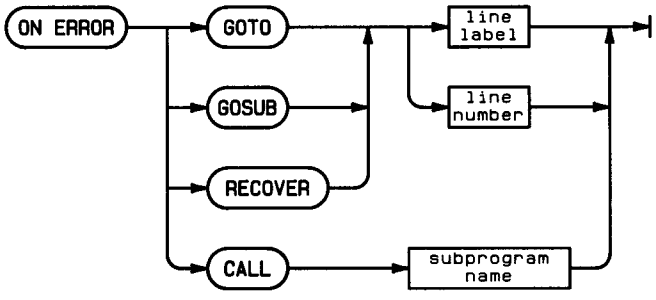
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EOT is disabled by DISABLE and deactivated by OFF EOT.

ON ERROR

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch which results from a trappable error. This allows you to write your own error handling routines.



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON ERROR GOTO 1200
ON ERROR CALL Report
```

Semantics

The ON ERROR statement has the highest priority of any event-initiated branch. ON ERROR can interrupt any event-initiated service routine.

Any specified line label or line number must be in the same context as the ON ERROR statement. RECOVER forces the program to go directly to the specified line in the context containing the ON ERROR statement.

Returns from ON ERROR GOSUB or ON ERROR CALL routines are different from regular GOSUB or CALL returns. When ON ERROR is in effect, the program resumes at the beginning of the line where the error occurred. If the ON ERROR routine did not correct the cause of the error, the error is repeated. This causes an infinite loop between the line in error and the error handling routine. When execution returns from the ON ERROR routine, system priority is restored to that which existed before the ON ERROR branch was taken.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. In this case, the error is reported to the user, as if ON ERROR had not been executed.

GOSUB and GOTO do not remain active when the context changes to a subprogram. If an error occurs, the error is reported to the user, as if ON ERROR had not been executed.

If an execution error occurs while servicing an ON ERROR CALL or ON ERROR GOSUB, program execution stops. If an execution error occurs while servicing an ON ERROR GOTO or ON ERROR RECOVER routine, an infinite loop can occur between the line in error and the GOTO or RECOVER routine.

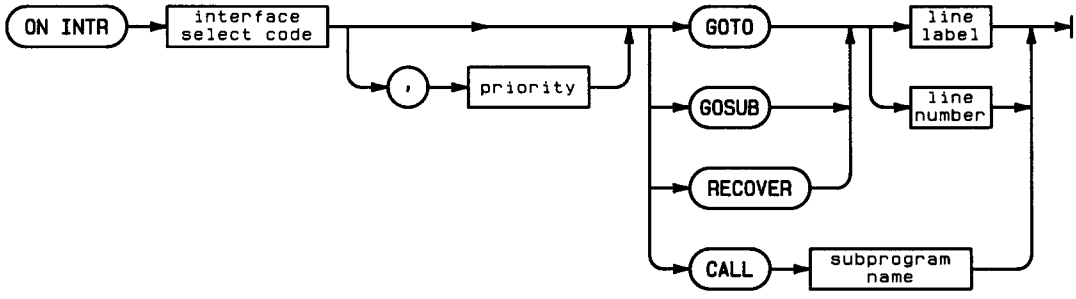
If an ON ERROR routine cannot be serviced because inadequate memory is available for the computer, the original error is reported and program execution pauses at that point.

ON ERROR is deactivated by OFF ERROR. DISABLE does not affect ON ERROR.

ON INTR

Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines an event-initiated branch to be taken when an interface card generates an interrupt. The interrupts must be explicitly enabled with an ENABLE INTR statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	5, 7 thru 31
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON INTR 7 GOSUB 500
ON INTR Isc,4 CALL Service
```

Semantics

The occurrence of an interrupt performs an implicit DISABLE INTR for the interface. An ENABLE INTR must be performed to re-enable the interface for subsequent event-initiated branches. Another ON INTR is not required, nor must the mask for ENABLE INTR be redefined.

The priority can be specified, with highest priority represented by 15. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON INTR can interrupt service routines of other event-initiated branches which have user-definable priorities, if the ON INTR priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON INTR statement. CALL and GOSUB will return to the next line that would have been executed if the INTR event had not been serviced, and the system priority is restored to that which existed before the ON INTR branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON INTR statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

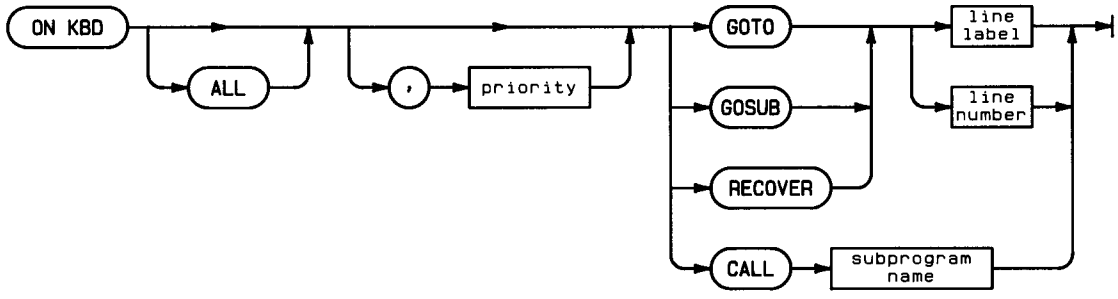
ON INTR is disabled by DISABLE INTR or DISABLE and deactivated by OFF INTR.

ON INTR and OFF INTR statements may be executed for **any** I/O card in the machine. It is not necessary to have a driver for the card.

ON KBD

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when a key is pressed.



Item	Description/Default	Range Restrictions
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KBD GOSUB 770
ON KBD,9 CALL Get_Key
```

Semantics

Specifying the secondary keyword ALL causes all keys except RESET, SHIFT, and CTRL to be trapped. When ALL is omitted, the untrapped keys are those just mentioned, the softkeys, PAUSE, STOP, CLR I/O, BREAK, System, User, Menu, and SHIFT Menu. When not trapped, these keys perform their normal functions. When the softkeys are trapped, ON KBD branching overrides any ON KEY branching.

A keystroke triggers a keyboard interrupt and initiates a branch to the specified routine when priority allows. If keystrokes occur while branching is held off by priority, the keystrokes are stored in a special buffer. When keystrokes are in the buffer, branching will occur when priority allows. This buffer is read and cleared by the KBD$ function (see the KBD$ entry).

Knob rotation will generate ON KBD interrupts unless an ON KNOB statement has been executed. Clockwise rotation of the knob produces right-arrow keystrokes; counterclockwise rotation produces left-arrow keystokes. If the (SHIFT) key is pressed while turning the knob then clockwise rotation of the knob produces up-arrow keystrokes; counterclockwise rotation produces down-arrow key strokes. Since one rotation of the knob is equivalent to 20 keystrokes, keyboard buffer overflow may occur if the BASIC service routine does not process the keys rapidly.

Live keyboard, editing, and display control functions are suspended during ON KBD. To restore a key's normal function the keystroke may be OUTPUT to select code 2.

The most recent ON KBD (or OFF KBD) definition overrides any previous ON KBD definition. If the overriding ON KBD definition occurs in a context different from the one in which the overridden ON KBD occurs, the overridden ON KBD is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON KBD can interrupt sevice routines of other event-initiated branches with user-definable priorities, if the ON KBD priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KBD statement. CALL and GOSUB will return to the next line that would have been executed if the KBD event had not been serviced, and the system priority is restored to that which existed before the ON KBD branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KBD statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

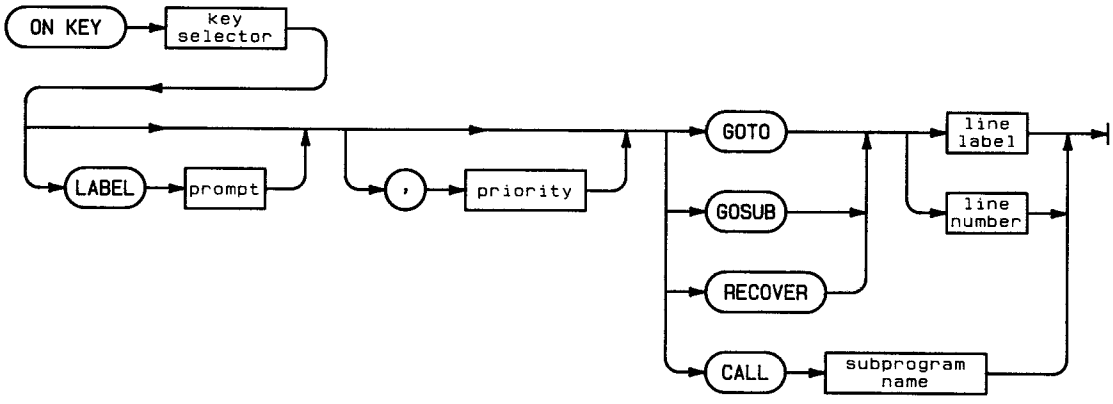
ON KBD is disabled by DISABLE, deactivated by OFF KBD, and temporarily deactivated when the program is executing LINPUT, INPUT, or ENTER 2.

You can use a relative pointing device, such as the HP 46060A on an HP 46020A keyboard, if the KBD BIN is present.

ON KEY

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when a softkey is pressed.



Item	Description/Default	Range Restrictions
key selector	numeric expression, rounded to an integer	0 thru 23
prompt	string expression; Default = no label	—
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KEY 0 GOTO 150
ON KEY 5 LABEL "Print",3 GOSUB Report
```

Semantics

The most recently executed ON KEY (or OFF KEY) definition for a particular softkey overrides any previous key definition. If the overriding ON KEY definition occurs in a context different from the one in which the overridden ON KEY occurs, the overridden ON KEY is restored when the calling context is restored.

Labels appear in the two bottom lines of the CRT. The label of any key is bound to the current ON KEY definition. Therefore, when a definition is changed or restored, the label changes accordingly. If no label is specified, that label field is blank. Refer to the BASIC Programming Techniques manual for a discussion of these labels.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). On KEY can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON KEY priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KEY statement. CALL and GOSUB will return to the next line that would have been executed if the KEY event had not been serviced, and the system priority is restored to that which existed before the ON KEY branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KEY statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

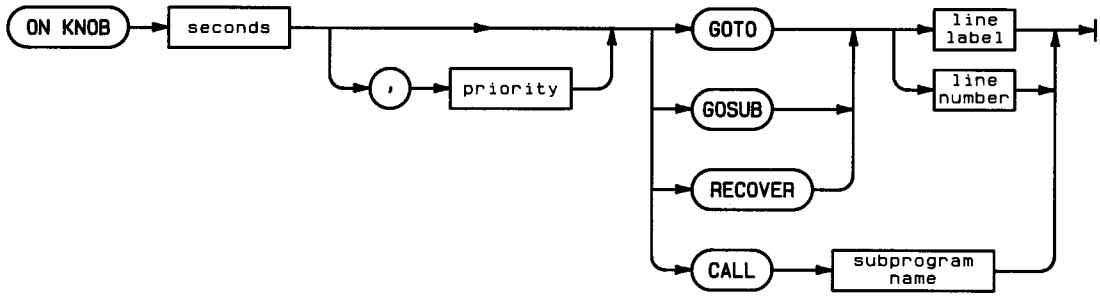
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON KEY is disabled by DISABLE, deactivated by OFF KEY, and temporarily deactivated when the program is paused or executing LINPUT, INPUT, or ENTER 2.

ON KNOB

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when the knob is turned.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest 0.01 second	0.01 thru 2.55
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KNOB .1 GOSUB 250
ON KNOB .333,Priority CALL Pulses
```

Semantics

Turning the knob (cursor wheel) generates pulses. After ON KNOB is activated (or re-activated), the first pulse received starts a sampling interval. The "seconds" parameter establishes the length of that sampling interval. At the end of the sampling interval, the ON KNOB branch is taken if the net number of pulses received during the interval is not zero and priority permits. The KNOBX and KNOBY functions can be used to determine the number of pulses received during the interval. If the ON KNOB branch is held off for any reason, the KNOBX and KNOBY functions accumulate the pulses (see KNOBX and KNOBY).

The most recent ON KNOB (or OFF KNOB) definition overrides any previous ON KNOB definition. If the overriding ON KNOB definition occurs in a context different from the one in which the overridden ON KNOB occurs, the overridden ON KNOB is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON KNOB can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON KNOB priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KNOB statement. CALL and GOSUB will return to the next line that would have been executed if the KNOB event had not been serviced, and the system priority is restored to that which existed before the ON KNOB branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KNOB statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

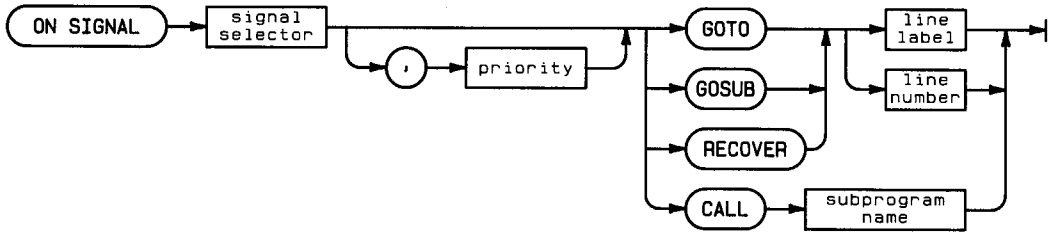
ON KNOB is disabled by DISABLE and deactivated by OFF KNOB.

You can use a relative pointing device, such as the HP 46060A, on an HP 46020A keyboard, if the KBD option is loaded.

ON SIGNAL

Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when a SIGNAL statement with the same signal selector is executed.



Item	Description/Default	Range Restrictions
signal selector	numeric expression, rounded to an integer	0 thru 15
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
suprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON SIGNAL 5 GOSUB 550
ON SIGNAL Bailout,15 RECOVER Bail_here
```

Semantics

The most recent ON SIGNAL (or OFF SIGNAL) definition for a given signal selector overrides any previous ON SIGNAL definition. If the overriding ON SIGNAL definition occurs in a context different from the one in which the overridden ON SIGNAL occurs, the overridden ON SIGNAL is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON SIGNAL can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON SIGNAL priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON SIGNAL statement. CALL and GOSUB will return to the next line that would have been executed if the SIGNAL event had not been serviced, and the system priority is restored to that which existed before the ON SIGNAL branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON SIGNAL statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

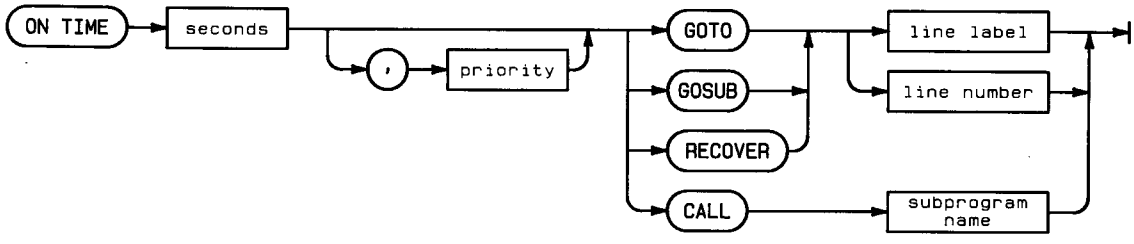
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON SIGNAL is disabled by DISABLE and deactivated by OFF SIGNAL.

ON TIME

Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when the real-time clock reaches a specified time.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest 0.01 second	0 thru 86 399.99
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
suprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON TIME 3600*8 GOTO Work
ON TIME (TIMEDATE+3600) MOD 86400 CALL One_hour
```

Semantics

The most recent ON TIME (or OFF TIME) definition overrides any previous ON TIME definition. If the overriding ON TIME definition occurs in a context different from the one in which the overridden ON TIME occurs, the overridden ON TIME is restored when the calling context is restored, but the time value of the more recent ON TIME remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON TIME can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON TIME priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

CALL and GOSUB will return to the next line that would have been executed if the TIME event had not been serviced, and the system priority is restored to that which existed before the ON TIME branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON TIME statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

Any specified line label or line number must be in the same context as the ON TIME statement. CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

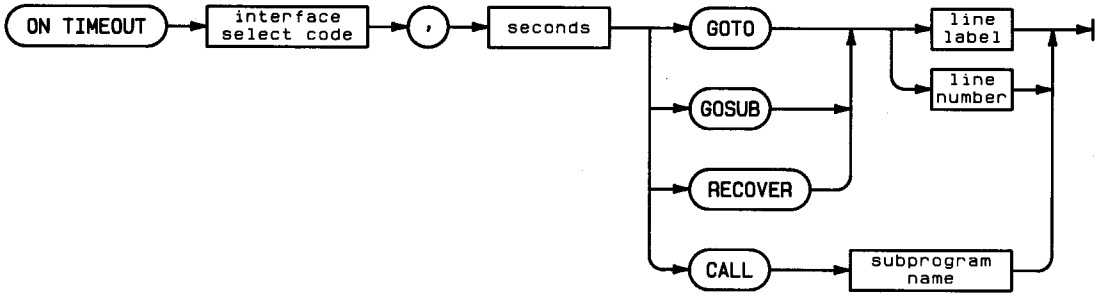
Unlike ON CYCLE, an ON TIME statement requires an exact match between the clock and the time specified in the defining statement. If the event was missed and not logged, re-executing the ON TIME statement will not result in a branch being taken.

ON TIME is disabled by DISABLE and deactivated by OFF TIME.

ON TIMEOUT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when an I/O timeout occurs on the specified interface. (If using ON TIMEOUT with SRM, also refer to the "SRM" section of this manual.)



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
seconds	numeric expression, rounded to the nearest 0.001 second	0.001 thru 32.767
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON TIMEOUT 7 GOTO 770
ON TIMEOUT Printer,Time GOSUB Message
```

Semantics

There is no default system timeout. If ON TIMEOUT is not in effect for an interface, a device can cause the program to wait forever.

The specified branch occurs if an input or output is active on the interface and the interface has not responded within the number of seconds specified. The computer waits at least the specified time before generating an interrupt; however, it may wait up to an additional 25% of the specified time.

Timeouts apply to ENTER and OUTPUT statements, and operations involving the PRINTER IS, PRINTALL IS, and PLOTTER IS devices when they are external. Timeouts do not apply to CONTROL, STATUS, READIO, WRITEIO, CRT alpha or graphics I/O, real time clock I/O, keyboard I/O, or mass storage operations.

The priority associated with ON TIMEOUT is higher than priority 15. ON END and ON ERROR have the same priority as ON TIMEOUT, and can interrupt an ON TIMEOUT service routine.

Any specified line label or line number must be in the same context as the ON TIMEOUT statement. CALL and GOSUB will return to the line immediately following the one during which the timeout occurred, and the system priority is restored to that which existed before the ON TIMEOUT branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON TIMEOUT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

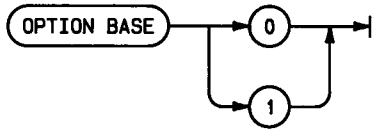
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram.

ON TIMEOUT is deactivated by OFF TIMEOUT. DISABLE does not affect ON TIMEOUT.

OPTION BASE

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement specifies the default lower bound of arrays.



Example Statements

```
OPTION BASE 0
OPTION BASE 1
```

Semantics

This statement can occur only once in each context. If used, OPTION BASE must precede any explicit variable declarations in a context. Since arrays are passed to subprograms by reference, they maintain their orginal lower bound, even if the new context has a different OPTION BASE. Any context that does not contain an OPTION BASE statement assumes default lower bounds of zero.

The OPTION BASE value is determined at prerun, and is used with all arrays declared without explicit lower bounds in COM, DIM, INTEGER, and REAL statements as well as with all implicitly dimensioned arrays. OPTION BASE is also used at runtime for any arrays declared without lower bounds in ALLOCATE.

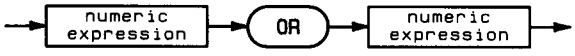
OPTIONAL

See the DEF FN and SUB statements.

OR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based on the logical inclusive-or of the arguments.



Example Statements

```
X=Y OR Z
IF File_type OR Device THEN Process
```

Semantics

An expression which evaluates to a non-zero value is treated as a logical 1. An expression must evaluate to zero to be treated as a logical 0.

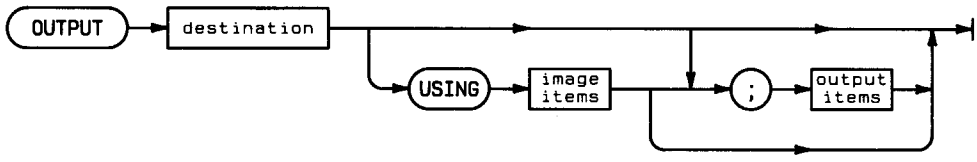
The truth table is:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

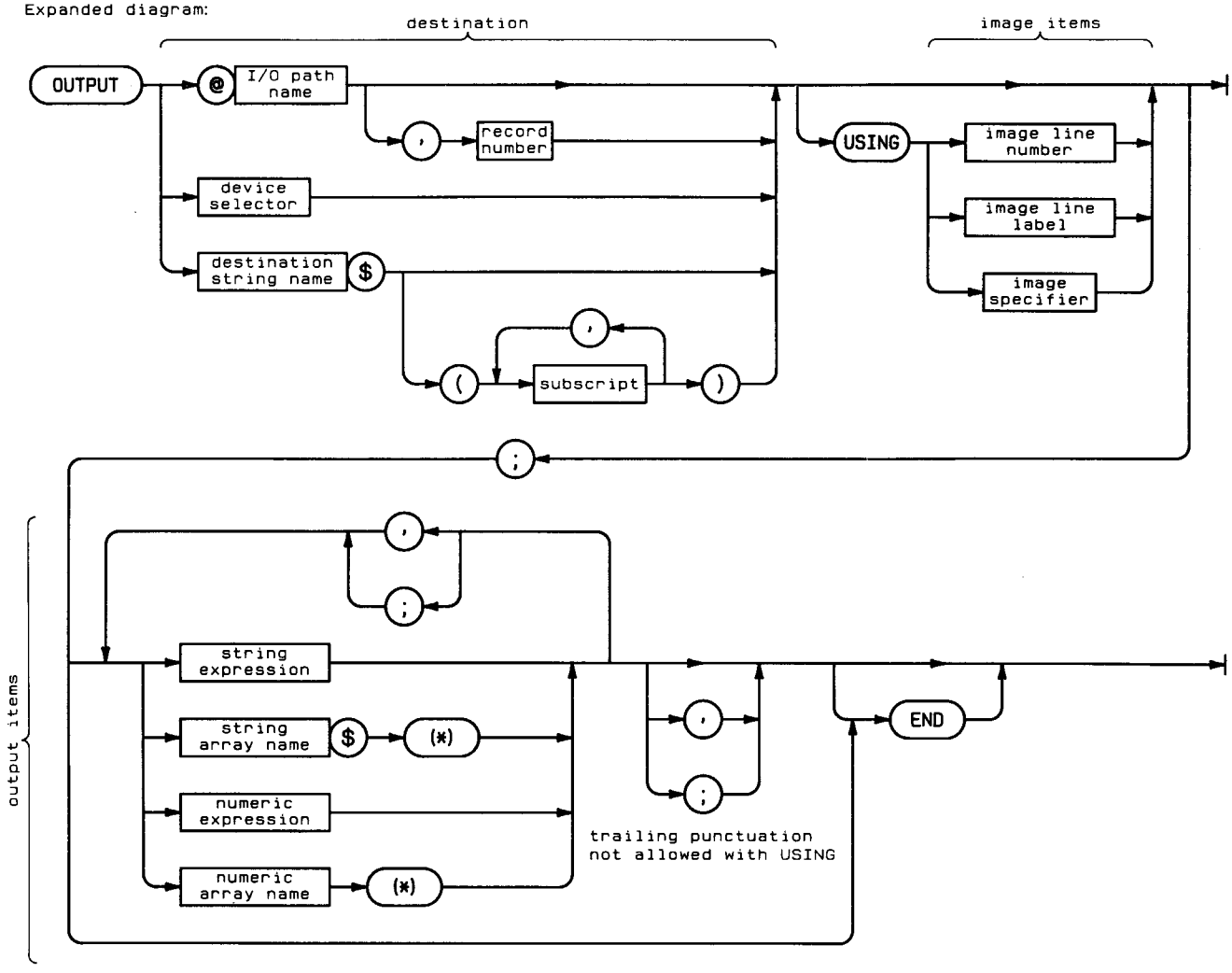
OUTPUT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

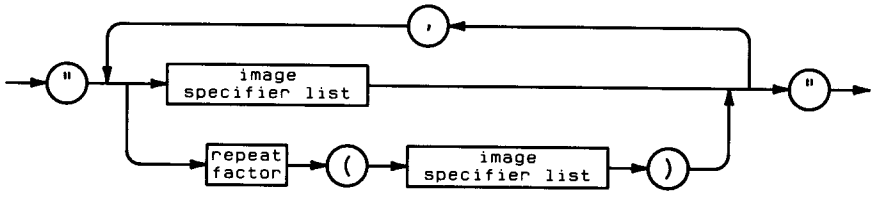
This statement outputs items to the specified destination. (If using OUTPUT with SRM, also refer to the "SRM" section of this manual.)



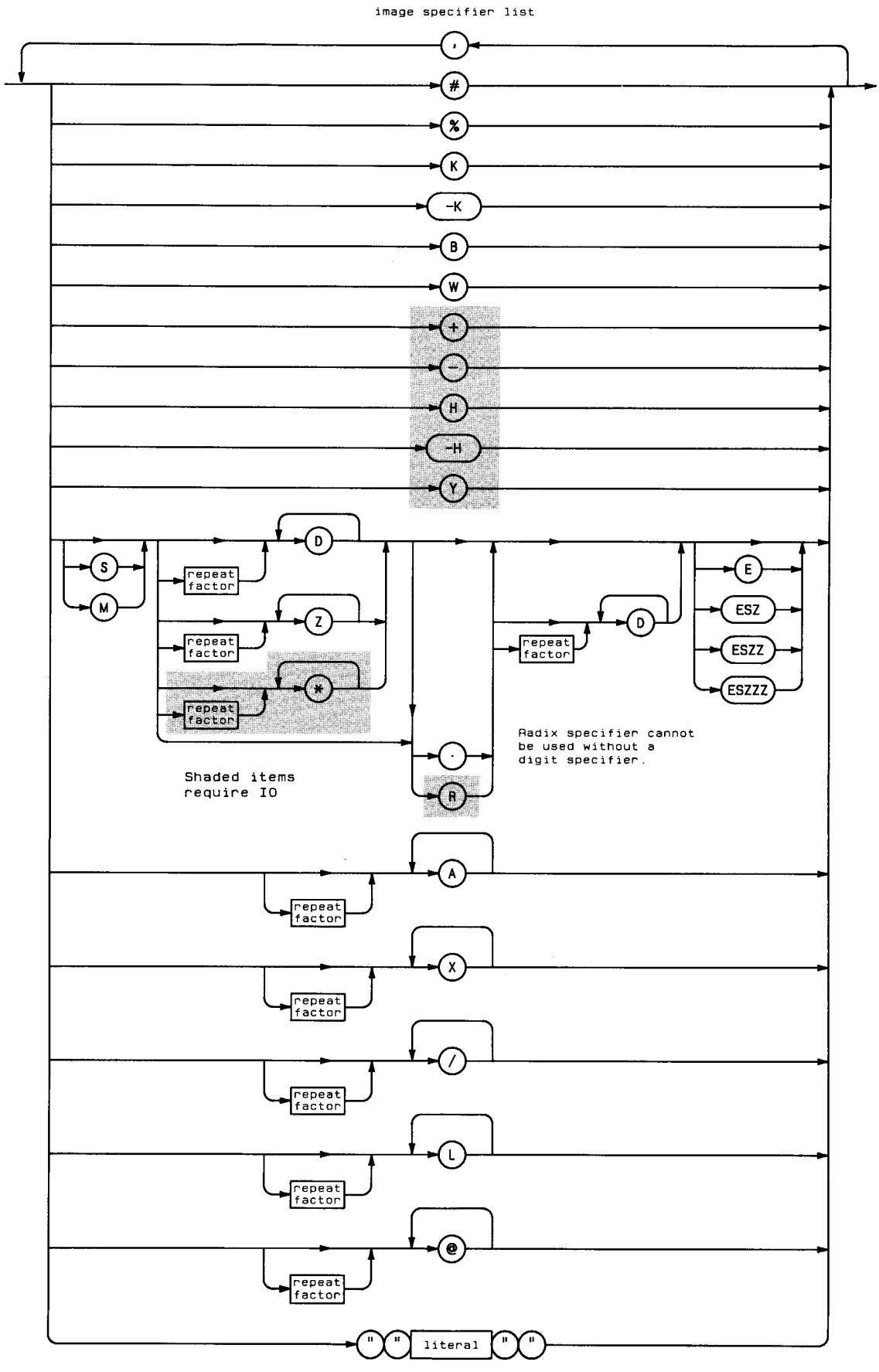
Expanded diagram:



literal form of file specifier:



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, mass storage file, or buffer	any valid name
record number	numeric expression, rounded to an integer	1 thru $2^{31}-1$
device selector	numeric expression, rounded to an integer	(see Glossary)
destination string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
image line number	integer constant identifying an IMAGE statement	1 thru 32 766
image line label	name identifying an IMAGE statement	any valid name
image specifier	string expression	(see drawing)
string array name	name of a string array	any valid name
numeric array name	name of a numeric array	any valid name
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed


image specifier list
,
#
%
K
-K
B
W
+
-
H
-H
Y
S
M
repeat factor
D
Z
*
E
ESZ
ESZZ
ESZZZ
Shaded items require IO
.
R
Radix specifier cannot be used without a digit specifier.
A
X
/
L
@
" " literal " "

Example Statements

```
OUTPUT 701;Number,String$;
OUTPUT @File;Array(*),END
OUTPUT @Rand,5 USING Fmt1;Item(5)
OUTPUT 12 USING "#,6A";B$[2;6]
OUTPUT @Printer;Rank;Id;Name$
```

Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E – 4 and less than 1E + 6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Arrays

Entire arrays may be output by using the asterisk specifier. Each element in an array is treated as an item by the OUTPUT statement, as if the items were listed separately, separated by the punctuation following the array specifier. If no punctuation follows the array specifier, a comma is assumed. The array is output in row major order (rightmost subscript varies fastest.)

Files as Destination

If an I/O path has been assigned to a file, the file may be written to with OUTPUT statements. The file must be an ASCII or BDAT file. The attributes specified in the ASSIGN statement are used if the file is a BDAT file.

Serial access is available for both ASCII and BDAT files. Random access is available for BDAT files. The end-of-file marker (EOF) and the file pointer are important to both serial and random access. The file pointer is set to the beginning of the file when the file is opened by an ASSIGN. The file pointer always points to the next byte to be written by serial OUTPUT operations. The EOF pointer is read from the media when the file is opened by an ASSIGN. On a newly-created file, EOF is set to the beginning of the file. After each OUTPUT operation, the EOF is updated **internally** to the maximum of the file pointer or the previous EOF value. The EOF pointer is updated on the **media** at the following times:

- When the current end-of-file changes.
- When END is specified in an OUTPUT statement directed to the file.
- When a CONTROL statement directed to the I/O path name changes the position of the EOF.

Random access uses the record number parameter to write items to a specific location in a file. The OUTPUT begins at the start of the specified record and must fit into one record. The record specified cannot be beyond the record containing the EOF, if EOF is at the first byte of a record. The record specified can be one record beyond the record containing the EOF, if EOF is not at the first byte of a record. Random access is always allowed to records preceding the EOF record. If you wish to write randomly to a newly created file, either use a CONTROL statement to position the EOF in the last record, or write some "dummy" data into every record.

When data is written to an ASCII file, each item is sent as an ASCII representation with a 2-byte length header. Data sent to a BDAT file is sent in internal format if FORMAT is OFF, and is sent as ASCII characters if FORMAT is ON. (See "Devices as Destination" for a description of these formats.)

Devices as Destination
An I/O path or a device selector may be used to direct OUTPUT to a device. If a device selector is used, the default system attributes are used (see ASSIGN). If an I/O path is used, the ASSIGN statement used to associate the I/O path with the device also determines the attributes used. If multiple listeners were specified in the ASSIGN, the OUTPUT is directed to all of them. If FORMAT ON is the current attribute, the items are sent in ASCII. Items followed by a semicolon are sent with nothing following them. Numeric items followed by a comma are sent with a comma following them. String items followed by a comma are sent with a CR/LF following them. If the last item in the OUTPUT statement has no punctuation following it, the current end-of-line (EOL) sequence is sent after it. Trailing punctuation eliminates the automatic EOL.

If FORMAT OFF is the current attribute, items are sent to the device in the computer's internal format. Punctuation following items has no effect on the OUTPUT. Two bytes are sent for each INTEGER, eight bytes for each REAL. Each string output consists of a four byte header containing the length of the string, followed by the actual string characters. If the number of characters is odd, an additional byte containing a blank is sent after the last character.

CRT as Destination
If the device selector is 1, the OUTPUT is directed to the CRT. OUTPUT 1 and PRINT differ in their treatment of separators and print fields. The OUTPUT format is described under "Devices as Destination". See the PRINT keyword for a discussion of that format. OUTPUT 1 USING and PRINT USING to the CRT produce similar actions.

Keyboard as Destination
Outputs to device selector 2 may be used to simulate keystrokes. ASCII characters can be sent directly (i.e. "hello"). Non-ASCII keys (such as EXECUTE) are simulated by a two-byte sequence. The first byte is CHR$(255), and the second byte can be found in the "Second Byte of Non-ASCII Key Sequences" table in the back of this book.

When simulating keystrokes, unwanted characters (such as the EOL sequence) can be avoided with an image specifier (such as "#,B" or "#,K"). See "OUTPUT with USING".

Strings as Destination
If a string is used for the destination, the string is treated similarly to a file. However, there is no file pointer; each OUTPUT begins at the beginning of the string, and writes serially within the string.

Buffers as Destination (Requires TRANS)
When the destination is an I/O path name assigned to a buffer, data is placed in the buffer beginning at the location indicated by the buffer's fill pointer. As data is sent, the current number-of-bytes register and fill pointer are adjusted accordingly. Encountering the empty pointer (buffer full) produces an error unless a continuous outbound TRANSFER is emptying the buffer. In this case, the OUTPUT will wait until there is more room in the buffer for data.

If an I/O path is currently being used in an inbound TRANSFER, and an OUTPUT statement uses it as a destination, execution of the OUTPUT is deferred until the completion of the TRANSFER. An OUTPUT can be concurrent with an outbound TRANSFER only if the destination is the I/O path assigned to the buffer.

An OUTPUT to a string variable that is also a buffer will not update the buffer's pointers and will probably corrupt the data in the buffer.

Using END with Devices
The secondary keyword END may be specified following the last item in an OUTPUT statement. The result, when USING is not specified, is to suppress the EOL (End-of-Line) sequence that would otherwise be output after the last byte of the last item. If a comma is used to separate the last item from the END keyword, the corresponding item terminator is output (CR/LF for string items or comma for numeric items).

With HP-IB interfaces, END specifies an EOI signal to be sent with the last data byte of the last item. However, if no data is sent from the last output item, EOI is not sent. With Data Communications interfaces, END specifies an end-of-data indication to be sent with the last byte of the last output item.

OUTPUT With USING
When the computer executes an OUTPUT USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the output items, the field specifier is acted upon without accessing the output list. When the field specifier requires characters, it accesses the next item in the output list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching output item. If the image specifiers are exhausted before the output items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifier, trailing blanks are used to fill out the field.

Effects of the image specifiers on the OUTPUT statement are shown in the following table:

Image Specifier	Meaning
K	Compact field. Outputs a number or string in standard form with no leading or trailing blanks.
−K	Same as K.
H	Similar to K, except the number is output using the European number format (comma radix). (Requires IO)
−H	Same as H. (Requires IO)
S	Outputs the number's sign (+ or −).
M	Outputs the number's sign if negative, a blank if positive.
D	Outputs one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is output, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are output.
*	Like D, except that asterisks are output instead of leading zeros. (Requires IO)
.	Outputs a decimal-point radix indicator.
R	Outputs a comma radix indicator (European radix). (Requires IO)
E	Outputs an E, a sign, and a two-digit exponent.
ESZ	Outputs an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Outputs an E, a sign, and a three-digit exponent.
A	Outputs a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Outputs a blank.
literal	Outputs the characters contained in the literal.
B	Outputs the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.

Image Specifier	Meaning
W	Outputs a 16-bit word as a two's-complement integer. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is sent; if it is less than −32 768, then −32 768 is sent. If either an I/O path name with the BYTE attribute or a device selector is used to access an 8-bit interface, two bytes will be output; the most-significant byte is sent first. If an I/O path name with the BYTE attribute is used to access a 16-bit interface, the BYTE attribute is overridden, and one word is output in a single operation. If an I/O path name with the WORD attribute is used to access a 16-bit interface, a null pad byte is output whenever necessary to achieve alignment on a word boundary. If the destination is a BDAT file, string variable, or buffer, the BYTE or WORD attribute is ignored and all data are sent as bytes; however, pad byte(s) will be output when necessary to achieve alignment on a word boundary. The pad character may be changed by using the CONVERT attribute; see the ASSIGN statement for further information.
Y	Like W, except that no pad bytes are output to achieve word alignment. If an I/O path with the BYTE attribute is used to access a 16-bit interface, the BYTE attribute is not overridden (as with the W specifier above). (Requires IO)
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence following the last output item.
%	Ignored in OUTPUT images.
+	Changes the automatic EOL sequence that normally follows the last output item to a single carriage-return. (Requires IO)
−	Changes the automatic EOL sequence that normally follows the last output item to a single line-feed. (Requires IO)
/	Outputs a carriage-return and a line-feed.
L	Outputs the current end-of-line (EOL) sequence. The default EOL characters are CR and LF; see ASSIGN for information on re-defining the EOL sequence. If the destination is an I/O path name with the WORD attribute, a pad byte may be sent after the EOL characters to achieve word alignment.
@	Outputs a form-feed.

END with OUTPUT...USING

Using the optional secondary keyword END in an OUTPUT...USING statement produces results which differ from those in an OUTPUT statement without USING. Instead of always suppressing the EOL sequence, the END keyword only suppresses the EOL sequence when no data is output from the last output item. Thus, the # image specifier generally controls the suppression of the otherwise automatic EOL sequence.

With HP-IB interfaces, END specifies an EOI signal to be sent with the last byte output. However, no EOI is sent if no data is sent from the last output item or the EOL sequence is suppressed. With Data Communications interfaces, END specifies an end-of-data indication to be sent at the same times an EOI would be sent on HP-IB interfaces.

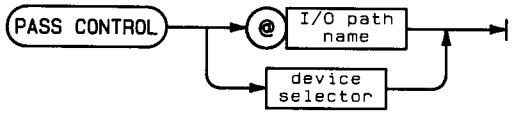
PARITY

See the ASSIGN statement.

PASS CONTROL

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to pass the capability of Active Controller to a specified HP-IB device.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an HP-IB device	any valid name
device selector	numeric expression, rounded to an integer	must contain primary address (see Glossary)

Example Statements

```
PASS CONTROL 719
PASS CONTROL @Controller_19
```

Semantics

Executing this statement first addresses the specified device to talk and then sends the Take Control message (TCT), after which Attention is placed in the False state. The computer then assumes the role of a bus device (a non-active controller).

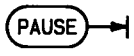
The computer must currently be the active controller to execute this statement, and primary addressing (but not multiple listeners) must be specified.

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN TAD $\overline{\text{TCT}}$ $\overline{\text{ATN}}$	Error	ATN TAD $\overline{\text{TCT}}$ $\overline{\text{ATN}}$
Not Active Controller	Error			

PAUSE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement suspends program execution. (Also see TRACE PAUSE.)



Semantics

PAUSE suspends program execution before the next line is executed, until the CONTINUE key is pressed or CONT is executed. If the program is modified while paused, RUN must be used to restart program execution.

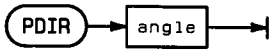
When program execution resumes, the computer attempts to service any ON INTR events that occurred while the program was paused. ON END, ON ERROR, or ON TIMEOUT events generate errors if they occur while the program is paused. ON KEY and ON KNOB events are ignored while the program is paused.

Pressing the PAUSE (or STOP on HP 46020A keyboard) key, or typing PAUSE and pressing EXECUTE ENTER or RETURN will suspend program execution at the end of the line currently being executed.

PDIR

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the angle with which IPLOT, RPLOT, POLYGON, POLYLINE, and RECTANGLE output are rotated.



Item	Description/Default	Range Restrictions
angle	numeric expression in current units of angle; Default = 0.	—

Example Statements

```
PDIR 20
PDIR ACS(Side)
```

Semantics

The rotation is about the local origin of the RPLOT, POLYGON, POLYLINE or RECTANGLE.

The angle is interpreted as counter-clockwise rotation from the X-axis.

PEN

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement selects a pen value to be used for all subsequent lines. (For information about PEN as a secondary keyword, see the AREA statement.)



Item	Description/Default	Range Restrictions
pen selector	numeric expression, rounded to an integer	−32 768 thru +32 767 (device dependent)

Example Statements

```
PEN 4
PEN Select
PEN Pen_number(I,J)
```

Semantics

For devices which support more than one line color (color CRT), or physical pen (external hard copy plotters), this statement specifies the line color or physical pen to be used for all subsequent lines until the execution of another PEN statement or until the execution of a PLOT, IPLOT, RPLOT, or SYMBOL statement with an array argument which changes the pen color (see Operation Selector 3 of these statements). The sign of the pen selectors affects the drawing mode.

In color map mode, specifying PEN 14 actually means "write a 14 into the frame buffer." The value of the frame buffer specifies the entry in the color map to be used, which in turn describes the actual color to be used.

The PEN statement can also be used to specify that the current drawing mode is to erase lines on all devices which support such an operation. This is specified with a negative pen number. An alternate mode of operation which allows non-dominant and complementing drawing may be accessed through the GESCAPE function.

When the PEN statement is executed, the pen used is mapped into the appropriate range, retaining the sign. For example, if you specify pen +8 on a device whose pens range from −7 through 7, it would actually use pen +1. The formulae used are as follows:

For monochromatic displays:

If pen selector>0 then use PEN 1 (draw lines)
If pen selector=0 then use PEN 0 (complement[1] lines)
If pen selector<0 then use PEN −1 (erase lines)

For the four-plane color displays *not* in COLOR MAP mode, and the HP 98627A:

If pen selector>0 then use PEN (pen selector − 1) MOD 7 + 1
If pen selector=0 then use PEN 0 (complement)
If pen selector<0 then use PEN −((ABS(pen selector) − 1) MOD 7 + 1)

For the four-plane color displays in COLOR MAP mode:

If pen selector>0 then use PEN (pen selector − 1) MOD MaxPen + 1
If pen selector=0 then use PEN 0
If pen selector<0 then use PEN − ((ABS(pen selector) − 1) MOD MaxPen + 1)

where MaxPen is the highest pen number (the lowest is 0). Four planes: MaxPen=15; eight planes: MaxPen=255.

For an HPGL plotter:

use PEN pen selector

On an HPGL plotter, no checking is done to determine if the requested pen actually exists. Pen zero puts away any pen if the plotter supports such an operation.

Non-Color Map Mode

The value written into the frame buffer depends not only on what pen is being used, but whether or not the computer is in color map mode. The colors for the default (non-color map) mode are given because the color map cannot be changed in this mode.

The meanings of the different pen values are shown in the tables below. The pen value can cause either a 1 (draw), a 0 (erase), no change, or invert the value in each location in the frame buffer.

Non-Color Map Mode

Pen	Color	Plane 1 (Red)	Plane 2 (Green)	Plane 3 (Blue)
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1

1 "Complement" means to change the state of pixels; that is, to draw lines where there are none, and to erase where lines already exist.

Drawing with the pen numbers indicated in the above table results in the memory planes being set to the indicated values. Drawing with the negatives of the pen numbers while in *normal pen mode* causes the bits to be cleared where there are 1s in the table. Drawing with the negatives of the pen numbers while in *alternate pen mode* causes the bits to be inverted where there are 1s in the table. In either case, no change will take place where there are 0s in the table. Although complementing lines can be drawn, complementing area fills cannot be executed.

Positive pen numbers in alternate drawing mode allows non-dominant drawing. (Non-dominant drawing causes the values in the frame buffer to be inclusively OR ed with the value of the pen.) Pen 0 in normal mode complements. Pen 0 in alternate mode draws in the background color. Since the table represents the computer in non-color map mode, the fourth memory plane is always cleared.

Color Map Mode

When operating the color display in color map mode, pen colors can be redefined at will. For this reason, no colors are mentioned in the following table. Unlike non-color-map mode, the fourth bit in the frame buffer is used when in color map mode. Also, memory planes 1, 2, and 3 are not associated with red, green, and blue.

Drawing with a pen merely puts the pen number into that pixel's location. The computer looks into the corresponding entry in the color map to determine what the actual color the pixel is to exhibit.

Color Map Mode

Pen	Action	Plane 1	Plane 2	Plane 3	Plane 4
0	Background	0	0	0	0
1	Draw Pen 1	1	0	0	0
2	Draw Pen 2	0	1	0	0
3	Draw Pen 3	1	1	0	0
4	Draw Pen 4	0	0	1	0
5	Draw Pen 5	1	0	1	0
6	Draw Pen 6	0	1	1	0
7	Draw Pen 7	1	1	1	0
8	Draw Pen 8	0	0	0	1
9	Draw Pen 9	1	0	0	1
10	Draw Pen 10	0	1	0	1
11	Draw Pen 11	1	1	0	1
12	Draw Pen 12	0	0	1	1
13	Draw Pen 13	1	0	1	1
14	Draw Pen 14	0	1	1	1
15	Draw Pen 15	1	1	1	1

Drawing with the negatives of the pen numbers while in *normal pen mode* causes the bits to be cleared where there are 1s in the table. Drawing with the negatives of the pen numbers while in *alternate pen mode* causes the bits to be inverted where there are 1s in the table. In either case, no change will take place where there are 0s in the table.

Pen 0 merely draws in the background color. Although complementing lines can be drawn, complementing area fills cannot be executed.

Default Colors

The RGB and HSL values for the default pen colors while in color map mode are shown below. These can be changed by the SET PEN statement. First, the RGB (red/green/blue) values:

Color Map Default Color Definitions (RGB)

Pen	Color	Red	Green	Blue
0	Black	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1
8	Black	0	0	0
9	Olive Green	.80	.73	.20
10	Aqua	.20	.67	.47
11	Royal Blue	.53	.40	.67
12	Maroon	.80	.27	.40
13	Brick Red	1.00	.40	.20
14	Orange	1.00	.47	0.00
15	Brown	.87	.53	.27

The same default color map colors are represented below in their HSL (hue/saturation/luminosity) representations:

Color Map Default Color Definitions (HSL)

Pen	Color	Hue	Sat.	Lum.
0	Black	0	0	0
1	White	0	0	1
2	Red	0	1	1
3	Yellow	.17	1	1
4	Green	.33	1	1
5	Cyan	.50	1	1
6	Blue	.67	1	1
7	Magenta	.83	1	1
8	Black	0	0	0
9	Olive Green	.15	.75	.80
10	Aqua	.44	.75	.68
11	Royal Blue	.75	.36	.64
12	Maroon	.95	.65	.78
13	Brick Red	.04	.80	1.00
14	Orange	.08	1.00	1.00
15	Brown	.08	.70	.85

PENUP

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lifts the pen on the current plotting device.



PI

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns 3.141 592 653 589 79, which is an approximate value for π.



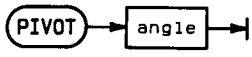
Example Statements

```
Area=PI*Radius^2
PRINT X,X*2*PI
```

PIVOT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies a rotation of coordinates which is applied to all subsequently drawn lines.



Item	Description/Default	Range Restrictions
angle	numeric expression in current units of angle	(same as COS)

Example Statements

```
PIVOT 30
IF Special THEN PIVOT Radians
```

Semantics

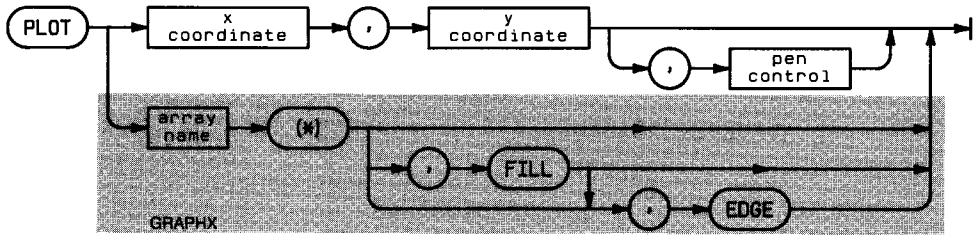
The specified angle is interpreted according to the current angle mode (RAD or DEG).

The specified angular rotation is performed about the logical pen's position at the time the PIVOT is executed. This rotation is applied only to lines drawn subsequent to the PIVOT; logical pen movement is **not** affected by PIVOT. Consequently, PIVOT generally causes the logical and physical pens to be left at different positions. Other operations which cause similar effects are attempts to draw outside clip limits and direct HPGL output to plotters.

PLOT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement moves the pen from the current pen position to the specified X and Y coordinates. It can be used to move without drawing, or to draw a line, depending on the pen control value.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression, in current units	—
y coordinate	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move).	−32 768 thru +32 767
array name	name of two-dimensional, two-column or three-column numeric array. (Requires GRAPHX)	any valid name

Example Statements

```
PLOT X,Y,-1
PLOT -5,12
PLOT Shape(*),FILL,EDGE
```

Semantics

Non-Array Parameters

The specified X and Y position information is interpreted according to the current unit-of-measure. Lines are drawn using the current pen color and line type.

PLOT is affected by the PIVOT transformation.

The line is clipped at the current clipping boundary. If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

The optional pen control parameter specifies the following plotting actions; the default value is +1 (down after move).

Pen Control Parameter

Pen Control	Resultant Action
−Even	Pen up before move
−Odd	Pen down before move
+Even	Pen up after move
+Odd	Pen down after move

The above table is summed up by: even is up, odd is down, positive is after pen motion, negative is before pen motion.

Array Parameters

When using the PLOT statement with an array, either a two-column or a three-column array may be used. If a two-column array is used, the third parameter is assumed to be +1: pen down after move.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the ranges 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the PLOT statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the PLOT statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the PLOT statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

When using a PLOT statement with an array, the following table of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth. See the list below.

Column 1	Column 2	Operation Selector	Meaning
X	Y	−2	Pen up before moving
X	Y	−1	Pen down before moving
X	Y	0	Pen up after moving (Same as +2)
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	ignored	3	Select pen
line type	repeat value	4	Select line type
color	ignored	5	Color value
ignored	ignored	6	Start polygon mode with FILL
ignored	ignored	7	End polygon mode
ignored	ignored	8	End of data for array
ignored	ignored	9	NOP (no operation)
ignored	ignored	10	Start polygon mode with EDGE
ignored	ignored	11	Start polygon mode with FILL and EDGE
ignored	ignored	12	Draw a FRAME
pen number	ignored	13	Area pen value
red value	green value	14	} Color
blue value	ignored	15	} Value
ignored	ignored	>15	Ignored

Moving and Drawing

If the operation selector is less than or equal to two, it is interpreted in exactly the same manner as the third parameter in a non-array PLOT statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens

An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types

An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color

Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation Selector 14 is used in conjunction with Operation Selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessarily immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation Selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word, that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities (which range from 0 thru 1) for red, green, and blue in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME

Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the PLOT statement, so one probably would not have more than one operation selector 12 in an array to PLOT, since the last FRAME will overwrite all the previous ones.

Premature Termination

Operation selector 8 causes the PLOT statement to be terminated. The PLOT statement will successfully terminate if the actual end of the array has been reached, so use of operation selector 8 is optional.

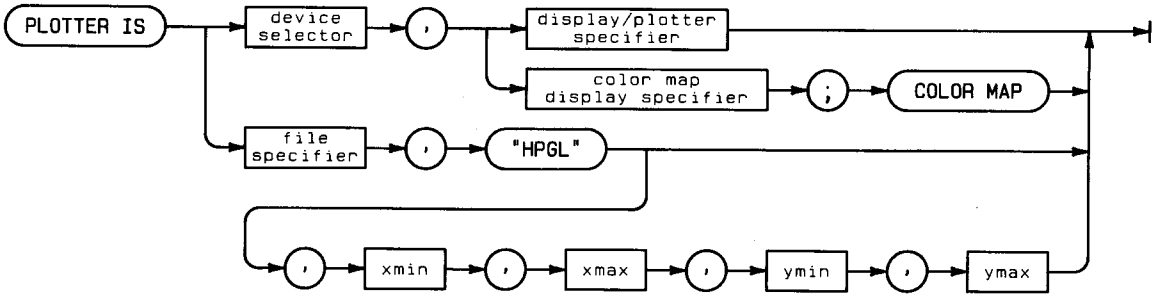
Ignoring Selected Rows in the Array

Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

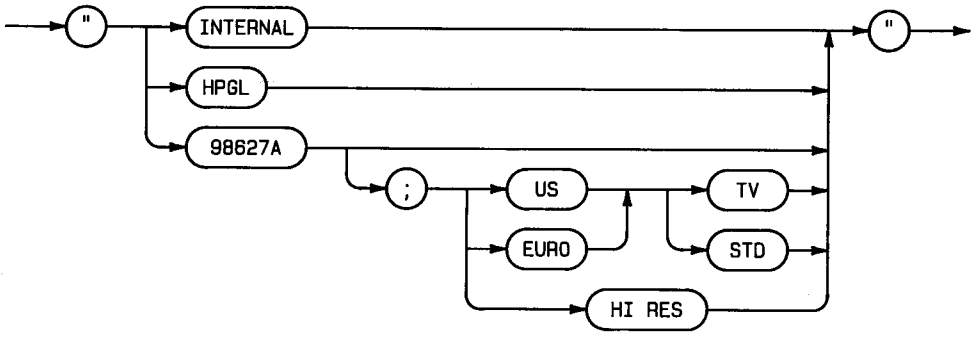
PLOTTER IS

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

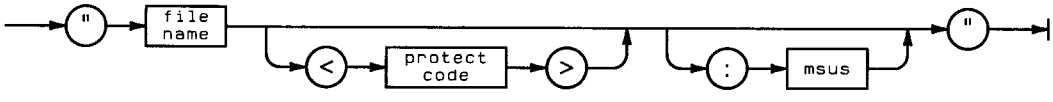
This statement selects a plotting device. (If using PLOTTER IS with SRM, also refer to the "SRM" section of this manual.)



literal form of display/plotter specifier:



literal form of file specifier:



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
display/plotter specifier	string expression	(see drawing)
color map display specifier	string expression	INTERNAL
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed

Item	Description/Default	Range Restrictions
msus	literal	(see MASS STORAGE IS)
xmin	numeric expression; Default = −392.75mm	device dependent
xmax	numeric expression; Default = 392.75mm	device dependent
ymin	numeric expression; Default = −251.5mm	device dependent
ymax	numeric expression; Default = 251.5mm	device dependent

Example Statements

```
PLOTTER IS 3,I$
PLOTTER IS CRT,"INTERNAL";COLOR MAP
PLOTTER IS Dsg,"HPGL"
PLOTTER IS "Newfile","HPGL"
PLOTTER IS "Plotfile:REMOTE","HPGL",6.25,256.25,6.975,186.975
```

Semantics

Files

The file must be a BDAT file. This statement causes all subsequent plotter output to go to the specified file.

Xmin,xmax,ymin,ymax are the hard clip limits of the plotter in millimetres.

This assumes .025mm per plotter unit. The default size is for an HP 7580 or HP 7585 D-size drawing. See the plotter manual for more information on plotter limits.

The PLOTTER IS statement positions the file pointer to the beginning of the file.

The file is closed when another PLOTTER IS statement is executed or SCRATCH A, GINIT or Reset is executed.

An end of file error occurs when the end of file is reached.

Plotters

The hard clip limits of the plotter are read in when this statement is executed. Therefore, the specified device must be capable of responding to this interrogation.

Displays

The statement `PLOTTER IS CRT, "INTERNAL"` is executed whenever a graphics statement is executed which needs a plotter (see GINIT) and no plotter is active. The plotter activated is the first device encountered in the following order:

1. The alpha display, if it has graphics capabilities,
2. Internal 98542A, 98543A, 98544A, 98545A, or 98700 at select code 6,
3. Non-bit-mapped alpha display with graphics capabilities at select code 3,
4. External 98700 at select code >7,
5. 98627A at select code >7.

If the COLOR MAP option is not included and the plotting device is the Model 236 color display, the 4th memory plane is cleared.

If the COLOR MAP option is specified and the plotting device has a color map, the capability of changing the color map will be enabled (see SET PEN). Also, the values written into the frame buffer are different than they would be if color map mode was not enabled.

HP 98627A Emulation

To emulate the HP 98627A non-color-mapped device on a color display, execute a PLOTTER IS statement *without* the COLOR MAP keyword. This causes the color map to be defined as follows, where 0 is zero intensity and 1 is full intensity.

HP 98627A Non-Color Map Emulation

Pen	Color	Red	Green	Blue
0	Black	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1
8	Black	0	0	0
9-15	White	1	1	1

The complementing cursor will be white on top of all colors except white, in which case it will be black. In this detail, the cursor implementation is not an emulation of the 98627A.

COLOR MAP

In the COLOR MAP mode, the color map is initialized so that the first eight colors are the same as they were in the default mode, and the second eight colors simulate HP's designer colors of plotter pen ink.

Although the pen numbers select the same color in color map mode as in non-color map mode (for the first eight pens), the actual values written to the frame buffer are different. This results from the different interpretation of the values in the frame buffer: in non-color map mode, the values are RGB values; in color-map mode, the values are indices into the color map. This means that a picture drawn in non-color map mode will change colors if a PLOTTER IS with the COLOR MAP option is executed. The reverse is also true.

When the `PLOTTER IS` statement is executed, the color map is initialized to a default state. If the graphics write-enable mask is left in the default mode, the entire color map will be initialized as before. Otherwise, the following algorithm is used: all color map entries whose binary representation has 1s only in graphics planes are initialized; color map entries whose binary representation has 1s in non-graphics planes will remain unchanged. This is done to insure that only pens dedicated to graphics are initialized. For example, with a graphics write mask of 7 (binary 0000 0111), only pens 0 through 7 are initialized. Higher numbered pens would remain unchanged since their binary representation would have 1s in non-graphics planes.

Display Specifiers

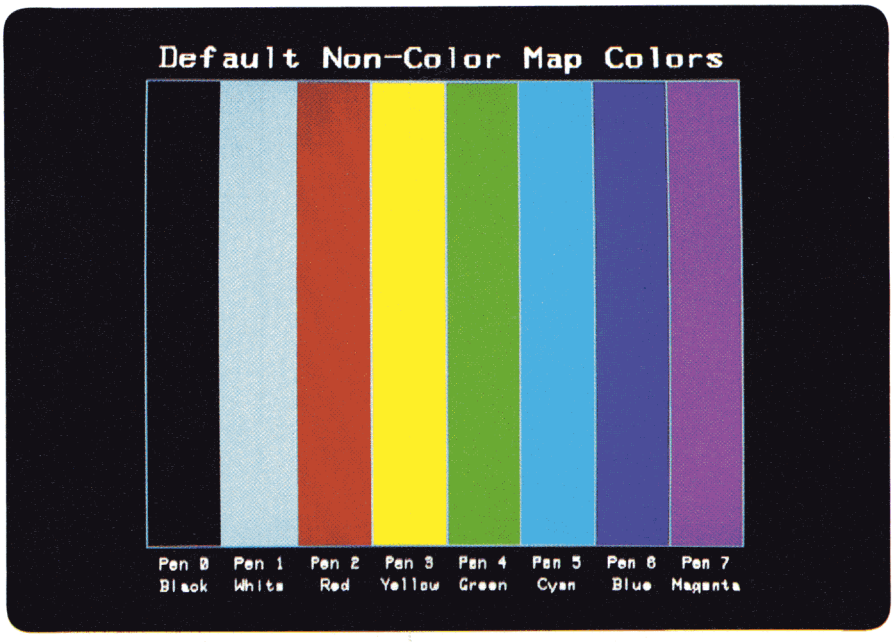
There are several values which can be used when specifying the display on which graphics operations are done:

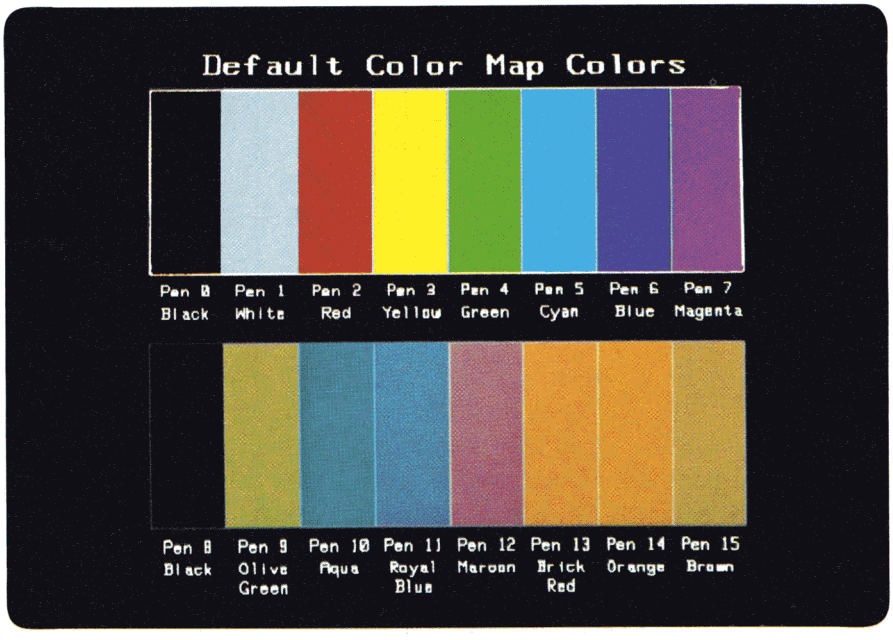
`PLOTTER IS CRT,"INTERNAL"` or `PLOTTER IS 1,"INTERNAL"`	This is the safest of the possibilities. "`CRT`" is a built-in function which returns the value 1, and the value 1 is interpreted by the graphics system as "the default display." The default display may be an external display if no internal display exists.
`PLOTTER IS 3,"INTERNAL"`	This specifies a non-bit-mapped display if there is one; otherwise, the action is equivalent to "`PLOTTER IS 1,"INTERNAL"`". Specifying a value of 3 makes sense for all Series 200 displays except the Model 237.
`PLOTTER IS 6,"INTERNAL"`	Always specifies a bit-mapped display. If one is not found, an error results.
`PLOTTER IS` ⟨*device selector*⟩`,"98627A"`[1]	This specifies a color graphics display connected through the 98627A interface card. This may have any one of several options specifying television format, etc.
`PLOTTER IS` ⟨*device selector*⟩`,"INTERNAL"`	With the 98700 display, it is possible to configure the display card so that it is at an external select code. For example, if you set the select code to 25, you would say: `PLOTTER IS 25,"INTERNAL"`

1 `PLOTTER IS` ⟨*device selector*⟩`,"INTERNAL"` is also accepted, and acts the same as `"98627A"`.

Default Pen Colors

The PLOTTER IS statement defines the color map to default values. Thes value are different depending on whether or not the COLOR MAP option was selected. Below are two color plates showing the eight default colors available with non-color map mode, and the sixteen default colors in color map mode.





The values, both in RGB and HSL, of the sixteen default pen colors are given below:

Color Map Default Color Definitions (RGB)

Pen	Color	Red	Green	Blue
0	Black	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1
8	Black	0	0	0
9	Olive Green	.80	.73	.20
10	Aqua	.20	.67	.47
11	Royal Blue	.53	.40	.67
12	Maroon	.80	.27	.40
13	Brick Red	1.00	.40	.20
14	Orange	1.00	.47	0.00
15	Brown	.87	.53	.27

The same default color map colors are represented below in their HSL (hue/saturation/luminosity) representations:

Color Map Default Color Definitions (HSL)

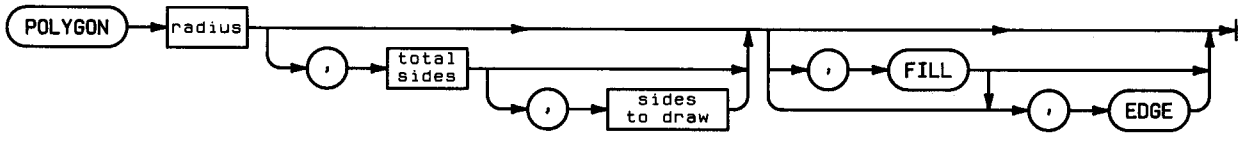
Pen	Color	Hue	Sat.	Lum.
0	Black	0	0	0
1	White	0	0	1
2	Red	0	1	1
3	Yellow	.17	1	1
4	Green	.33	1	1
5	Cyan	.50	1	1
6	Blue	.67	1	1
7	Magenta	.83	1	1
8	Black	0	0	0
9	Olive Green	.15	.75	.80
10	Aqua	.44	.75	.68
11	Royal Blue	.75	.36	.64
12	Maroon	.95	.65	.78
13	Brick Red	.04	.80	1.00
14	Orange	.08	1.00	1.00
15	Brown	.08	.70	.85

Eight-plane machines have 256-entry color maps. In these machines, pens 16 through 255 are defined to a variety of shades. For exact values, interrogate the color map with GESCAPE.

POLYGON

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws all or part of a closed regular polygon. The polygon can be filled and/or edged.



Item	Description/Default	Range Restrictions
radius	numeric expression, in current units	—
total sides	numeric expression, rounded to an integer. Default = 60	3 thru 32 767
sides to draw	numeric expression, rounded to an integer. Default = all sides.	1 thru 32 767

Example Statements

```
POLYGON 1.5,5,4,FILL,EDGE
POLYGON 4
```

Semantics

The radius is the distance that the vertices of the polygon will be from the logical pen position. The first vertex will be at a distance specified by "radius" in the direction of the positive X-axis. Specifying a negative radius results in the figure being rotated 180°. POLYGON is affected by the PIVOT and the PDIR transformations.

The total sides and the number of sides drawn need not be the same. Thus

```
POLYGON 1.5,8,5
```

will start to draw an octagon whose vertices are 1.5 units from the current pen position, but will only draw five sides of it before closing the polygon to the first point. If the number of sides to draw is greater than the specified total sides, sides to draw is treated as if it were equal to total sides.

POLYGON forces polygon closure, that is, the first vertex is connected to the last vertex, so there is always an inside and an outside area. This is true even for the degenerate case of drawing only one side of a polygon, in which case a single line results. This is actually two lines, from the first point to the last point, and back to the first point.

Polygon Shape

The shape of the polygon is affected by the viewing transformation specified by SHOW or WINDOW. Therefore, anisotropic scaling causes the polygon to be distorted; stretched or compressed along the axes. If a rotation transformation is in effect, the polygon will be rotated first, then stretched or compressed along the unrotated axes.

The pen status also affects the final shape of a polygon if sides to draw is less than total sides. If the pen is up at the time POLYGON is specified, the first vertex specified is connected to the last vertex specified, *not* including the center of the polygon, which is the current pen position. If the pen is down, however, the center of the polygon is also included in it. If sides to draw is less than total sides, piece-of-pie shaped polygon segments are created.

FILL and EDGE

FILL causes the interior of the polygon or polygon segment to be filled with the current fill color as defined by AREA PEN, AREA COLOR, or AREA INTENSITY. EDGE causes the edges of the polygon to be drawn using the current pen and line type. If both FILL and EDGE are specified, the interior will be filled, then the edge will be drawn. If neither FILL nor EDGE is specified, EDGE is assumed.

Polygons sent to an HPGL plotter are edged but not filled regardless of any FILL or EDGE directives on the statement.

After POLYGON has executed, the pen is in the same position it was before the statement was executed, and the pen is up. The polygon is clipped at the current clip limits.

Applicable Graphics Transformations

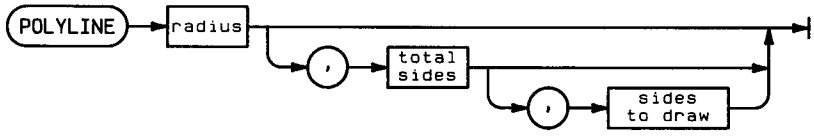
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

POLYLINE

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws all or part of an open regular polygon.



Item	Description/Default	Range Restrictions
radius	numeric expression, in current units.	—
total sides	numeric expression, rounded to an integer. Default = 60	3 thru 32 767
sides to draw	numeric expression, rounded to an integer. Default = all sides	1 thru 32 767

Example Statements

```
POLYLINE Radius,Sides,Sides_to_draw
POLYLINE 12,5
```

Semantics

The radius is the distance that the vertices of the polygon will be from the current pen position. The first vertex will be at a distance specified by "radius" in the direction of the positive X-axis. Specifying a negative radius results in the figure being rotated 180°. POLYLINE is affected by the PIVOT and PDIR transformation.

The total sides specified need not be the same as the sides to draw. Thus

```
POLYLINE 1.5,8,5
```

will start to draw an octagon whose vertices are 1.5 units from the current pen position, but will only draw five sides of it. If the number of sides to draw is greater than the total sides specified, it is treated as if it were equal to the total sides.

Shape of Perimeter

POLYLINE does not force polygon closure, that is, if sides to draw is less than total sides, the first vertex is not connected to the last vertex, so there is no "inside" or "outside" area.

The shape of the polygon is affected by the viewing transformation specified by SHOW or WINDOW. Therefore, anistropic scaling causes the perimeter to be distorted; stretched or compressed along the axes. If a rotation transformation is in effect, the polygon will be rotated first, then stretched or compressed along the unrotated axes.

The pen status affects the way a POLYLINE statement works. If the pen is up at the time POLYLINE is specified, the first vertex is on the perimeter. If the pen is down, the first point is the current pen position, which is connected to the first point on the perimeter.

After POLYLINE has executed, the current pen position is in the same position it was before the statement was executed, and the pen is up. The polygon is clipped at the current clip limits.

Applicable Graphics Transformations

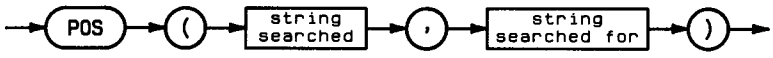
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

POS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the first position of a substring within a string.



Item	Description/Default	Range Restrictions
string searched	string expression	—
string searched for	string expression	—

Example Statements

```
Point=POS(Big$,Little$)
IF POS(A$,CHR$(10)) THEN Line_end
```

Semantics

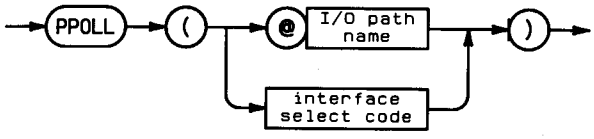
If the value returned is greater than 0, it represents the position of the first character of the string being searched for in the string being searched. If the value returned is 0, the string being searched for does not exist in the string being searched (or the string searched for is the null string).

Note that the position returned is the relative position within the string expression used as the first argument. Thus, when a substring is searched, the position value refers to that substring, not to the parent string from which the substring was taken.

PPOLL

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value representing eight status-bit messages of devices on the HP-IB.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
Stat=PPOLL(7)
IF BIT(PPOLL(@Hpib),3) THEN Respond
```

Semantics

The computer must be the active controller to execute this function.

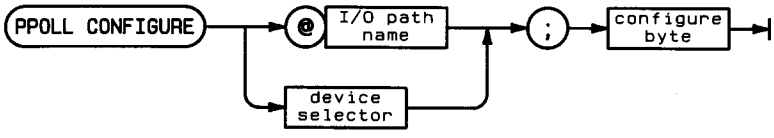
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration≥25μs) Read byte $\overline{EOI}$ Restore ATN to previous state	Error	ATN & EOI (duration≥25μs) Read byte $\overline{EOI}$ Restore ATN to previous state	Error
Not Active Controller	Error			

PPOLL CONFIGURE

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement programs the logical sense and data bus line on which a specified device responds to a parallel poll.



Item	Description/Default	Range Restrictions	Recommended Range
I/O path name	name assigned to a device or devices	any valid name	—
device selector	numeric expression, rounded to an integer	must contain a primary address (see Glossary)	—
configure byte	numeric expression, rounded to an integer	−32 768 thru +32 767	0 thru 15

Example Statements

```
PPOLL CONFIGURE 711;2
PPOLL CONFIGURE @Dvm;Response
```

Semantics

This statement assumes that the device's response is bus-programmable. The computer must be the active controller to execute this statement.

The configure byte is coded. The three least significant bits determine the data bus line for the response. The fourth bit determines the logical sense of the response.

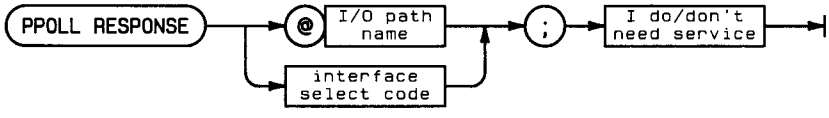
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN MTA UNL LAG PPC PPE	Error	ATN MTA UNL LAG PPC PPE
Not Active Controller	Error			

PPOLL RESPONSE

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines a response to be sent when an Active Controller performs a Parallel Poll on an HP-IB Interface. The response indicates whether this computer does or does not need service.



Item	Description/Default	Range Restrictions	Recommended Range
I/O path name	name assigned to an interface select code	any valid name	—
interface select code	numeric expression, rounded to an integer	7 thru 31	—
I do/don't need service	numeric expression, rounded to an integer	0 thru 32 767	0 or 1

Examples

```
PPOLL RESPONSE @Hp_ib;I_need_service
PPOLL RESPONSE Interface;0
```

Semantics

This statement defines the computer's response to a Parallel Poll (ATN & EOI) performed by the current Active Controller on the specified HP-IB Interface. This statement only sets up a potential response; no actual response is generated when the statement is executed.

If the value of the "I do/don't need service" parameter is 0, the computer is directed to place a logical false on the bit on which it has been defined to respond; this response will tell the Active Controller that this (non-active) controller does not need service. Any non-zero, positive value of this parameter (within the stated range) directs the computer to set up a true response, which will tell a polling Active Controller that the computer requires service.

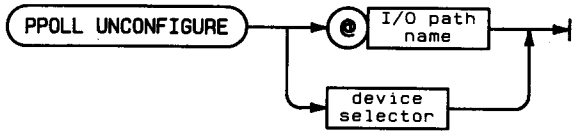
The bit on which the computer is to place its Parallel Poll response is determined by the value of the last "configure byte" written to CONTROL Register 5 of the corresponsing HP-IB Interface. In general, this configure byte can be read from HP-IB STATUS Register 7 by the service routine that responds to Parallel-Poll-Configuration-Change interrupts (Bit 14 of the Interrupt Enable Register). This configure byte may then be written into HP-IB CONTROL Register 5, and the response desired by the Active Controller will be sent when a Parallel Poll is conducted.

This statement may be executed by either an Active Controller or a non-active controller.

PPOLL UNCONFIGURE

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables the parallel poll response of a specified device or devices.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
PPOLL UNCONFIGURE 7
PPOLL UNCONFIGURE @Plotter
```

Semantics

The computer must be the active controller to execute PPOLL UNCONFIGURE.

If multiple devices are specified by an I/O path name, all specified devices are deactivated from parallel poll response. If the device selector or I/O path name refers only to an interface select code, all devices on that interface are deactivated from parallel poll response.

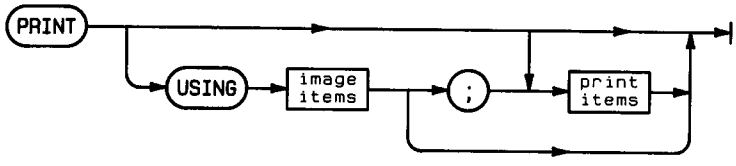
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			

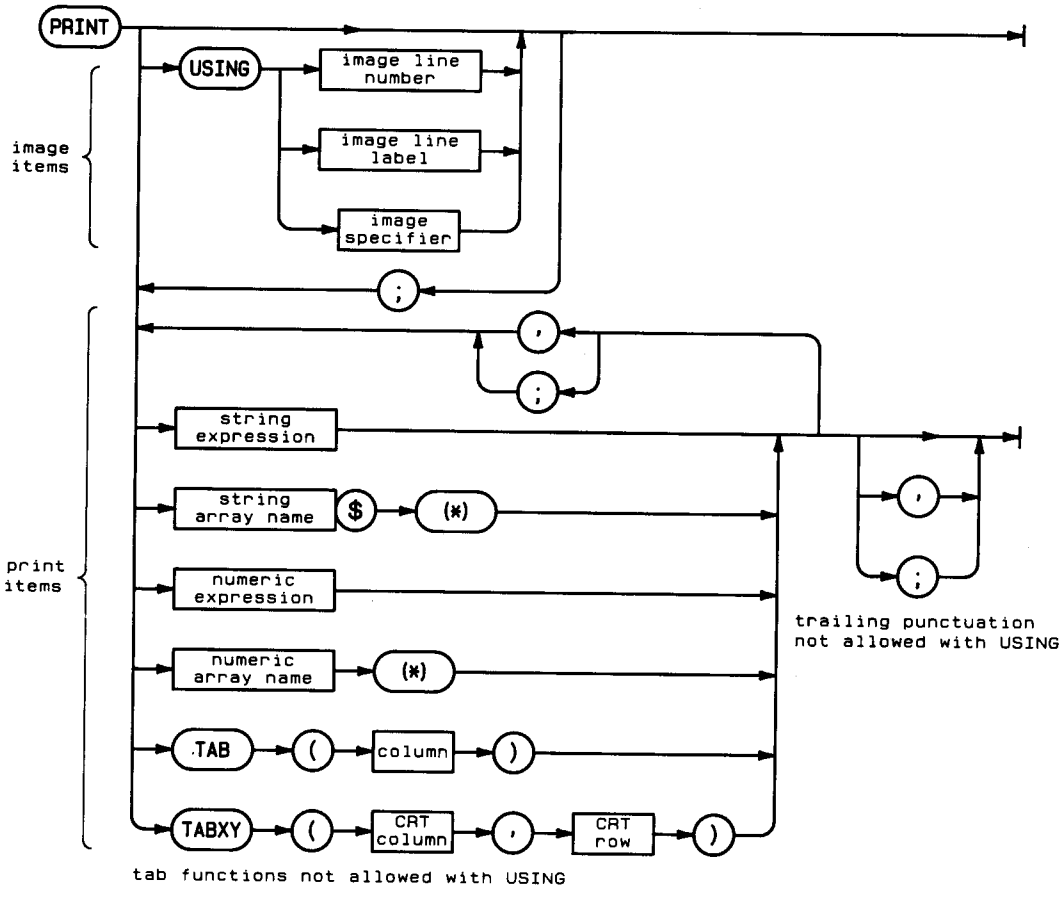
PRINT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

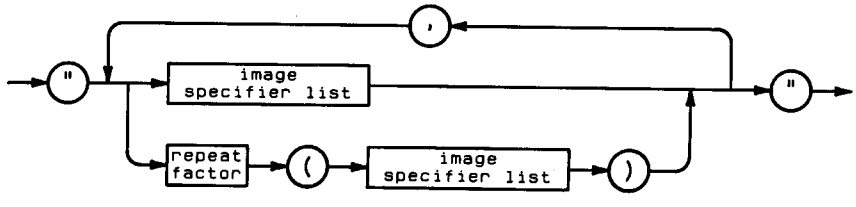
This statement sends items to the PRINTER IS device.

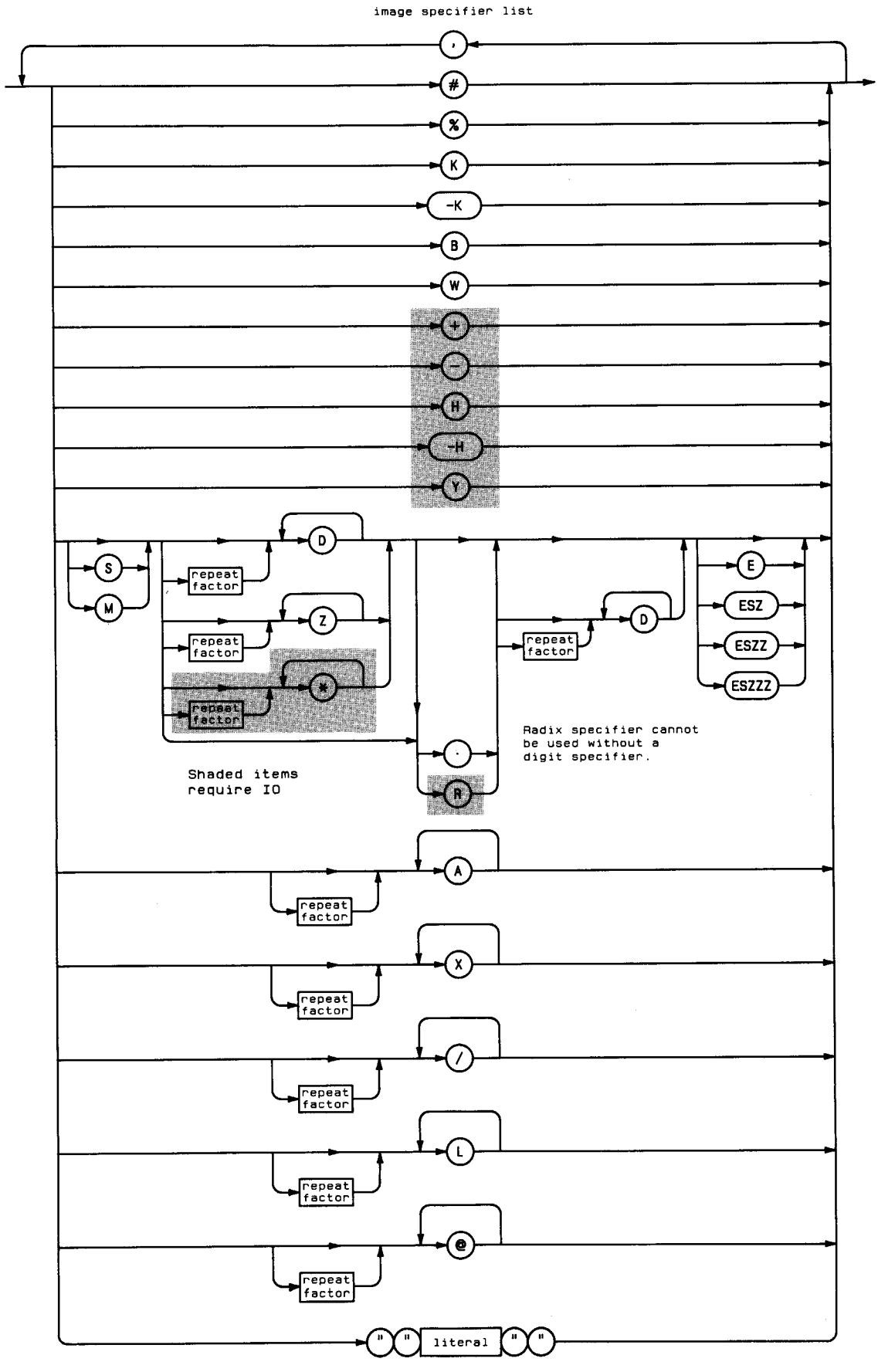


Expanded diagram:



literal form of image specifier:




image specifier list
,
#
%
K
-K
B
W
+
-
H
-H
Y
D
S
M
repeat factor
Z
*
E
ESZ
ESZZ
ESZZZ
Shaded items require IO
.
R
Radix specifier cannot be used without a digit specifier.
A
X
/
L
@
" " literal " "

Item	Description/Default	Range Restrictions	Recommended Range
image line number	integer constant identifying an IMAGE statement	1 thru 32 766	—
image line label	name identifying an IMAGE statement	any valid name	—
image specifier	string expression	(see drawing)	—
string array name	name of a string array	any valid name	—
numeric array name	name of a numeric array	any valid name	—
column	numeric expression, rounded to an integer	−32 768 thru +32 767	device dependent
CRT column	numeric expression, rounded to an integer	0 thru 32 767	1 thru screen width
CRT row	numeric expression, rounded to an integer	0 thru 32 767	1 thru 18
image specifier list	literal	(see next drawing)	—
repeat factor	integer constant	1 thru 32 767	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed	—

Example Statements

```
PRINT "LINE";Number
PRINT Array(*);
PRINT TABXY(1,1),Header$,TABXY(Col,3),Message$
PRINT USING "5Z.DD";Money
PRINT USING Fmt3;Id,Item$,Kilograms/2.2
```

Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E − 4 and less than 1E + 6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the print list is exhausted, an End-Of-Line (EOL) sequence is sent to the PRINTER IS device, unless it is suppressed by trailing punctuation or a pound-sign (#) image specifier. The printer width for EOL sequences generation is set to the screen width (50, 80 or 128 characters) for CRTs and to 80 for external devices unless the WIDTH attribute of the PRINTER IS statement was specified. WIDTH is off for files. This "printer width exceeded" EOL is not suppressed by trailing punctuation, but can be suppressed by the use of an image specifier.

Control Codes

Some ASCII control codes have a special effect in PRINT statements if the PRINTER IS device is the CRT (device selector = 1):

Character	Keystroke	Name	Action
CHR$(7)	CTRL-G	bell	Sounds the beeper
CHR$(8)	CTRL-H	backspace	Moves the print position back one character.
CHR$(10)	CTRL-J	line-feed	Moves the print position down one line.
CHR$(12)	CTRL-L	form-feed	Prints two line-feeds, then advances the CRT buffer enough lines to place the next item at the top of the CRT.
CHR$(13)	CTRL-M	carriage-return	Moves the print position to column 1.

The effect of ASCII control codes on a printer is device dependent. See your printer manual to find which control codes are recognized by your printer and their effects.

CRT Enhancements

There are several character enhancements (such as inverse video and underlining) available on some CRT's. They are accessed through characters with decimal values above 127. For a list of the characters and their effects, see the "Display Enhancement Characters" table in "Useful Tables" at the back of this book.

Arrays

Entire arrays may be printed using the asterisk specifier. Each element in an array is treated as a separate item, as if the elements were all listed and separated by the punctuation following the array specifier. If no punctuation follows the array specifier, a comma is assumed. The array is printed in row-major order (right-most subscript varies fastest).

PRINT Fields

If PRINT is used without USING, the punctuation following an item determines the width of the item's print field; a semicolon selects the compact field, and a comma selects the default print field. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the print field to be used for the print item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are printed with one trailing blank. String items are printed with no leading or trailing blanks.

The default print field prints items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is printed with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

TAB

The TAB function is used to position the next character to be printed on a line. In the TAB function, a column parameter less than one is treated as one. A column parameter greater than zero is subjected to the following formula: TAB position = ((column − 1) MOD width) + 1; where "width" is 50 for the Model 226 CRT, 128 for Model 237 and 80 for all other devices. If the TAB position evaluates to a column number less than or equal to the number of characters printed since the last EOL sequence, then an EOL sequence is printed, followed by (TAB position − 1) blanks. If the TAB position evaluates to a column number greater than the number of characters printed since the last EOL, sufficient blanks are printed to move to the TAB position.

TABXY

The TABXY function provides X-Y character positioning on the CRT. It is ignored if a device other than the CRT is the PRINTER IS device. TABXY(1,1) specifies the upper left-hand corner of the CRT. If a negative value is provided for CRT row or CRT column, it is an error. Any number greater than the screen width for CRT column is treated as the screen width. Any number greater than 18 for CRT row is treated as 18. (On a Model 237 this is extended to 41 rows). If 0 is provided for either parameter, the current value of that parameter remains unchanged.

PRINT With Using

When the computer executes a PRINT USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the print items, the field specifier is acted upon without accessing the print list. When the field specifer requires characters, it accesses the next item in the print list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching print item. If the image specifiers are exhausted before the print items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than are specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

Effects of the image specifiers on the PRINT statement are shown in the following table:

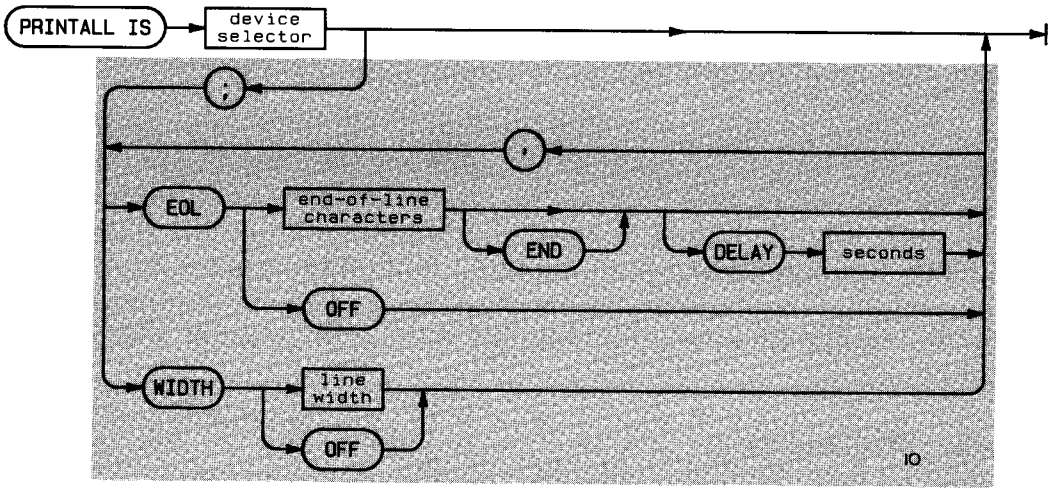
Image Specifier	Meaning
K	Compact field. Prints a number or string in standard form with no leading or trailing blanks.
−K	Same as K.
H	Similar to K, except the number is printed using the European number format (comma radix). (Requires IO)
−H	Same as H. (Requires IO)

Image Specifier	Meaning
S	Prints the number's sign (+ or −).
M	Prints the number's sign if negative, a blank if positive.
D	Prints one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is printed, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are printed.
*	Like Z, except that asterisks are printed instead of leading zeros. (Requires IO)
.	Prints a decimal-point radix indicator.
R	Prints a comma radix indicator (European radix). (Requires IO)
E	Prints an E, a sign, and a two-digit exponent.
ESZ	Prints an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Prints an E, a sign, and a three-digit exponent.
A	Prints a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Prints a blank.
literal	Prints the characters contained in the literal.
B	Prints the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.
W	Prints two characters represented by the two bytes in a 16-bit, two's-complement integer word. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is used; if it is less than −32 768, then −32 768 is used. On an 8-bit interface, the most-significant byte is sent first. On a 16-bit interface, the two bytes are sent as one word in a single operation.
Y	Same as W. (Requires IO)
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence following the last print item.
%	Ignored in PRINT images.
+	Changes the automatic EOL sequence that normally follows the last print item to a single carriage-return. (Requires IO)
−	Changes the automatic EOL sequence that normally follows the last print item to a single line-feed. (Requires IO)
/	Sends a carriage-return and a line-feed to the PRINTER IS device.
L	Sends the current EOL sequence to the PRINTER IS device. The default EOL characters are CR and LF; see PRINTER IS for information on re-defining the EOL sequence. If the destination is an I/O path name with the WORD attribute, a pad byte may be sent after the EOL characters to achieve word alignment.
@	Sends a form-feed to the PRINTER IS device.

PRINTALL IS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement assigns a logging device for recording operator interaction and troubleshooting messages.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
end-of-line characters	string expression; Default = CR/LF	0 thru 8 characters
seconds	numeric expression, rounded to the nearest 0.001 seconds; Default = 0	0.001 thru 32.767
line width	numeric expression, rounded to an integer; Default = infinity (see text)	1 thru 32 767

Example Statements

```
PRINTALL IS 701
PRINTALL IS Gpio
PRINTALL IS 701;EOL CHR$(13) END,WIDTH 65
```

Semantics

The printall device must be enabled by the [PRT ALL] key on the computer. The [PRT ALL] key is a toggle action device, enabling and disabling the printall operation. When the printall mode is enabled, all items generated by DISP, all operator input followed by the [RETURN], [ENTER], [CONTINUE], or [EXECUTE] key, and all error messages from the computer are logged on the printall device. All TRACE activity is logged on the printall device if tracing is enabled.

An asterisk (*) is displayed on the PRINTALL softkey label of models with HP 46020A keyboards, if print all mode is enabled.

At power-on and SCRATCH A, the printall device is the CRT (device selector = 1).

The EOL Attribute (Requires IO)

The EOL attribute re-defines the end-of-line (EOL) sequence, which is sent at the following times: after the number of characters specified by *line width* and after each line of text. Up to eight characters may be specified as the EOL characters; an error is reported if the string contains more than eight characters. If END is included in the EOL attribute, an interface-dependent END indication is sent with the last character of the EOL sequence. If DELAY is included, the computer delays the specified number of seconds (after sending the last character) before continuing. The default EOL sequence consists of a carriage-return and a line-feed character with no END indication and no delay period.

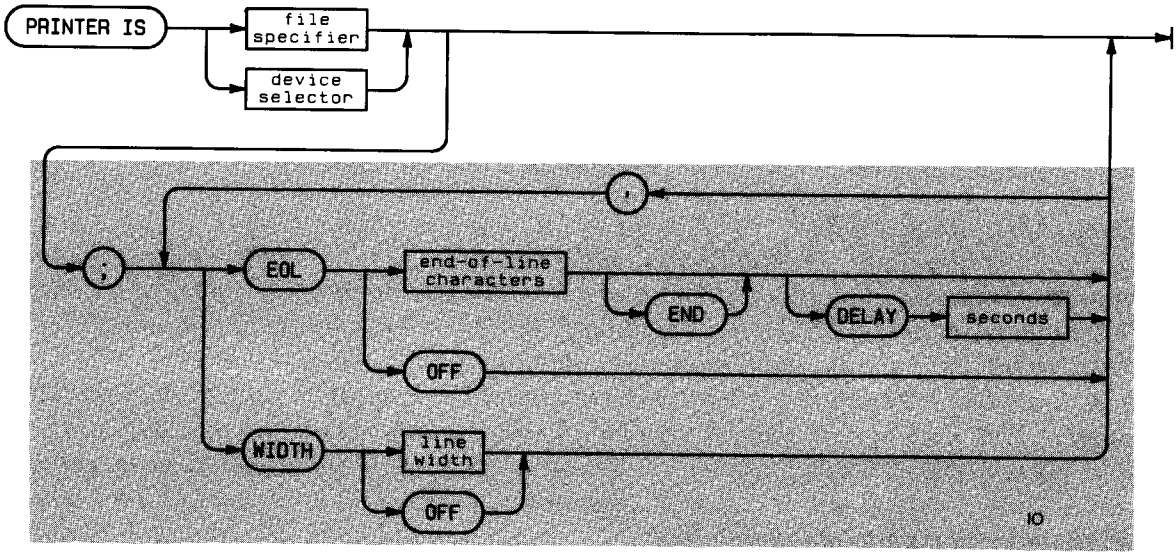
The WIDTH Attribute (Requires IO)

The WIDTH attribute specifies the maximum number of characters which will be sent to the printing device before an EOL sequence is automatically sent. The EOL characters are not counted as part of the line width. The default width for the Model 226 CRT is 50, Model 237 with HP 98781A CRT is 128, and the default for all other devices is 80. Specifying WIDTH OFF sets the width to infinity. If the default is desired, it must be restored explicitly. If the USING clause is included in the PRINT statement, the WIDTH attribute is ignored.

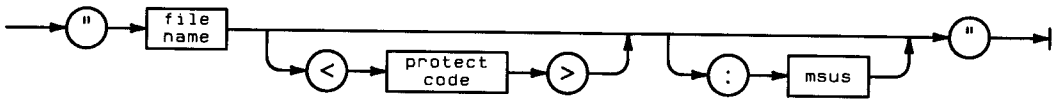
PRINTER IS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the system printing device or file. (If using PRINTER IS with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
end-of-line characters	string expression; Default = CR/LF	0 thru 8 characters
seconds	numeric expression, rounded to the nearest 0.001 seconds; Default = 0	0.001 thru 32.767
line width	numeric expression, rounded to an integer; Default = (see text)	1 thru 32 767
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
PRINTER IS 701
PRINTER IS Gpio
PRINTER IS 701;EOL CHR$(13) END,WIDTH 65
PRINTER IS "Myfile";WIDTH 80
PRINTER IS "Spooler:REMOTE"
```

Semantics

The system printing device or file receives all data sent by the PRINT statement and all data sent by CAT and LIST statements in which the destination is not explicitly specified.

The default printing device is the CRT (select code 1) at power-on and after executing SCRATCH A.

The EOL Attribute (Requires IO)

The EOL attribute re-defines the end-of-line (EOL) sequence, which is sent at the following times: after the number of characters specified by *line width*, after each line of text, and when an "L" specifier is used in a PRINT USING statement. Up to eight characters may be specified as the EOL characters; an error is reported if the string contains more than eight characters. If END is included in the EOL attribute, an interface-dependent END indication is sent with the last character of the EOL sequence. If DELAY is included, the computer delays the specified number of seconds (after sending the last character) before continuing. The default EOL sequence consists of a carriage-return and a line-feed character with no END indication and no delay period. END and DELAY are ignored for files.

The WIDTH Attribute (Requires IO)

The WIDTH attribute specifies the maximum number of characters which will be sent to the printing device before an EOL sequence is automatically sent. The EOL characters are not counted as part of the line width. The default width for the Model 226 CRT is 50, Model 237 with HP 98781A CRT is 128, and the default for all other devices is 80. Specifying WIDTH OFF sets the width to infinity. If the default is desired, it must be restored explicitly. If the USING clause is included the PRINT statement, the WIDTH attribute is ignored. Default WIDTH for files is OFF.

PRINTER IS file

The file must be a BDAT file.

The PRINTER IS file statement positions the file pointer to the beginning of the file.

The file is closed when another PRINTER IS statement is executed and at SCRATCH A.

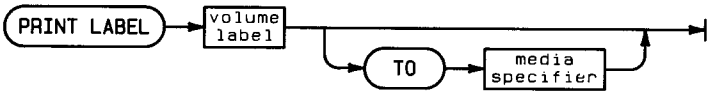
An end of file error occurs when the end of the file is reached.

You can read the file with ENTER if it is ASSIGNed with FORMAT ON.

PRINT LABEL

Option Required	MS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement gives a name to a mass storage volume.



Item	Description/Default	Range Restrictions
volume label	Name to be given to the volume	—
media specifier	string expression; Default = the default mass storage unit	(see MASS STORAGE IS)

Example Statements

```
PRINT LABEL "Vers3" TO ":INTERNAL"
PRINT LABEL Volume$ TO Msus$
```

Semantics

The new name overrides any previous name for the volume.

The volume label can be zero to six characters in length consisting of letters and numbers. For maximum interchange, the characters should be limited to uppercase letters (A-Z) and digits (0-9) with the first character being a letter.

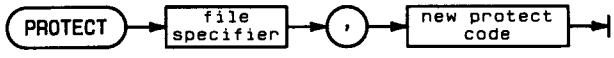
PRIORITY

See the SYSTEM PRIORITY statement.

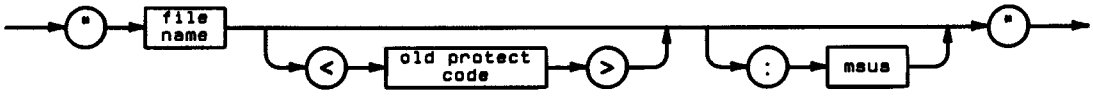
PROTECT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the protect code used on PROG, BDAT, and BIN files. (If using PROTECT with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
new protect code	string expression; first two non-blank characters are significant	">" not allowed
file name	literal	any valid file name
old protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
PROTECT Name$,Pc$
PROTECT "George<xy>:INTERNAL","NEW"
```

Semantics

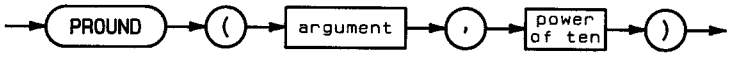
A protect code guards against accidental changes to an individual file. Once a file is protected, the protect code must be included in its file specifier for all operations except LOAD and LOADSUB.

Protect codes are trimmed before they are used. Therefore, leading and trailing blanks are insignificant. Removing a protect code from a file is accomplished by assigning a protect code that is the null string or contains all blanks.

PROUND

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of the argument rounded to the specified power-of-ten.



Item	Description/Default	Range Restrictions
argument	numeric expression	—
power of ten	numeric expression, rounded to an integer	—

Example Statements

```
Money=PROUND(Result,-2)
PRINT PROUND(Quantity,Decimal_place)
```

PRT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This INTEGER function returns 701, the default (factory set) device selector for an external printer.



Example Statements

```
PRINTER IS PRT
OUTPUT PRT;A$
```

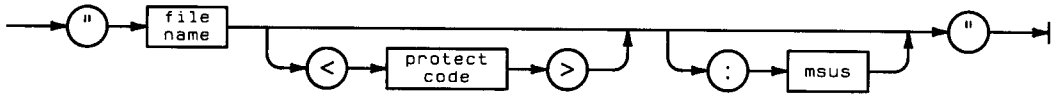
PURGE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deletes a file entry from the directory of the mass storage media (If using PURGE with SRM, also refer to the "SRM" section of this manual.).



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
PURGE Name$
PURGE "George<PC>"
```

Semantics

Once a file is purged, you cannot access the information which was in the file. The records of a purged file are returned to "available space." An open file must be closed before it can be purged. Any file can be closed by ASSIGN TO * (see ASSIGN).

RAD

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects radians as the unit of measure for expressing angles.



Semantics

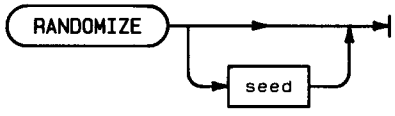
All functions which return an angle will return an angle in radians. All operations with parameters representing angles will interpret the angle in radians. If no angle mode is specified in a program, the default is radians (also see DEG).

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

RANDOMIZE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a seed for the RND function.



Item	Description/Default	Range Restrictions	Recommended Range
seed	numeric expression, rounded to an integer; Default = pseudorandom	—	1 thru $2^{31}-2$

Example Statements

```
RANDOMIZE
RANDOMIZE Old_seed*PI
```

Semantics

The seed actually used by the random number generator depends on the absolute value of the seed specified in the RANDOMIZE statement.

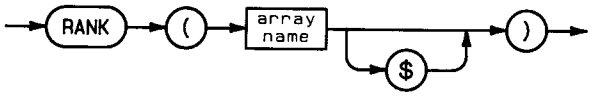
Absolute Value of Seed	Value Used
less than 1	1
1 thru $2^{31}-2$	INT(ABS(seed))
greater than $2^{31}-2$	$2^{31}-2$

The seed is reset to 37 480 660 by power-up, SCRATCH A, SCRATCH, and program prerun.

RANK

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the number of dimensions in an array. The value returned is an INTEGER.



Item	Description/Default	Range Restrictions
array name	name of an array	any valid name

Example Statement

```
IF RANK(A)=2 THEN PRINT "A is a matrix"
R=RANK(Array)
```

RATIO

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the ratio of the X hard clip limits to the Y hard clip limits for the current PLOTTER IS device.



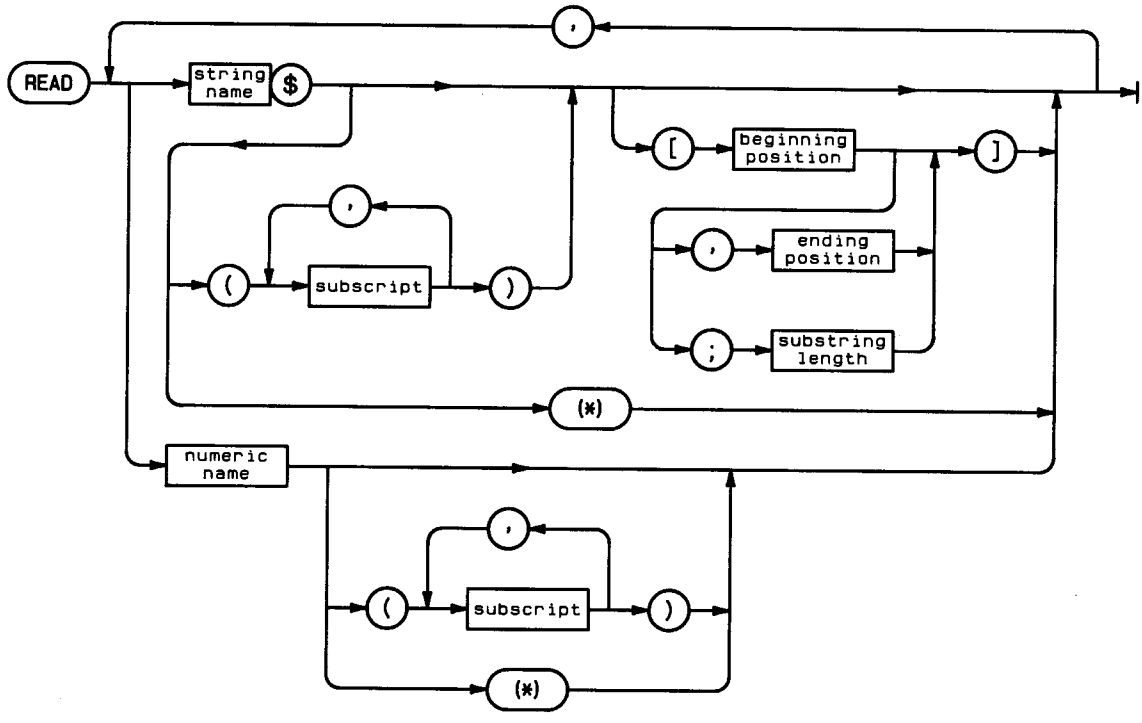
Example Statements

```
WINDOW 0,10*RATIO,-10,10
Turn=1/RATIO
```

READ

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement reads values from DATA statements and assigns them to variables.



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
READ Number,String$
READ Array(*)
READ Item(1,1),Item(2,1),Item(3,1)
```

Semantics

The numeric items stored in DATA statements are considered strings by the computer, and are processed with a VAL function to be read into numeric variables in a READ statement. If they are not of the correct form, error 32 may result. Real DATA items will be rounded into an INTEGER variable if they are within the INTEGER range (−32 768 thru 32 767). A string variable may read numeric items, as long as it is dimensioned large enough to contain the characters.

The first READ statement in a context accesses the first item in the first DATA statement in the context unless RESTORE has been used to specify a different DATA statement as the starting point. Successive READ operations access following items, progressing through DATA statements as necessary. Trying to READ past the end of the last DATA statement results in error 36. The order of accessing DATA statements may be altered by using the RESTORE statement.

An entire array can be specified by replacing the subscript list with an asterisk. The array entries are made in row major order (right most subscript varies most rapidly).

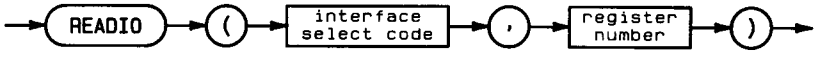
READIO

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function reads the contents of the specified hardware register on the specified interface.

Note

Unexpected results may occur with select codes outside the given range.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	1 thru 31
register number	numeric expression, rounded to an integer	interface dependent

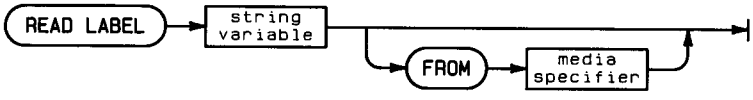
Example Statements

```
Upper_byte=READIO(Gpio,4)
PRINT "Register";I;"=";READIO(7,I)
```

READ LABEL

Option Required	MS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement reads a volume label into a string variable.



Item	Description/Default	Range Restrictions
string variable	string variable which returns the volume name	—
media specifier	string expression; Default = the default mass storage unit	(see MASS STORAGE IS)

Example Statements

```
READ LABEL Volume_name$ FROM ":INTERNAL"
IF Inserted$="Yes" THEN READ LABEL Volume$ FROM msus$
```

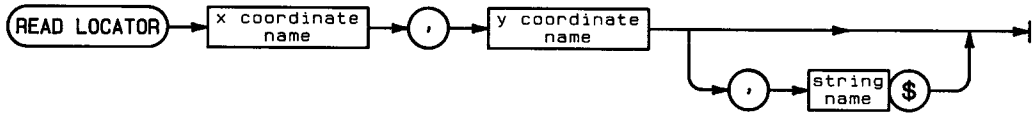
Semantics

A LIF volume label consists of a maximum of 6 characters, letters, and digits. Other volumes can return labels up to 16 characters.

READ LOCATOR

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement samples the locator device, without waiting for a digitizing operation.



Item	Description/Default	Range Restrictions
x coordinate name	name of a numeric variable	any valid name
y coordinate name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name

Example Statements

```
READ LOCATOR X_pos,Y_pos
READ LOCATOR X,Y,Status$
```

Semantics

Executing this statement issues a request to the current locator device to return a set of coordinates. The coordinates are sampled immediately, without waiting for a digitizing action on the part of the user. GRAPHICS INPUT IS is used to establish the current locator device. The returned coordinates are in the unit-of-measure currently defined for the PLOTTER IS and GRAPHICS INPUT IS devices. The unit-of-measure may be default units or those defined by either the WINDOW or SHOW statement. If an INTEGER numeric variable is specified, and the value returned is out of range, Error 20 is reported.

The optional string variable is used to input the device status of the GRAPHICS INPUT IS device. This status string contains eight bytes, defined as follows.

Byte	1	2	3	4	5	6	7	8
Meaning	Digitize Status	,	Point Significance	,	Tracking On/Off	,	Button Number	

Byte 1: Button status; This value represents the status of the digitizing button on the locator. A "0" means the button is not depressed, and a "1" means the button is depressed. This is an unprocessed value, and a "1" does not necessarily represent successful digitization. If the numeric value represented by this byte is used as the pen control value for a PLOT statement, continuous digitizing will be copied to the display device.

Bytes 2, 4, and 6: commas; used as delimiters.

Bytes 3: Significance of digitized point; "0" indicates that the point is outside the P1, P2 limits; "1" indicates that the point is outside the viewport, but inside the P1, P2 limits; "2" indicates that the point is inside the current viewport limits.

Byte 5: Tracking status; "0" indicates off, "1" indicates on.

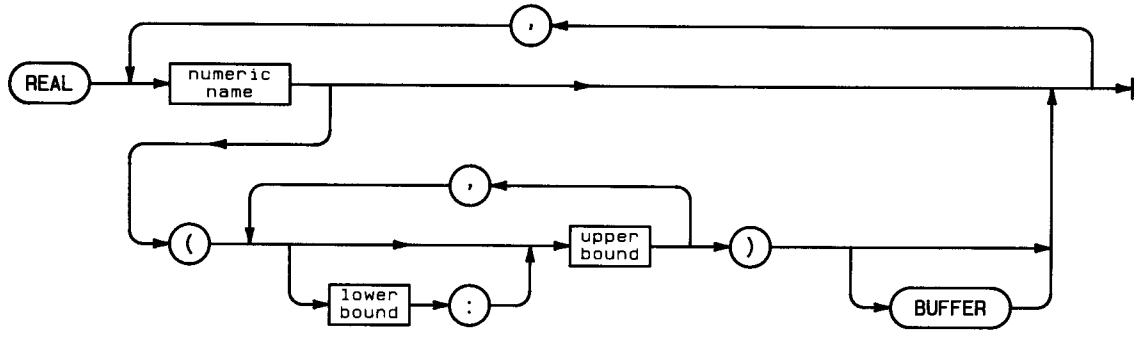
Byte 7 and 8: The number of the buttons which are currently down. To interpret the ASCII number returned, change the number to its binary form and look at each bit. If the bit is "1", the corresponding button is down. If the bit is "0", the corresponding button is not down.

If the locator device (e.g., stylus or puck) goes out of proximity, a "button 7" is indicated in the "button number" bytes. The number will be exactly "64", regardless of whether any actual buttons are being held down at the time. The HP 9111A always returns "00" in bytes 7 and 8.

REAL

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement reserves storage for floating point variables and arrays. (For information about REAL as a secondary keyword, see the ALLOCATE, COM, DEF FN, or SUB statements.)



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)

Example Statements

```
REAL X,Y,Z
REAL Array(-128:127,15)
REAL A(512) BUFFER
```

Semantics

Each REAL variable or array element requires eight bytes of number storage. The maximum number of subscripts in an array is six, and no dimension may have more than 32 767 elements. The total number of elements in an array is limited by memory.

Declaring Buffers

To declare REAL variables to be buffers, each variable's name must be followed by the keyword BUFFER; the designation BUFFER applies only to the variable which it follows.

RECORDS

See the TRANSFER statement.

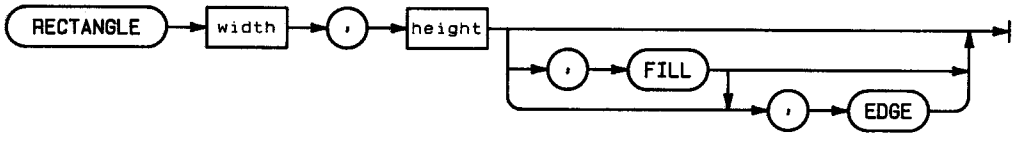
RECOVER

See the ON... statements.

RECTANGLE

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a rectangle. It can be filled, edged, or both.



Item	Description/Default	Range Restrictions
width	numeric expression	—
height	numeric expression	—

Example Statements

```
RECTANGLE 4,6
RECTANGLE 3,-2,FILL,EDGE
```

Semantics

The rectangle is drawn with dimensions specified as displacements from the current pen position. Thus, both the width and the height may be negative.

Which corner of the rectangle is at the pen position at the end of the statement depends upon the signs of the parameters:

Sign of X	Sign of Y	Corner of Rectangle at Pen Position
+	+	Lower left
+	−	Upper left
−	+	Lower right
−	−	Upper right

Shape of Rectangle

A rectangle's shape is affected by the current viewing transformation. If isotropic units are in effect, the rectangle will be the expected shape, but if anisotropic units are in effect the rectangle will be distorted; stretched or compressed along the axes.

RECTANGLE is affected by the PIVOT and PDIR transformations. If a rotation transformation *and* anisotropic units are in effect, the rectangle is rotated first, then stretched or compressed along the unrotated axes.

FILL and EDGE

FILL causes the rectangle to be filled with the current fill color, and EDGE causes the perimeter to be drawn with the current pen color and line type. If both FILL and EDGE are specified, the interior will be filled, then the edge will be drawn. If neither FILL nor EDGE is specified, EDGE is assumed.

Polygons sent to an HPGL plotter are edged but not filled regardless of any FILL or EDGE directives on the statement.

Applicable Graphics Transformations

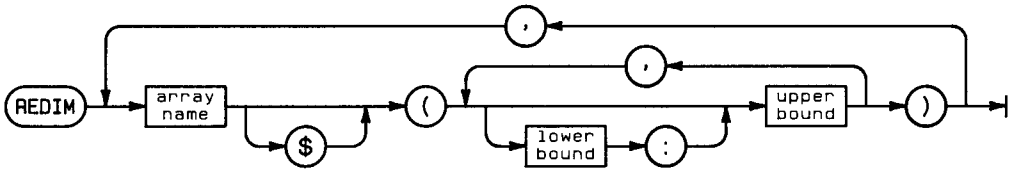
	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

REDIM

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement changes the subscript range of previously dimensioned arrays.



Item	Description/Default	Range Restrictions
array name	name of an array	any valid name
lower bound	numeric expression, rounded to an integer; Default = OPTION BASE value (0 or 1)	−32 768 thru +32 767 (see "array" in glossary)
upper bound	numeric expression, rounded to an integer	−32 768 thru +32 767 (see "array" in glossary)

Example Statements

```
REDIM Array(5)
REDIM B(3:5,6,-2:2)
REDIM Constants$(X,Y,Z)
```

Semantics

The following rules must be followed when redimensioning an array:

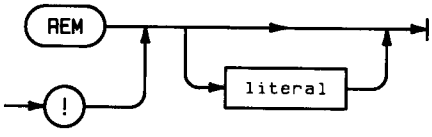
- The array to be redimensioned must have a currently dimensioned size known to the context (i.e., it must have been implicitly or explicitly dimensioned, or be currently allocated, or it must have been passed into the context.)
- You must retain the same number of dimensions as specified in the original dimension statement.
- The redimensioned array cannot have more elements than the array was originally dimensioned to hold.
- You cannot change the maximum string length of string arrays.

REDIM does not change any values in the array, although their locations will probably be different. The REDIM is performed left-to-right and if an error occurs, arrays to the left of the array the error occurs in will be redimensioned while those to the right will not be. If an array appears more than once in the REDIM, the rightmost dimensions will be in effect after the REDIM.

REM

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement allows comments in a program.



Item	Description/Default	Range Restrictions
literal	string constant composed of characters from the keyboard, including those generated with the ANY CHAR key	—

Example Program Lines

```
100  REM   Program Title
190  !
200  IF BIT(Info,2) THEN Branch  ! Test overrange bit
```

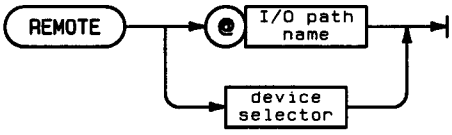
Semantics

REM must be the first keyword on a program line. If you want to add comments to a statement, an exclamation point must be used to mark the beginning of the comment. If the first character in a program line is an exclamation point, the line is treated like a REM statement and is not checked for syntax.

REMOTE

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement places HP-IB devices having remote/local capabilities into the remote state.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
REMOTE 712
REMOTE @Hpib
```

Semantics

If individual devices are not specified, the remote state for all devices on the bus having remote/local capabilities is enabled. The bus configuration is unchanged, and the devices switch to remote if and when they are addressed to listen. If primary addressing is used, only the specified devices are put into the remote state.

When the computer is the system controller and is switched on, reset, or ABORT is executed, bus devices are automatically enabled for the remote state and switch to remote when they are addressed to listen.

The computer must be the system controller to execute this statement, and it must be the active controller to place individual devices in the remote state.

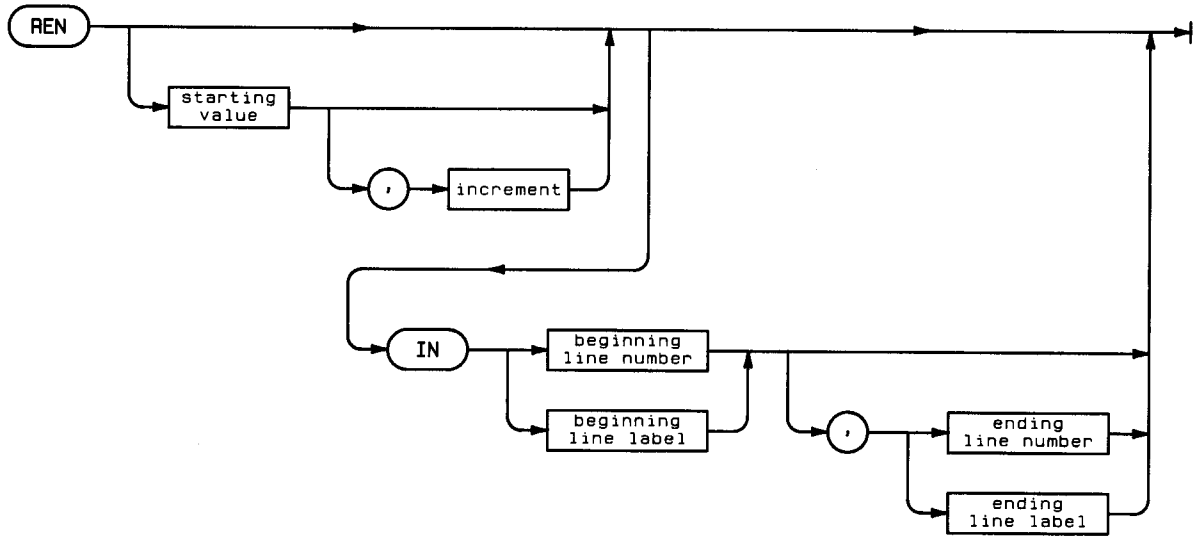
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	REN $\overline{\text{ATN}}$	REN ATN MTA UNL LAG	Error	
Not Active Controller	REN	Error	Error	

REN

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command allows you to renumber all or a portion of the program currently in memory.



Item	Description/Default	Range Restrictions
starting value	integer constant identifying a program line; Default = 10	1 thru 32 766
increment	integer constant; Default = 10	1 thru 32 767
beginning line number	integer constant identifying program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
REN
REN 1000,5
REN 270,1 IN 260,Label1
```

Semantics

The program segment to be renumbered is delimited by the beginning line number or label (or the first line in the program) and the ending line number or label (or the last line in the program). The first line in the renumbered segment is given the specified starting value, and subsequent line numbers are separated by the increment. If a renumbered line is referenced by a statement (such as GOTO or GOSUB), those references will be updated to reflect the new line numbers. Renumbering a paused program causes it to move to the stopped state.

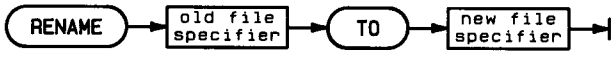
REN cannot be used to move lines. If renumbering would cause lines to overlap preceding or following lines, an error occurs and no renumbering takes place.

If the highest line number resulting from the REN command exceeds 32 766, an error message is displayed and no renumbering takes place. An error occurs if the beginning line is after the ending line, or if one of line labels specified doesn't exist.

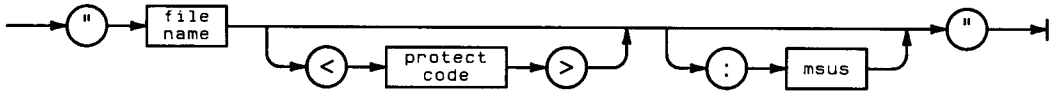
RENAME

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement changes a file's name in the mass storage media's directory. (If using RENAME with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
old file specifier	string expression	(see drawing)
new file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
RENAME "TEMP<PC>" TO "FINAL"
RENAME Name$&Msus$ TO Temp$
```

Semantics

The new file name must not duplicate the name of any other file in the directory. A protected file retains its old protect code, which must be included in the old file specifier. Because you cannot move a file from one mass storage device to another with RENAME, the msus of the new file specifier is ignored.

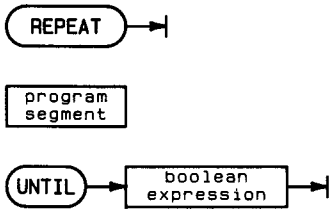
REORDER

See the MAT REORDER statement.

REPEAT...UNTIL

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the boolean expression in the UNTIL statement evaluates to be logically true (evaluates to non-zero).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if zero	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
530  REPEAT
540    PRINT Factor
550    Factor=Factor*1.1
560  UNTIL Factor>10

680  REPEAT
690    INPUT "Enter a positive number",Number
700  UNTIL Number>=0
```

Semantics

The REPEAT...UNTIL construct allows program execution dependent on the outcome of a relational test performed at the **end** of the loop. Execution starts with the first program line following the REPEAT statement, and continues to the UNTIL statement where a relational test is performed. If the test is false a branch is made to the first program line following the REPEAT statement.

When the relational test is true, program execution continues with the first program line following the UNTIL statement.

Branching into a REPEAT...UNTIL construct (via a GOTO) results in normal execution up to the UNTIL statement, where the test is made. Execution will continue as if the construct had been entered normally.

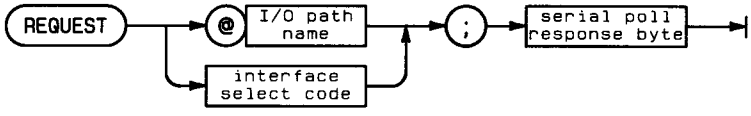
Nesting Constructs Property

REPEAT...UNTIL constructs may be nested within other constructs provided the inner construct begins and ends before the outer construct can end.

REQUEST

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used by a non-active controller to send a Service Request (SRQ) on an HP-IB interface.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an HP-IB interface	any valid name
interface select code	numeric expression, rounded to an integer	7 thru 31
serial poll response byte	numeric expression, rounded to an integer	0 thru 255

Example Statements

```
REQUEST @Hp_ib;Bit_6+Bit_0
REQUEST Isc;Response
```

Semantics

To request service, the value of the serial poll response must have bit 6 set; this bit asserts the SRQ line. SRQ will remain set until either the Active Controller performs a Serial Poll or until the computer executes another REQUEST with bit 6 clear.

Only the interface select code may be specified to receive the Request; if a device selector that contains address information, or an I/O path assigned to a device selector with address information is specified, an error results. An error will also results if the computer is currently the Active Controller.

RES

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This function returns the result of the last numeric computation which was executed from the keyboard.

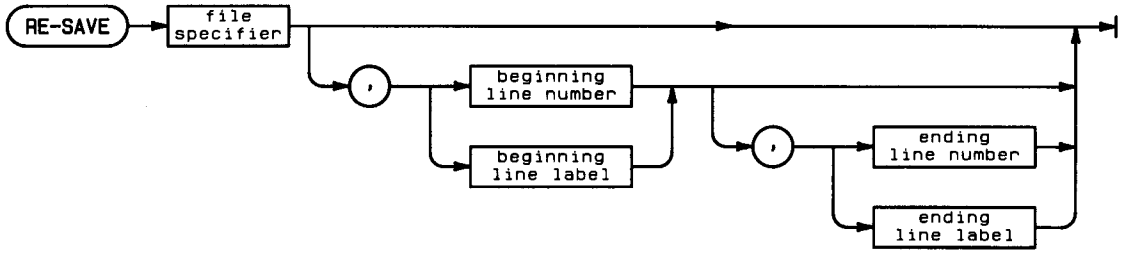
Example Statements

```
RES
3.5*RES+A
```

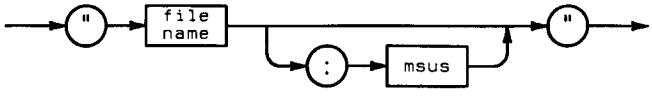
RE-SAVE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file and copies program lines as strings into that file. (If using RE-SAVE with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
beginning line number	integer constant identifying program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name
file name	literal	any valid file name
msus	literal	(see MASS STORAGE IS)

Example Statements

```
RE-SAVE "Nailfile"
RE-SAVE Name$,1,Sort
```

Semantics

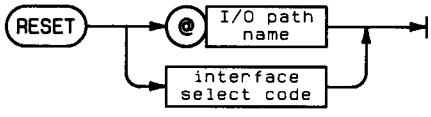
An entire program can be saved, or the portion delimited by beginning and (if needed) ending line labels or line numbers. If the file name already exists, the old file entry is removed from the directory after the new file is succesfully saved on the mass storage media. Pressing (RESET) during a RE-SAVE operation results in the old file being retained. Attempting to RE-SAVE any file that is not an ASCII file results in an error.

If a specified line label does not exist, error 3 occurs. If a specifed line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs.

RESET

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement resets an interface or the pointers of either a mass storage file or buffer. (For information about RESET as a Secondary keyword, see the SUSPEND INTERACTIVE statement. If using RESET with SRM, also refer to the "SRM" section of this manual.)



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface, mass storage file, or buffer	any valid name
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
RESET Hpib
RESET 20
RESET @Buffer_x
```

Semantics

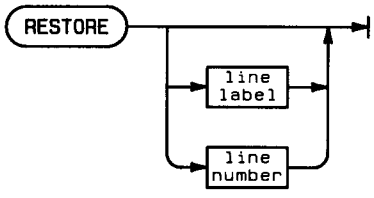
A RESET directed to an interface initiates an interface-dependent action; see the "Interface Registers" section for further details. A RESET directed to a mass storage file resets the file pointer to the beginning of the file. A RESET directed to a buffer resets all registers to their initial values: the empty and fill pointers are set to 1, and the current-number-of-bytes and all other registers are reset to zero.

If a TRANSFER is currently being made to or from the specified resource, the computer waits until the TRANSFER is complete before executing the RESET. If the TRANSFER is not to be completed, an ABORTIO may be executed to halt the TRANSFER before executing the RESET. If a busy buffer is specified in a RESET statement, error 612 results.

RESTORE

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

RESTORE specifies which DATA statement will be used by the next READ operation.



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line; Default = first DATA statement in context	1 thru 32 766

Example Statements

```
RESTORE
RESTORE Third_array
```

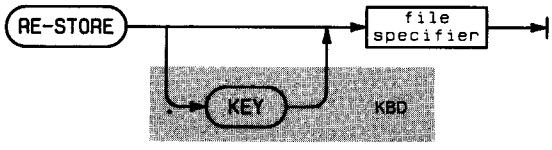
Semantics

If a line is specified which does not contain a DATA statement, the computer uses the first DATA statement after the specified line. RESTORE can only refer to lines within the current context. An error results if the specified line does not exist.

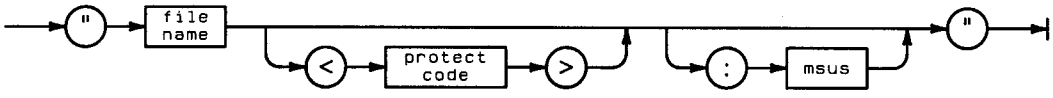
RE-STORE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a file and stores the program or typing-aid key definitions into it. (If using RE-STORE with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
RE-STORE Filename$&Msus$
RE-STORE KEY "KEYS"
```

Semantics

If the file already exists, the old file is removed from the directory after the new file is successfully saved on the mass storage media. If an old file does not exist, a new one is created as if this were the STORE statement. Pressing (RESET) during a RE-STORE operation causes the old file to be retained. If the old file had a protect code, the same protect code must be used in the RE-STORE operation. Attempting to RE-STORE a file which is the wrong type results in an error. (RE-STORE creates a PROG file, and RE-STORE KEY creates a BDAT file.)

RESUME INTERACTIVE

Option Required	None
Keyboard Executable	Yes[1]
Programmable	Yes
In an IF...THEN...	Yes

This statement enables the [EXECUTE], [ENTER], [RETURN], [PAUSE], [STOP], [STEP], [CLR I/O], [BREAK] and [RESET] keys after a SUSPEND INTERACTIVE statement.



Example Statements

```
RESUME INTERACTIVE
IF Kbd_flag THEN RESUME INTERACTIVE
```

1 This statement is executable from the keyboard, but onlywhile SUSPEND INTERACTIVE is **not** in effect.

RETURN

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement returns program execution to the line following the invoking GOSUB. The keyword RETURN is also used in user-defined functions (see DEF FN).



REV$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a string formed by reversing the sequence of characters in the specified string.



Example Statements

```
Reverse$=REV$("palindrome")
Last_blank=LEN(Sentence$)-POS(REV$(Sentence$)," ")
```

Semantics

The REV$ function is useful when searching for the last occurrence of an item within a string.

RND

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a pseudo-random number greater than 0 and less than 1.



Example Statements

```
Percent=RND*100
IF RND<.5 THEN Case1
```

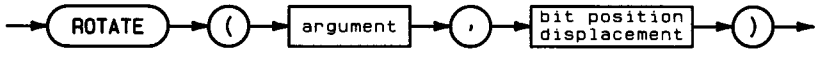
Semantics

The random number returned is based on a seed set to 37 480 660 at power-on, SCRATCH, SCRATCH A, or program prerun. Each succeeding use of RND returns a random number which uses the previous random number as a seed. The seed can be modified with the RANDOMIZE statement.

ROTATE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified. The shift is performed with wraparound.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	—
bit position displacement	numeric expression, rounded to an integer	−32 768 thru +32 767	−15 thru +15

Example Statements

```
New_word=ROTATE(Old_word,2)
Q=ROTATE(Q,Places)
```

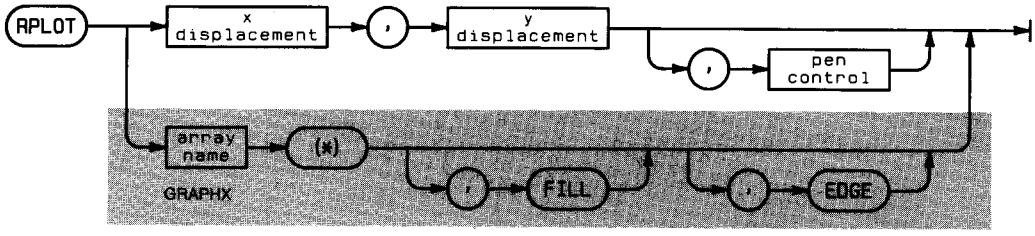
Semantics

The argument is converted into a 16-bit, two's-complement form. If the bit position displacement is positive, the rotation is towards the least-significant bit. If the bit position displacement is negative, the rotation is towards the most-significant bit. The rotation is performed without changing the value of any variable in the argument.

RPLOT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement moves the pen from the current pen position to the point specified by adding the x and y displacements to the local origin. It can be used to move with or without drawing a line depending on the pen control parameter.



Item	Description/Default	Range Restrictions
x displacement	numeric expression in current units	—
y displacement	numeric expression in current units	—
pen control	numeric expression, rounded to an integer; Default = 1	−32 768 thru +32 767
array name	name of two-dimensional, two-column or three-column numeric array. Requires GRAPHX	any valid name

Example Statements

```
RPLOT Rel_x,Rel_y,Pen_action
RPLOT 5,12
RPLOT Shape(*),FILL,EDGE
```

Semantics

This statement moves the pen to the specified X and Y coordinates relative to the local coordinate origin. Both moves and draws may be generated, depending on the pen control parameter. Lines are drawn using the current pen color and line type.

The local coordinate origin is the logical pen position at the completion of one of the following statements. The local coordinate origin is **not** changed by the RPLOT statement.

AXES DRAW FRAME GINIT GRID IDRAW IMOVE
IPLOT LABEL MOVE PLOT POLYGON POLYLINE RECTANGLE
SYMBOL

The line is clipped at the current clipping boundary. RPLOT is affected by the PIVOT and PDIR transformations. If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Non-Array Parameters

The specified X and Y displacements information is interpreted according to the current unit-of-measure. Lines are drawn using the current pen color and line type.

If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

The optional pen control parameter specifies the following plotting actions; the default value is +1 (down after move).

Pen Control Parameter

Pen Control	Resultant Action
−Even	Pen up before move
−Odd	Pen down before move
+Even	Pen up after move
+Odd	Pen down after move

The above table is summed up by: even is up, odd is down, positive is after pen motion, negative is before pen motion.

Array Parameters

When using the RPLOT statement with an array, either a two-column or a three-column array may be used. If a two-column array is used, the third parameter is assumed to be +1: pen down after move.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the ranges 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the RPLOT statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the RPLOT statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the RPLOT statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

When using an RPLOT statement with an array, the following list of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth.

Column 1	Column 2	Operation Selector	Meaning
X	Y	−2	Pen up before moving
X	Y	−1	Pen down before moving
X	Y	0	Pen up after moving (Same as +2)
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	ignored	3	Select pen
line type	repeat value	4	Select line type
color	ignored	5	Color value
ignored	ignored	6	Start polygon mode with FILL
ignored	ignored	7	End polygon mode
ignored	ignored	8	End of data for array
ignored	ignored	9	NOP (no operation)
ignored	ignored	10	Start polygon mode with EDGE
ignored	ignored	11	Start polygon mode with FILL and EDGE
ignored	ignored	12	Draw a FRAME
pen number	ignored	13	Area pen value
red value	green value	14	} Color
blue value	ignored	15	} Value
ignored	ignored	>15	Ignored

Moving and Drawing
If the operation selector is less than or equal to two, it is interpreted in exactly the same manner as the third parameter in a non-array RPLOT statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens
An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types
An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color
Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color
Operation Selector 14 is used in conjunction with Operation Selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessariy immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation Selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word, that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities (which range from 0 thru 1) for red, green, and blue in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons
A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME
Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the RPLOT statement, so one probably would not have more than one operation selector 12 in an array to RPLOT, since the last FRAME will overwrite all the previous ones.

Premature Termination
Operation selector 8 causes the RPLOT statement to be terminated. The RPLOT statement will successfully terminate if the actual end of the array has been reached, so use of operation selector 8 is optional.

Ignoring Selected Rows in the Array
Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

RPT$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the string repeated a given number of times.



Item	Description/Default	Range Restrictions
argument	string expression	—
repeat factor	numeric expression, rounded to an integer	0 thru 32 767

Example Statements

```
PRINT RPT$("*",80)
Center$=RPT$(" ",(Right-Left-Length)/2)
```

Semantics

The value of the numeric expression is rounded to an integer. If the numeric expression evaluates to a zero, a null string is returned.

An error will result if the numeric expression evaluates to a negative number or if the string created by RPT$ contains more than 32 767 characters.

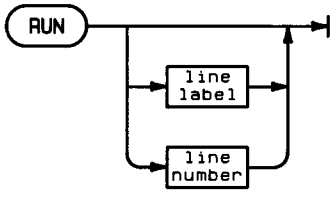
RSUM

See the MAT statement.

RUN

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command starts program execution at a specified line.



Item	Description/Default	Range Restrictions
line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
RUN 10
RUN Part2
```

Semantics

Pressing the [RUN] key is the same as executing RUN with no label or line number. RUN is executed in two phases: prerun initialization and program execution.

The prerun phase consists of:

- Reserving memory space for variables specified in COM statements (both labeled and blank). See COM for a description of when COM areas are initialized.
- Reserving memory space for variables specified by DIM, REAL, INTEGER, or implied in the main program segment. This does not include variables used with ALLOCATE, which is done at run-time. Numeric variables are initialized to 0; string variables are initialized to the null string.
- Checking for syntax errors which require more than one program line to detect. Included in this are errors such as incorrect array references, and mismatched parameter or COM lists.

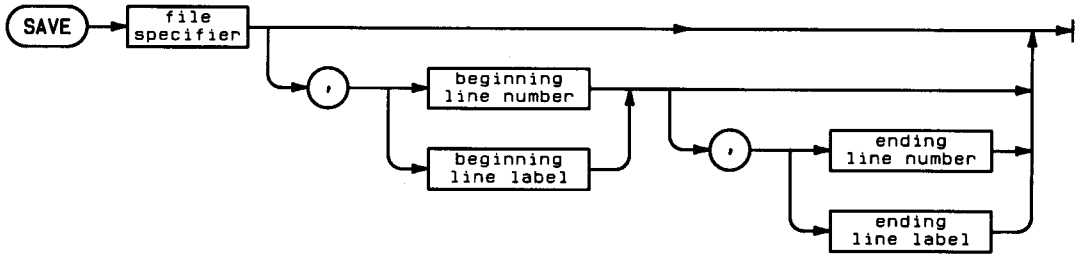
If an error is detected during prerun phase, prerun halts and an error message is displayed on the CRT.

After successful completion of prerun initialization, program execution begins with either the lowest numbered program line or the line specified in the RUN command. If the line number specified does not exist in the main program, execution begins at the next higher-numbered line. An error results if there is no higher-numbered line available within the main program, or if the specified line label cannot be found in the main program.

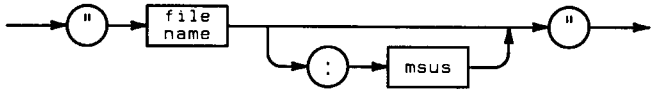
SAVE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file and copies program lines as strings into that file. (If using SAVE with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name
file name	literal	any valid file name
msus	literal	(see MASS STORAGE IS)

Example Statements

```
SAVE "WHALES"
SAVE "TEMP",1,Sort
```

Semantics

An entire program can be saved, or any portion delimited by the beginning and (if needed) ending line numbers or labels. This statement is for creating new files. Attempting to SAVE a file name that already exists causes error 54. If you need to replace an old file, see RE-SAVE.

If a specified line label does not exist, error 3 occurs. If a specified line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs. If no program lines are in the specified range, error 46 occurs.

SC

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the interface select code associated with an I/O path name.



Item	Description/Default	Range Restrictions
I/O path name	name of a currently assigned I/O path	any valid name

Example Statements

```
Isc=SC(@Device)
Drive_isc=SC(@File)
```

Semantics

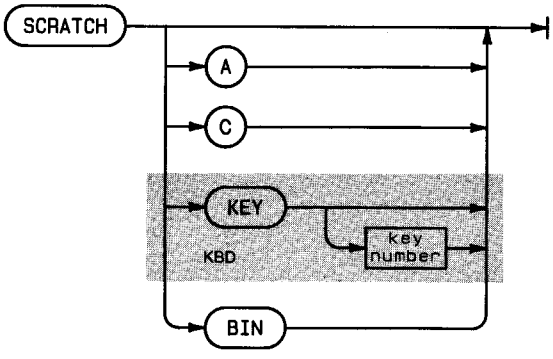
If the I/O path name is assigned to a device selector (or selectors) with primary and/or secondary addressing, only the interface select code is returned. If the specified I/O path name is assigned to a mass storage file, the interface select code of the drive is returned. If the specified I/O path name is assigned to a buffer, a zero is returned.

If the I/O path name is not currently assigned to a resource, an error is reported.

SCRATCH

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command erases all or selected portions of memory.



Item	Description/Default	Range Restrictions
key number	integer constant	0 thru 23

Example Statements

```
SCRATCH
SCRATCH A
SCRATCH KEY
SCRATCH KEY 14
```

Semantics

SCRATCH clears the BASIC program and all variables not in COM. Key definitions are left intact.

SCRATCH C clears all variables, including those in COM. The program and keys are left intact.

To scratch a key, type `SCRATCH KEY`, followed by the key number, and press EXECUTE, ENTER or RETURN. Also, pressing a softkey after typing `SCRATCH` will cause `SCRATCH KEY`, followed by the key number, to be displayed. When a key is specified, the definition for that key only is cleared. When an individual key is not specified, all key definitions are cleared. In either case, the program and all variables are left intact.

SCRATCH A clears the BASIC program memory, all the key definitions, and all variables (including those in COM). Most internal parameters in the computer are reset by this command. The clock is not reset and the recall buffer is not cleared. See the Master Reset Table in the "Useful Tables" section in the back of this manual for details.

SCRATCH BIN
SCRATCH BIN causes an extended SCRATCH A. It resets the computer to its power up state. All programs, variables, and BINs are deleted from memory. The BIN which contains the CRT driver for the current CRT is not deleted. Note that SCRATCH BIN will not remove any binaries that reside in ROM.

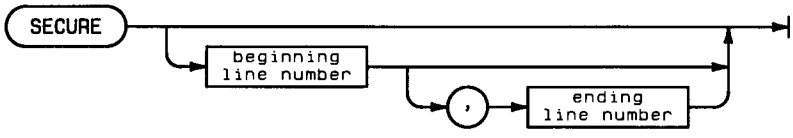
SEC

See the SEND statement.

SECURE

Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF..THEN..	No

This command protects program lines so that they cannot be listed.



Item	Description/Default	Range Restrictions
beginning line number	integer constant; Default = first line in program	—
ending line number	integer constant; Default = beginning line number if specified, or last line in program	—

Example Statements

```
SECURE
SECURE 45
SECURE 1,100
```

Semantics

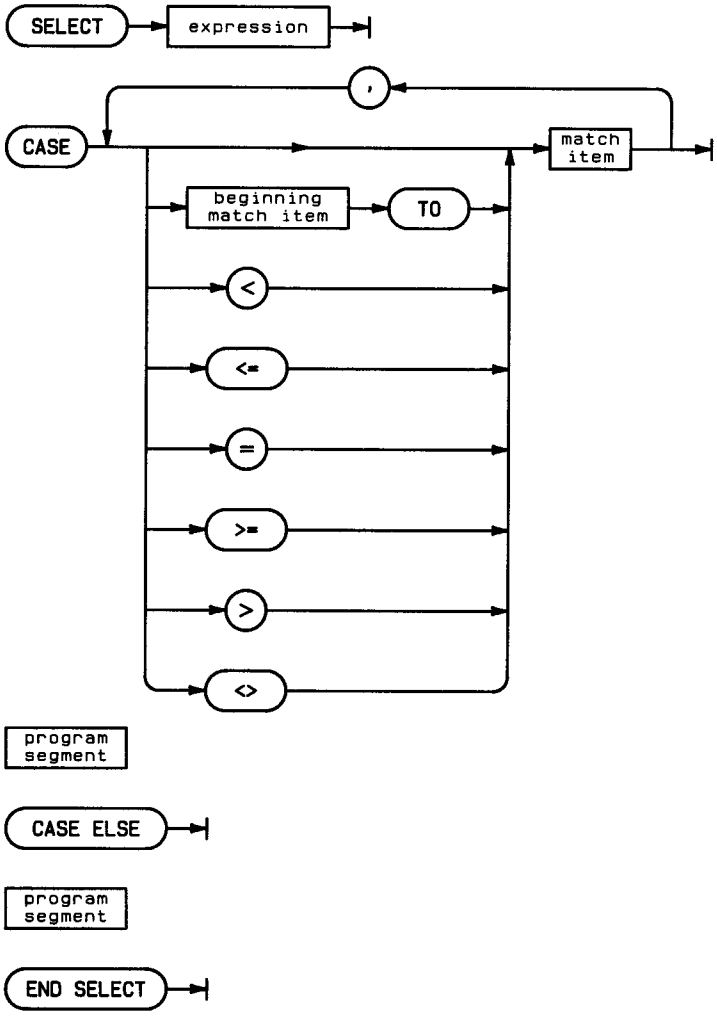
If no lines are specified, the entire program is secured. If one line number is specified, only that line is secured. If two lines are specified, all lines between and including those lines are secured.

Program lines which are secure are listed as an *. Only the line number is listed.

SELECT...CASE

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct provides conditional execution of one of several program segments.



Item	Description/Default	Range Restrictions
expression	a numeric or string expression	—
match item	a numeric or string expression; must be same type as the SELECT expression.	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
650   SELECT Expression
660     CASE <0
670       PRINT "Negative number"
680     CASE ELSE
690       PRINT "Non-negative number"
700   END SELECT

750   SELECT Expression$
760     CASE "A" TO "Z"
770       PRINT "Uppercase alphabetic"
780     CASE ":",";",",",","
790       PRINT "Punctuation"
800   END SELECT
```

Semantics

SELECT...END SELECT is similar to the IF...THEN...ELSE...END IF construct, but allows **several** conditional program segments to be defined; however, **only one segment will be executed** each time the construct is entered. Each segment starts after a CASE or CASE ELSE statement and ends when the next program line is a CASE, CASE ELSE, or END SELECT statement.

The SELECT statement specifies an expression, whose value is compared to the list of values found in each CASE statement. When a match is found, the corresponding program segment is executed. The remaining segments are skipped and execution continues with the first program line following the END SELECT statement.

All CASE expressions must be of the same type, (either string or numeric) and must agree in type with the corresponding SELECT statement expression.

The optional CASE ELSE statement defines a program segment to be executed when the selected expression's value fails to match any CASE statement's list.

Branching into a SELECT...END SELECT construct (via GOTO) results in normal execution until a CASE or CASE ELSE statement is encountered. Execution then branches to the first program line following the END SELECT statement.

Errors encountered in evaluating CASE statements will be reported as having occurred in the corresponding SELECT statement.

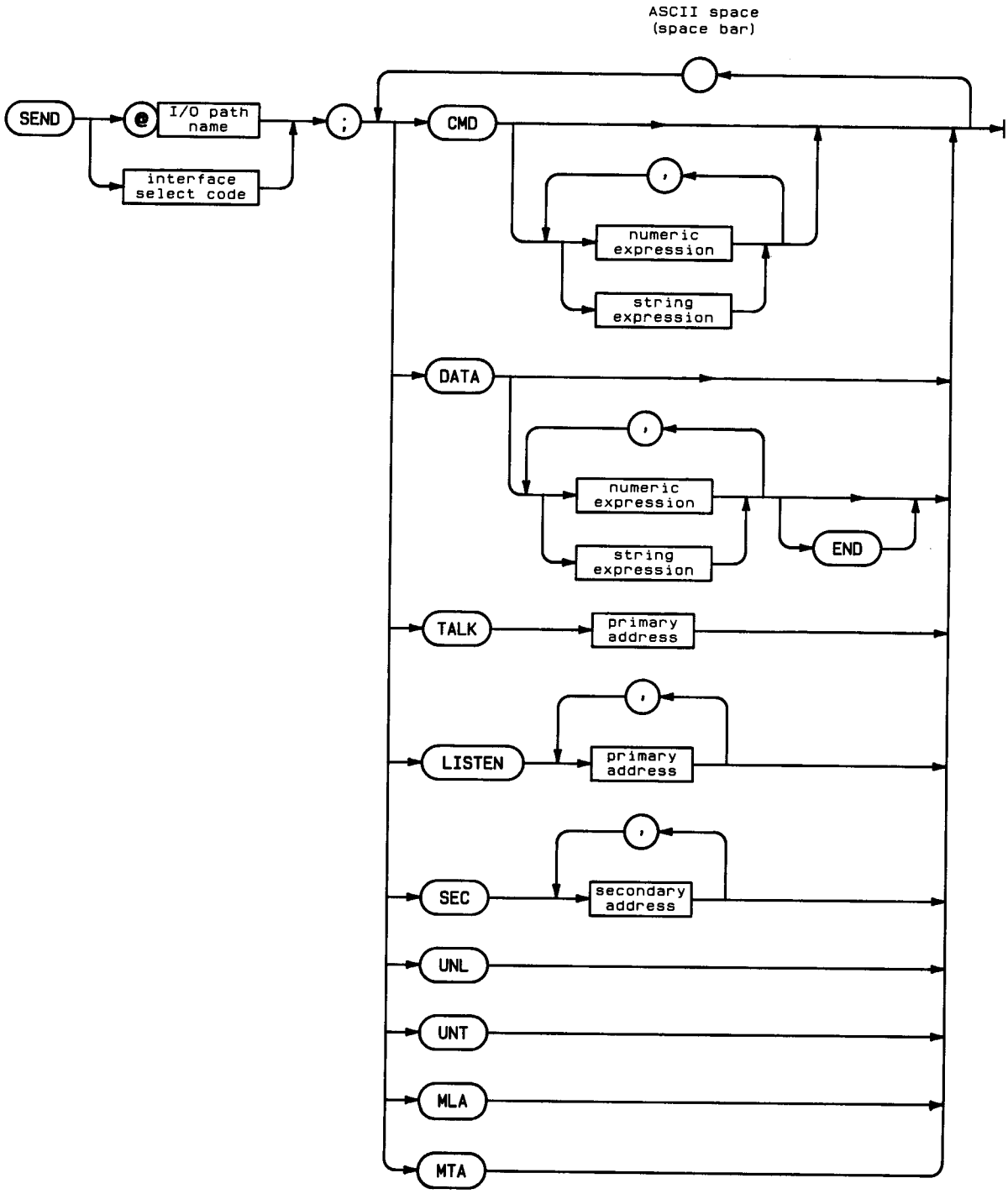
Nesting Constructs Properly

SELECT...END SELECT constructs may be nested, provided inner construct begins and ends before the outer construct can end.

SEND

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends messages to an HP-IB.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
primary address	numeric expression, rounded to an integer	0 thru 31
secondary address	numeric expression, rounded to an integer	0 thru 31

Example Statements

```
SEND 7;UNL MTA LISTEN 1 DATA "HELLO" END
SEND @Hpib;UNL MLA TALK Device CMD 24+128
```

Semantics

CMD

The expressions following a CMD are sent with ATN true. The ASCII characters representing the evaluated string expression are sent to the HP-IB. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. CMD with no items sets ATN true.

DATA

The expressions following DATA are sent with ATN false. The ASCII characters representing the evaluated string expression are sent. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. If END is added to the data list, EOI is set true before sending the last byte. DATA with no items sets ATN false without waiting to be addressed as a talker.

If the computer is active controller, and addressed as a talker, the data is sent immediately. If the computer is not active controller, it waits until it is addressed to talk before sending the data.

TALK

TALK sets ATN true and sends the specified talk address. Only one primary address is allowed for a single talker. An extended talker may be addressed by using SEC secondary address after TALK. A TALK address of 31 is equivalent to UNT (untalk).

UNT

UNT sets ATN true and sends the untalk command. (There is no automatic untalk.) A TALK address of 31 is equivalent to UNT.

LISTEN

LISTEN sets ATN true, sends one or more primary addresses, and addresses those devices to listen. A LISTEN address of 31 is equivalent to UNL (unlisten).

UNL

UNL set ATN true and sends the unlisten command. (There is no automatic unlisten.) A LISTEN address of 31 is equivalent to UNL.

SEC
SEC sets ATN true and sends one or more secondary addresses (commands).

MTA
MTA sets ATN true and sends the interface's talk address. It is equivalent to performing a status sequence on the interface and then using the returned talk address with a SEND..TALK sequence.

MLA
MLA sets ATN true and sends the interface's listen address. It is equivalent to performing a status sequence on the interface and then using the returned listen address with a SEND..LISTEN sequence.

Summary
The computer must be the active controller to execute SEND with CMD, TALK, UNT, LISTEN, UNL, SEC, MTA and MLA.

The computer does not have to be the active controller to send DATA. DATA is sent when the computer is addressed to talk.

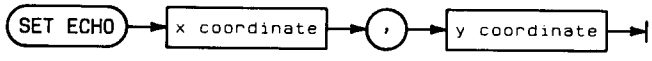
The following table lists the HP-IB message mnemonics, descriptions of the messages, and the secondary keywords required to send the messages. Any numeric values are decimal.

Mnemonic	Description	Secondary Keyword and Value
DAB	Data Byte	DATA 0 thru DATA 255
DCL	Device Clear	CMD 20 or CMD 148
EOI	End or Identify	DATA (data) END (sends EOI with ATN false, which is the END message; EOI with ATN true is the Identify message, sent automatically with the PPOLL function)
GET	Group Execute Trigger	CMD 8 or CMD 136
GTL	Go To Local	CMD 1 or CMD 129
IFC	Interface Clear	Not possible with SEND. An ABORT statement must be used.
LAG	Listen Address Group	LISTEN 0 thru LISTEN 31 or CMD 32 thru CMD 63
LLO	Local Lockout	CMD 17
MLA	My Listen Address	MLA
MTA	My Talk Address	MTA
PPC	Parallel Poll Configure	CMD 5 or CMD 133
PPD	Parallel Poll Disable	PPC (CMD 5 or CMD 133), followed by CMD 112, or CMD 240, or SEC 16.
PPE	Parallel Poll Enable	PPC (CMD 5 or CMD 133), followed by CMD 96 thru CMD 111, or CMD 224 thru CMD 239, or SEC 0 thru SEC 15. SEC 0 allows a mask to be specified by a numeric value.
PPU	Parallel Poll Unconfigure	CMD 21 or CMD 149
PPOLL	Parallel Poll	Not possible with SEND. PPOLL function must be used.
REN	Remote Enable	Not possible with SEND. REMOTE statement must be used.
SDC	Selected Device Clear	CMD 4 or CMD 132
SPD	Serial Poll Disable	CMD 25 or CMD 153
SPE	Serial Poll Enable	CMD 24 or CMD 152
TAD	Talk Address	TALK 0 thru TALK 31, or CMD 64 thru CMD 95, or CMD 192 thru CMD 223.
TCT	Take Control	CMD 9 or CMD 137
UNL	Unlisten	UNL, or LISTEN 31, or CMD 63, or CMD 191.
UNT	Untalk	UNT, or TALK 31, or CMD 95, or CMD 223.

SET ECHO

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sets an echo to the specified location on the current PLOTTER IS device.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression in current units	—
y coordinate	numeric expression in current units	—

Example Statements

```
SET ECHO Xin,Yin
SET ECHO 1000,10000
```

Semantics

If the current PLOTTER IS device is a CRT, a 9-by-9-dot cross-hair is displayed at the specified coordinates if they are within the hard clip limits; the soft clip limits are ignored. No echo is displayed if the coordinates are outside the hard clip limits.

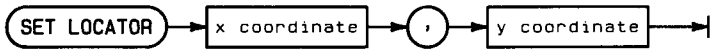
If the current PLOTTER IS device is an HPGL plotter, the pen is raised and moved to the specified coordinates if they are within the current clip limits. If the pen is inside the clip limits and the new echo position is not, it moves towards the new echo position but stops at the clip boundary. If the pen is outside the clip limits and the new echo position is outside the clip limits, the pen moves along the nearest clip boundary.

SET ECHO is frequently used with the READ LOCATOR statement.

SET LOCATOR

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF.. THEN...	Yes

This statement specifies a new position for the locator of the current graphics input device.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression specifying the x coordinate of the locator's new position in current units	range of REAL
y coordinate	numeric expression specifying the y coordinate of the locator's new position in current units	range of REAL

Example Statements

```
SET LOCATOR 12,95
SET LOCATOR X_cor,Y_cor
```

Semantics

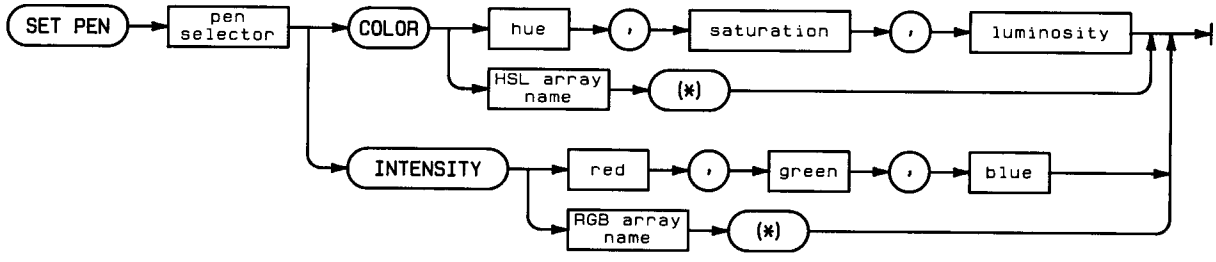
If any of the coordinates are outside the device's limits, they are truncated to the nearest boundary.

In order to change the X and Y coordinates of the locator, the graphics input device must have a programmable locator position, (e.g. graphics input is from the keyboard and other relative locators).

SET PEN

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement defines the color for one or more entries in the color map.



Item	Description/Default	Range Restrictions
pen selector	numeric expression, rounded to an integer	0 thru 32 767
hue	numeric expression	0 thru 1
saturation	numeric expression	0 thru 1
luminosity	numeric expression	0 thru 1
HSL array name	name of a two-dimensional, three-column REAL array	any valid name
red	numeric expression	0 thru 1
green	numeric expression	0 thru 1
blue	numeric expression	0 thru 1
RGB array name	name of a two-dimensional, three-column REAL array	any valid name

Example Statements

```
SET PEN 3 COLOR Hue,Saturation,Luminosity
SET PEN Pen_number INTENSITY Color_map_array(*)
SET PEN 0 INTENSITY 4/15,4/15,4/15
```

Semantics

This statement defines the color for one or more entries in the color map. Either the HSL (hue/saturation/luminosity) color model or the RGB (red/green/blue) color model may be used. This statement is ignored for non-color mapped devices and color mapped devices in non-color map mode.

For both SET PEN COLOR and SET PEN INTENSITY, the pen selector specifies the first color map entry to be defined. If individual RGB or HSL values are given, that entry in the color map is the only one defined. If an array is specified, the color map is redefined, starting at the specified pen, and continuing until either the highest-numbered entry in the color map is redefined or the source array is exhausted.

Specifying color with the SET PEN and AREA PEN statements (resulting in non-dithered color) results in a much more accurate representation of the desired color than specifying the color with an AREA statement. Compare the five color plates shown in this entry with the corresponding plates in the AREA statement.

Note

The following color plates do not exactly represent what your eye would see on the CRT. The reason for this is that photographic film cannot capture all the colors a CRT can produce, and the printing process cannot reproduce all the colors that film can capture.

The five following color plates are multiple exposures.

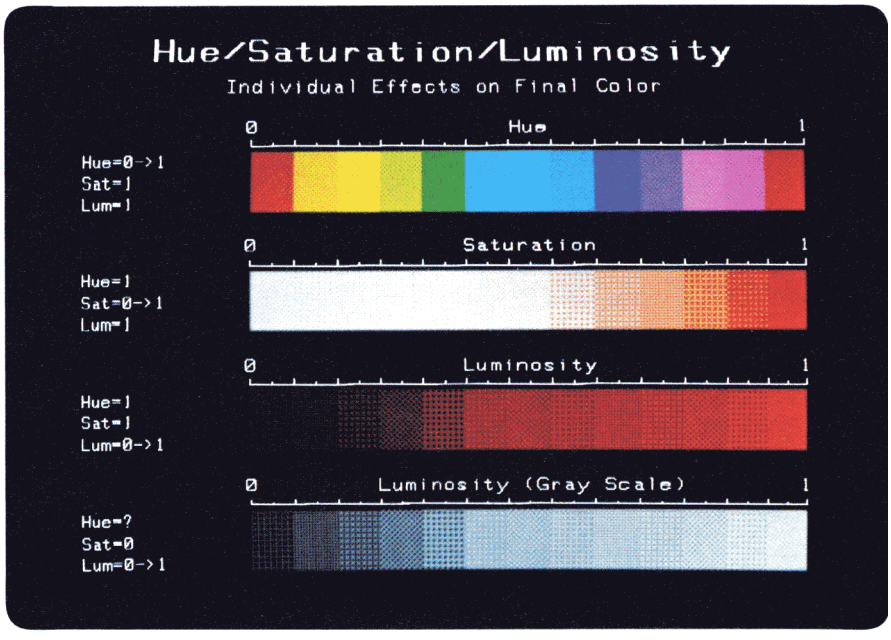
SET PEN COLOR

The hue value specifies the color. The hue ranges from zero to one, in a circular manner, with a value of zero resulting in the same hue as a value of one. The hue, as it goes from zero to one, proceeds through red, orange, yellow, green, cyan, blue, magenta, and back to red.

The saturation value, classically defined, is the inverse of the amount of white added to a hue. What this means is that saturation specifies the amount of hue to be mixed with white. As saturation goes from zero to one, there is 0% to 100% of pure hue added to white. Thus, a saturation of zero results in a gray, dependent only upon the luminosity; hue makes no difference.

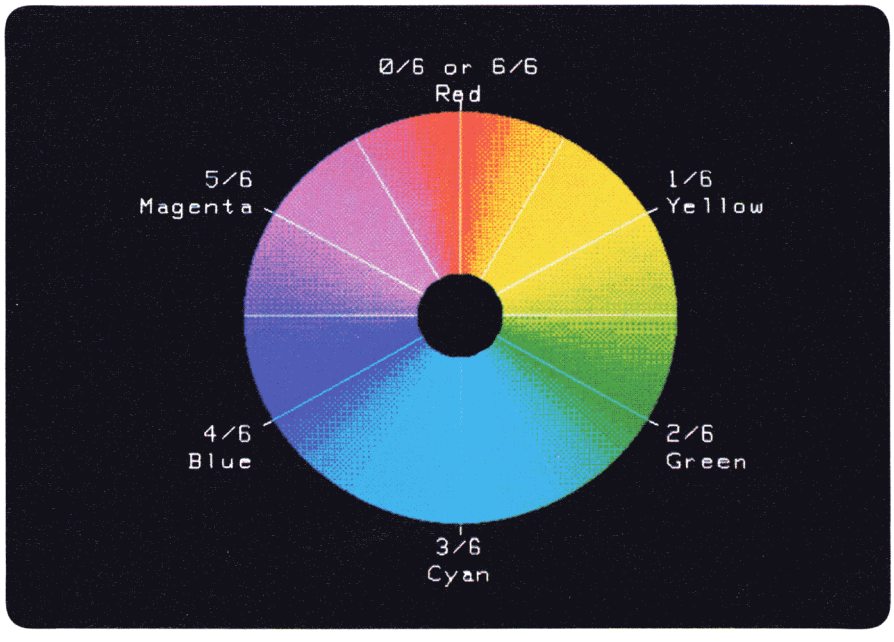
The luminosity value specifies the brightness per unit area of the color. A luminosity of zero results in black, regardless of hue or saturation; if there is no color, it makes no difference which color it is that is not there.

The following color plate shows the changes brought about by varying one of HSL parameters at a time. The bottom bar shows that when saturation (the amount of color) is zero, hue makes no difference, and varying luminosity results in a gray scale.



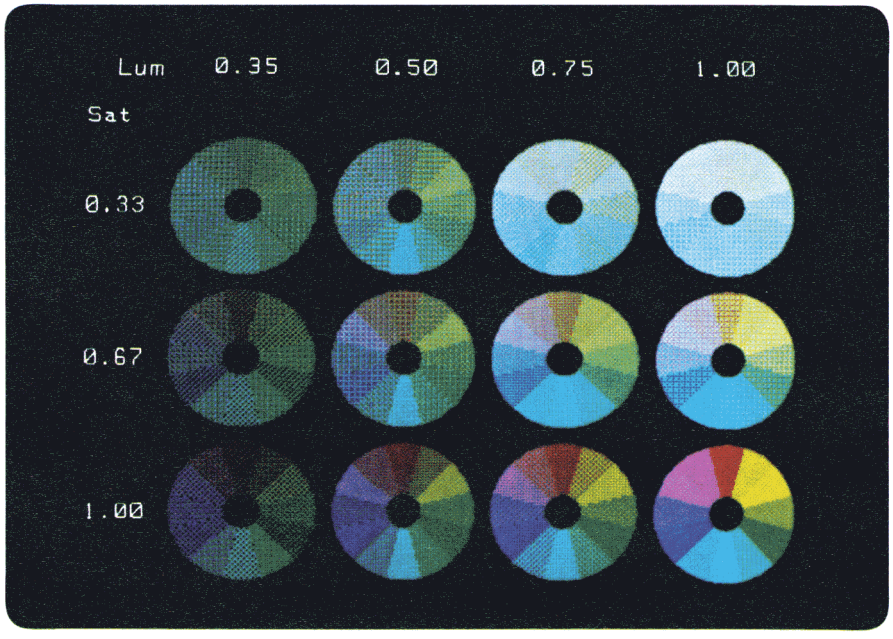
The following color wheel represents the fully saturated, fully luminous colors selected as the hue value goes from 0 through 1. Any value between zero and one, inclusive, can be chosen to select color, but the resolution (the amount the value can change before the color on the screen changes) depends on the value of hue, as well as the other two parameters.

HSL Color Wheel



The next color plate shows the effect that varying saturation and luminosity have on hue. Each of the small color wheels is a miniature version of the large one above, except it has fewer colors.

Effects of Saturation and Luminosity on Color

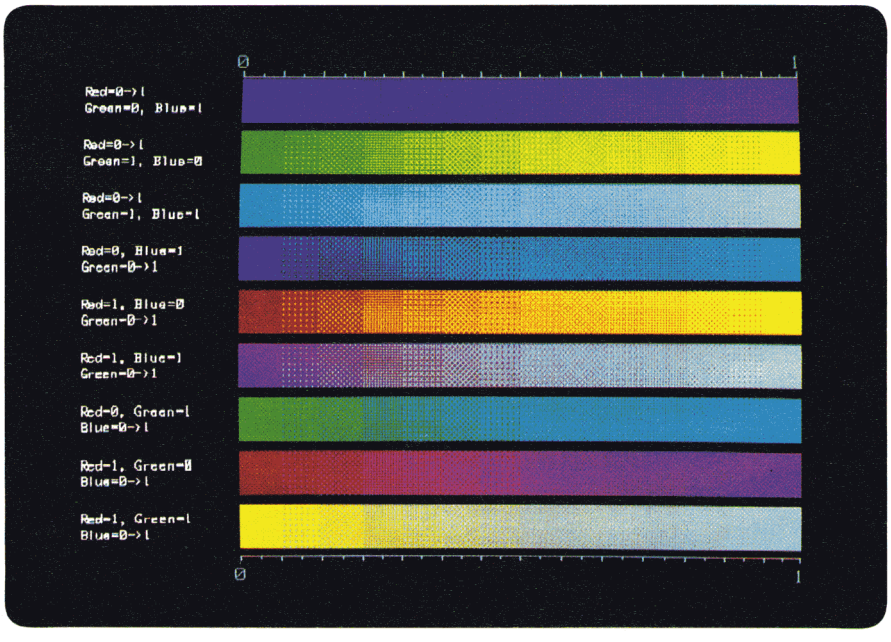


SET PEN INTENSITY

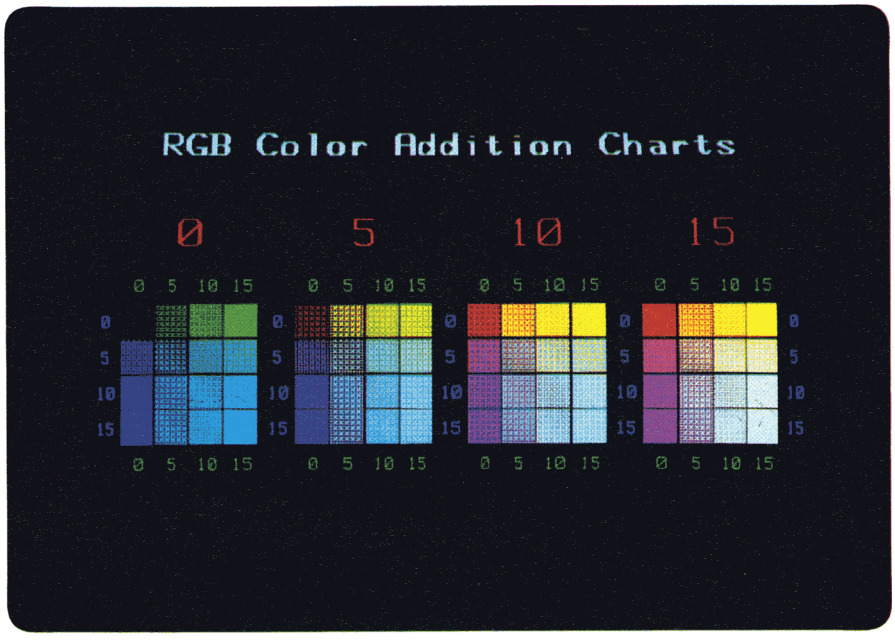
The red, green, and blue values specify the intensities of the red, green, and blue colors displayed on the screen.

The following color plate demonstrate the effect of varying the intensity of one color component while the other two remain the constant.

RGB Addition: One Color at a Time



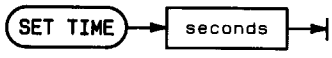
The next plate shows combinations of red, green and blue. The values are represented in fifteenths: 0 fifteenths, 5 fifteenths, 10 fifteenths, and 15 fifteenths – every fifth value. Fifteenths are the units. Thus, zero fifteenths through fifteen fifteenths made a total of sixteen levels. The values for each color component are represented in that color.



SET TIME

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement resets the time-of-day given by the real-time clock.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest hundredth	0 thru 86 399.99

Example Statements

```
SET TIME 0
SET TIME Hours*3600+Minutes*60
```

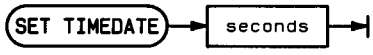
Semantics

SET TIME changes only the time within the current day, not the date. The new clock setting is equivalent to (TIMEDATE DIV 86 400) × 86 400 plus the specified setting.

SET TIMEDATE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement resets the absolute seconds (time and day) given by the real-time clock.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest hundredth	2.086 629 12 E+11 thru 2.143 252 223 999 9 E+11

Example Statements

```
SET TIMEDATE TIMEDATE+86400
SET TIMEDATE Strange_number
```

Semantics

The volatile clock is set to 2.086 629 12 E+11 (midnight March 1, 1900) at power-on. If there is a battery-backed (non-volatile) clock, then the volatile clock is synchronized with it at power-up. If the computer is on an SRM system (and has no battery-backed clock), then the volatile clock is synchronized with the SRM clock when the SRM and DCOMM binaries are loaded. The clock values represent Julian time, expressed in seconds.

SGN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns 1 if the argument is positive, 0 if it equals zero, and −1 if it is negative.



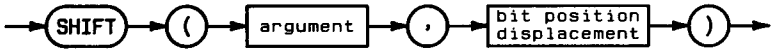
Example Statements

```
Root=SGN(X)*SQR(ABS(X))
Z=2*PI*SGN(Y)
```

SHIFT

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified, without wraparound.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	—
bit position displacement	numeric expression, rounded to an integer	−32 768 thru +32 767	−15 thru +15

Example Statements

```
New_word=SHIFT(Old_word,-2)
Mask=SHIFT(1,Position)
```

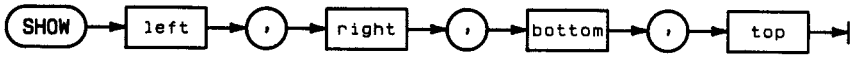
Semantics

If the bit position displacement is positive, the shift is towards the least-significant bit. If the bit position displacement is negative, the shift is towards the most-significant bit. Bits shifted out are lost. Bits shifted in are zeros. The SHIFT operation is performed without changing the value of any variable in the argument.

SHOW

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to define an isotropic current unit-of-measure for graphics operations.



Item	Description/Default	Range Restrictions
left	numeric expression	—
right	numeric expression	≠ left
bottom	numeric expression	—
top	numeric expression	≠ bottom

Example Statements

```
SHOW -5,5,0,100
SHOW Left,Right,Bottom,Top
```

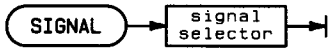
Semantics

SHOW defines the values which must be displayed within the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. SHOW creates isotropic units (units the same in X and Y). The direction of an axis may be reversed by specifying the left greater than the right or the bottom greater than the top. (Also see WINDOW.)

SIGNAL

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement generates a software interrupt.



Item	Description/Default	Range Restrictions
signal selector	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
SIGNAL 3
SIGNAL Bailout
```

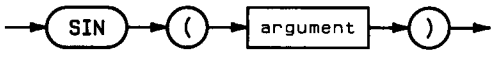
Semantics

If an ON SIGNAL statement for the specified signal selector exists, and all the other conditions for an event-initiated branch are fulfilled, the branch defined in the ON SIGNAL statement is taken. If no ON SIGNAL exists for the specified signal selector, the SIGNAL statement causes no action.

SIN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the sine of the angle represented by the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 1.708 312 781 2 E+10 deg. or 2.981 568 26 E+8 rad.

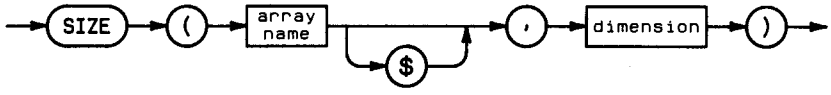
Example Statements

```
Sine=SIN(Angle)
PRINT "Sine of";Theta;"=";SIN(Theta)
```

SIZE

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the size (number of elements) of a dimension of an array. This INTEGER value represents the difference between the upper bound and the lower bound, plus 1.



Item	Description/Default	Range Restrictions
array name	name of an array	any valid name
dimension	numeric expression, rounded to an integer	1 thru 6; ≤ the RANK of the array

Example Statements

```
Upperbound(2)=BASE(A,2)+SIZE(A,2)-1
Number_words=SIZE(Words$,1)
```

SORT

See the MAT SORT statement.

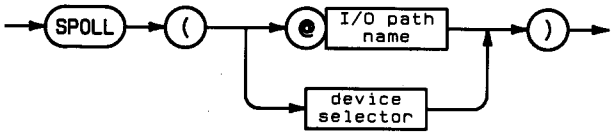
SPANISH

See the LEXICAL ORDER IS statement.

SPOLL

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer containing the serial poll response from the addressed device.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	must include a primary address (see Glossary)

Example Statements

```
Stat=SPOLL(707)
IF SPOLL(@Device) THEN Respond
```

Semantics

The computer must be the active controller to execute this function. Multiple listeners are not allowed. One secondary address may be specified to get status from an extended talker. Refer to the documentation provided with the device being polled for information concerning the device's status byte.

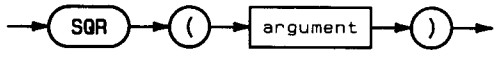
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT
Not Active Controller	Error			

SQR

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the square root of the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	⩾ 0

Example Statements

```
Amps=SQR(Watts/Ohms)
PRINT "Square root of";X;"=";SQR(X)
```

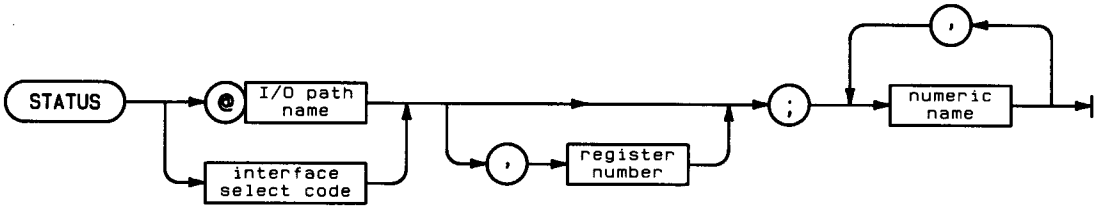
STANDARD

See the LEXICAL ORDER IS statement.

STATUS

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement returns the contents of interface or I/O path name status registers. (If using STATUS with SRM, also refer to the "SRM" section of this manual.)



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, mass storage file, or buffer	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	1 thru 40
register number	numeric expression, rounded to an integer; Default = 0	interface dependent
numeric name	name of a numeric variable	any valid name

Example Statements

```
STATUS 1;Xpos,Ypos
STATUS @File,5;Record
```

Semantics

The value of the beginning register number is copied into the first variable, the next register value into the second variable, and so on. The information is read until the variables in the list are exhausted; there is no wraparound to the first register. An attempt to read a nonexistent register generates an error.

The register meanings depend on the specified interface or on the resource to which the I/O path name is currently assigned. Register 0 of I/O path names can be interrogated with STATUS even if the I/O path name is currently invalid (i.e., unassigned to a resource). Note that the Status registers of an I/O path are different from the Status registers of an interface. All Status and Control registers are summarized in the "Interface Registers" section at the back of the book.

STEP

See the FOR...NEXT construct.

STOP

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement terminates execution of the program.



Semantics

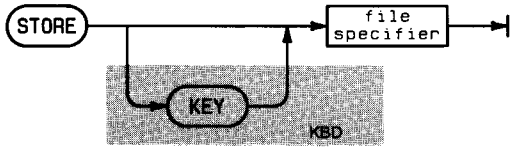
Once a program is stopped, it cannot be resumed by CONTINUE. RUN must be executed to restart the program. PAUSE should be used if you intend to continue execution of the program.

A program can have multiple STOP statements. Encountering an END statement or pressing the STOP (SHIFT STOP on the HP 46020A keyboards) key has the same effect as executing STOP. After a STOP, variables that existed in the main context are available from the keyboard.

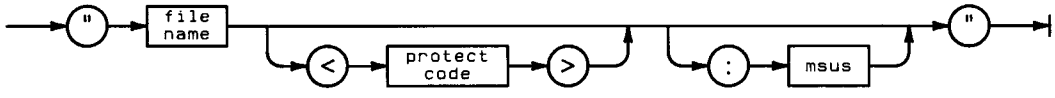
STORE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a file and stores the program or typing-aid key definitions into it. (If using STORE with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two non-blank characters are significant	">" not allowed
msus	literal	(see MASS STORAGE IS)

Example Statements

```
STORE Filename$&Msus$
STORE KEY "KEYS"
```

Semantics

In all STORE statements, an error will occur if the storage media cannot be found, the media or directory is full, or the file specified already exists. Also, if a protect code is specified, it will be applied to the new file. To update a file which already exists, see RE-STORE.

STORE

The STORE statement creates a PROG file and stores an internal form of the program into that file.

STORE KEY

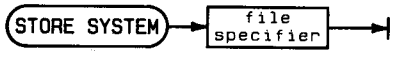
STORE KEY creates a file of type BDAT, and stores the current typing-aid key definitions (**not** ON KEY definitions) into it. These definitions may subsequently be reloaded into the computer with the LOAD KEY statement.

For each defined key, an integer and a string are sent to the file. The integer is the key number, and the string is the key definition. The string consists of a four-byte length followed by the key definition, padded to an even length. The data is written with FORMAT OFF (see the OUTPUT statement). Keys with no definition are not written to the file.

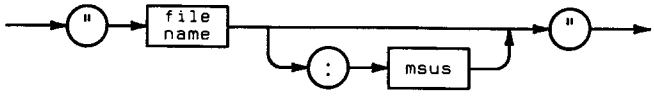
STORE SYSTEM

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

The command stores the entire BASIC operating system currently in memory including any BINs that are loaded. (If using STORE SYSTEM with SRM, also refer to the "SRM" section of this manual.)



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
msus	literal	(see MASS STORAGE IS)

Example Statements

```
STORE SYSTEM "SYSTEM_B1:INTERNAL"
STORE SYSTEM "BACKUP1"
```

Semantics

If the file name already exists, an error occurs.

The BASIC system and any BINs in memory are stored in the file. If the file name begins with SYSTEM_, the Boot ROM can load it at power up or SYSBOOT.

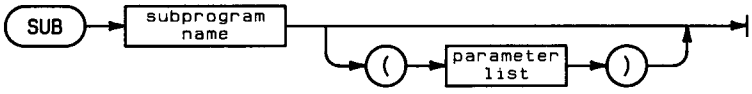
Note that if you did a SCRATCH BIN to remove the CRT driver you did not need, and then stored the system, when you reboot, the CRT driver for the other display is not available. If the CRT needs the other driver, you cannot use the display. Execute a LOAD BIN command to load the needed driver.

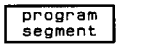
STORE SYSTEM cannot be used with ROM BASIC systems.

SUB

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

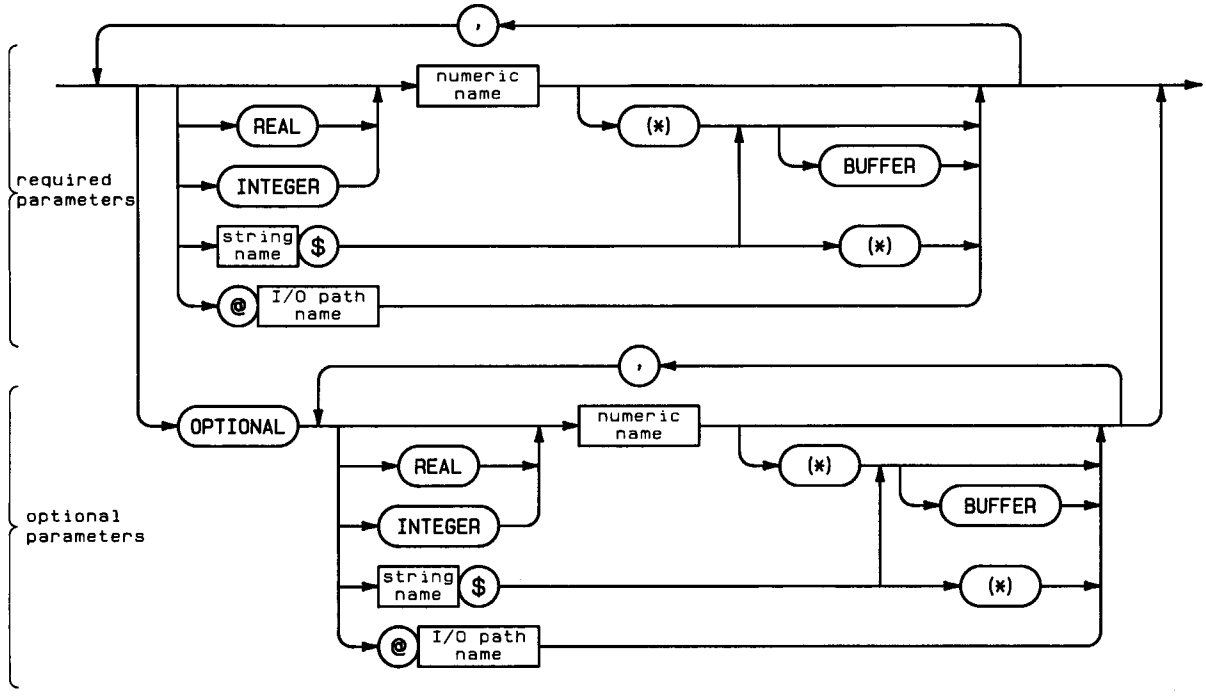
This is the first statement in a SUB subprogram and can specify the subprogram's formal parameters.







parameter list:



Item	Description/Default	Range Restrictions
subprogram name	name of the SUB subprogram	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Statements

```
SUB Parse(String$)
SUB Transform(@Printer,INTEGER Array(*),OPTIONAL Text$)
```

Semantics

SUB subprograms must appear after the main program. The first line of the subprogram must be a SUB statement. The last line must be a SUBEND statement. Comments after the SUBEND are considered to be part of the subprogram.

Parameters to the left of the keyword OPTIONAL are required and must be supplied whenever the subprogram is invoked (see CALL). Parameters to the right of OPTIONAL are optional, and only need to be supplied if they are needed for a specific operation. Optional parameters are associated from left to right with any remaining pass parameters until the pass parameter list is exhausted. An error is generated if the subprogram tries to use an optional parameter which did not have a value passed to it. The function NPAR can be used to determine the number of parameters supplied by the CALL statement invoking the subprogram.

Variables in a subprogram's formal parameter list may not be duplicated in COM or other declaratory statements within the subprogram. A subprogram may not contain any SUB statements, or DEF FN statements. Subprograms can be called recursively and may contain local variables. A unique labeled COM must be used if the local variables are to preserve their values between invocations of the subprogram.

SUBEXIT may be used to leave the subprogram at some point other than the SUBEND. Multiple SUBEXITs are allowed, and SUBEXIT may appear in an IF...THEN statement. SUBEND is prohibited in IF...THEN statements, and may only occur once in a subprogram.

If you want to use a formal parameter as a BUFFER, it must be declared as a BUFFER in both the formal parameter list and the calling context.

SUBEND

See the SUB statement.

SUBEXIT

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

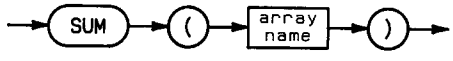
This statement may be used to return from a SUB subprogram at some point other than the SUBEND statement. It allows multiple exits from a subprogram.



SUM

Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the sum of all elements of a numeric array. The value returned is of the same type as the array.



Item	Description/Default	Range Restrictions
array name	name of a numeric array	any valid name

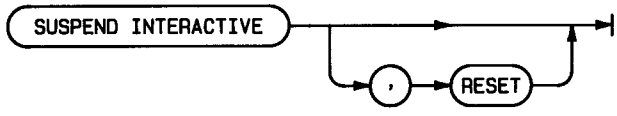
Example Statements

```
Array_sum=SUM(A)
Sum_squares=SUM(Squares)
```

SUSPEND INTERACTIVE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables the [EXECUTE], [ENTER], [RETURN], [PAUSE], [STOP], [CLR I/O], [BREAK], and (optionally) [RESET] key functions during a running program.



Example Statements

```
SUSPEND INTERACTIVE,RESET
IF NOT Kbd_flag THEN SUSPEND INTERACTIVE
```

Semantics

Execution of a PAUSE statement, a TRACE PAUSE statement, or a fatal execution error temporarily restores the suspended key functions. CONTINUE after a PAUSE will again disable the keys.

SUSPEND INTERACTIVE is cancelled by RESUME INTERACTIVE, STOP, END, RUN, SCRATCH, GET, LOAD, or [RESET]. Although LOAD cancels SUSPEND INTERACTIVE, LOADSUB does not. SUSPEND INTERACTIVE has no effect unless a program is running.

Note

Suspending the [RESET] key will prevent you from stopping a program before it ends.

[EXECUTE] [ENTER] and [RETURN] can still be used to respond to an ENTER or INPUT statement, but cannot be used for live keyboard execution.

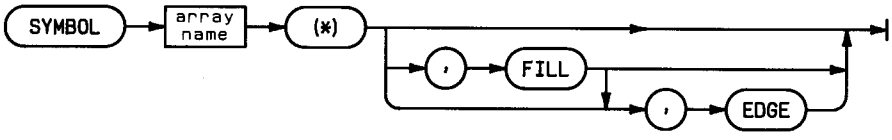
SWEDISH

See the LEXICAL ORDER IS statement.

SYMBOL

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows labelling with user-defined symbols.



Item	Description/Default	Range Restrictions
array name	name of a two-dimensional, two-column or three-column REAL array	any valid name

Example Statements

```
SYMBOL My_char(*)
SYMBOL Logo(*),FILL,EDGE
```

Semantics

The user-defined symbol is created with moves and draws defined in a *symbol coordinate system*. The symbol coordinate system is a rectangular area nine units wide and fifteen units high, that is, a character cell. A symbol can extend outside the limits of the 9×15 symbol coordinate system rectangle. A symbol defined in the symbol coordinate system is affected by the label transformations CSIZE, LDIR, and LORG. The symbol is drawn using the current pen and line type, and it will be clipped at the current clip boundary.

When defining a symbol in the symbol coordinate system, coordinates may be outside the 9×15 character cell; thus, characters can be made which are several character cells wide and several character cells high. For this reason, the current pen position is not updated to the next character's reference point, but it remains at the last X,Y coordinate specified in the array. A move is made to the first point regardless of the value in the third column of that row of the array.

The symbol may have polygons defined in its data, and the polygons may be filled and/or edged. The fill color and pen number/line type used are those defined at the time the polygon is closed.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the ranges 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the SYMBOL statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the SYMBOL statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the SYMBOL statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

When using an SYMBOL statement, the following table of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth. See the list below.

Column 1	Column 2	Operation Selector	Meaning
X	Y	−2	Pen up before moving
X	Y	−1	Pen down before moving
X	Y	0	Pen up after moving (Same as +2)
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	ignored	3	Select pen
line type	repeat value	4	Select line type
color	ignored	5	Color value
ignored	ignored	6	Start polygon mode with FILL
ignored	ignored	7	End polygon mode
ignored	ignored	8	End of data for array
ignored	ignored	9	NOP (no operation)
ignored	ignored	10	Start polygon mode with EDGE
ignored	ignored	11	Start polygon mode with FILL and EDGE
ignored	ignored	12	Draw a FRAME
pen number	ignored	13	Area pen value
red value	green value	14	⎫ Color
blue value	ignored	15	⎭ Value
ignored	ignored	>15	Ignored

Moving and Drawing
If the operation selector is less than or equal to two, it is interpreted in exactly the same manner as the third parameter in a non-array SYMBOL statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens
An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types
An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color
Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation Selector 14 is used in conjunction with Operation Selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessariy immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation Selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word, that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities(which range from 0 thru 1) for red, green, and blue in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME
Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the SYMBOL statement, so one probably would not have more than one operation selector 12 in an array to SYMBOL, since the last FRAME will overwrite all the previous ones.

Premature Termination
Operation selector 8 causes the SYMBOL statement to be terminated. The SYMBOL statement will successfully terminate if the actual end of the array has been reached, so use of operation selector 8 is optional.

Ignoring Selected Rows in the Array
Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

SYSBOOT

Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF..THEN..	No

This command returns control to the BOOT ROM to restart the system configuration and selection process.



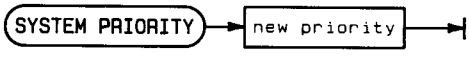
Example Statements

```
SYSBOOT
```

SYSTEM PRIORITY

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sets system priority to a specified value.



Item	Description/Default	Range Restrictions
new priority	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
SYSTEM PRIORITY Old
IF Critical_code THEN SYSTEM PRIORITY 15
```

Semantics

Zero is the lowest user-specifiable priority and 15 is the highest. The END, ERROR, and TIMEOUT events have an effective priority higher than the highest user-specifiable priority. If no SYSTEM PRIORITY has been executed, minimum system priority is 0.

This statement establishes the minimum for system priority. Once the minimum system priority is raised with this statement, any events of equal or lower priority will be logged but not serviced. In order to allow service of lower-priority events, minimum system priority must be explicitly lowered.

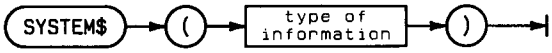
If SYSTEM PRIORITY is used to change the minimum system priority in a subprogram context, the former value is restored when the context is exited.

Error 427 results if SYSTEM PRIORITY is executed in a service routine for an ON ERROR GOSUB or ON ERROR CALL statement.

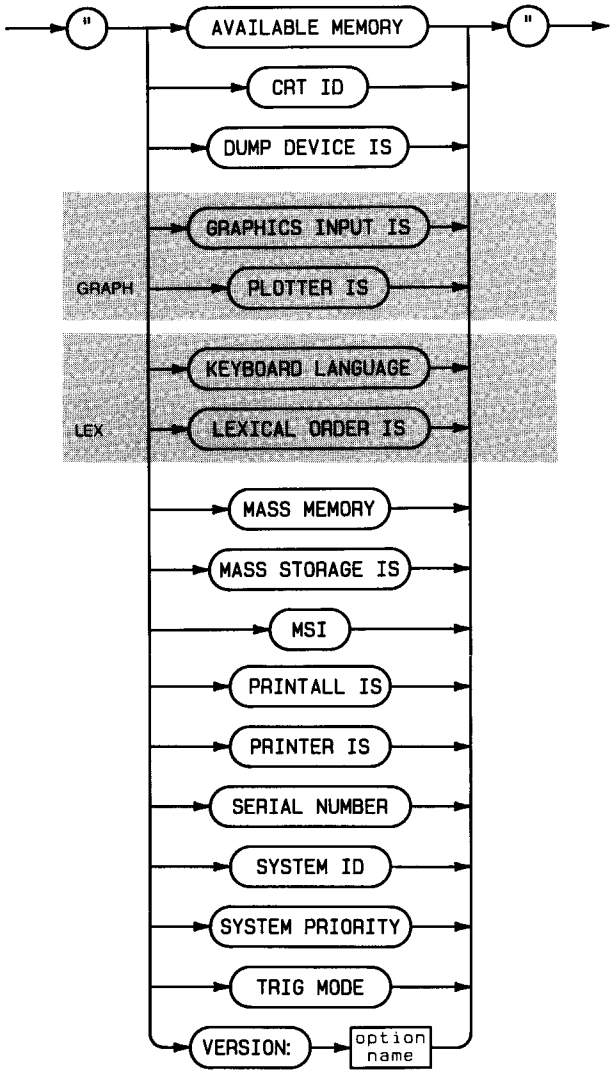
SYSTEM$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a string containing system status and configuration information. (If using SYSTEM$ with SRM, also refer to the "SRM" section of this manual.)



literal form of type of information:



Item	Description/Default	Range Restrictions
type of information	string expression	–
option name	literal specifying an option or BIN	BASIC, KBD, CLOCK, IO, MS, GRAPH, GRAPHX, LEX, MAT, PDEV, XREF, SRM, TRANS, ERR, DISC, CS80, HP9885, BUBBLE, EPROM, HPIB, FHPIB, GPIO, DCOMM, SERIAL, BCD, CRTA, CRTB

Example Statements

```
IF TRIM$(SYSTEM$("SYSTEM ID"))="9836A" THEN CALL New_machine
System_prior=VAL(SYSTEM$("SYSTEM PRIORITY"))
```

Semantics

The topic specifier is used to specify what system configuration information the system will return. The following table lists the valid topic specifiers and the type of information returned by the system for each of the topic specifiers.

Topic Specifier	Information Returned
AVAILABLE MEMORY	Bytes of available memory
CRT ID	6: 80HCGB B = Bit Map Display Space = Not Bit Map Display G = Graphics Available Space = No Graphics C = Color Available Space = No Color H = CRT Highlights Available Space = No Highlights CRT Width in Characters Distinguishes this format from Series 500 BASIC responses.
DUMP DEVICE IS	A string containing numerals which specify the device selector for the currently assigned DUMP DEVICE IS device.
GRAPHICS INPUT IS	A string containing numerals which specify the device selector for the currently assigned GRAPHICS INPUT IS device. Zero is returned if no device is currently selected. (Requires GRAPH)
KBD LINE	A string containing the current contents of the keyboard input line(s). Note that this operation does not change the contents of the line(s).
KEYBOARD LANGUAGE	ASCII, BELGIAN, CANADIAN ENGLISH, CANADIAN FRENCH, DANISH, DUTCH, FINNISH, FRENCH, GERMAN, ITALIAN, KATAKANA, LATIN, NORWEGIAN, SPANISH, SWEDISH, SWISS FRENCH, SWISS GERMAN, SWISS FRENCH*, SWISS GERMAN*, or UNITED KINGDOM (Requires LEX)
LEXICAL ORDER IS	ASCII, GERMAN, FRENCH, SPANISH, SWEDISH or USER DEFINED (Requires LEX)
MASS MEMORY	X000YZ0000000000 X = Number of internal disc drives Y = Number of initialized EPROM cards Z = Number of bubble memory cards If Y or Z exceed 9, an asterisk appears.
MASS STORAGE IS MSI	The mass storage unit specifier of the current MASS STORAGE IS device, as it appears in a CAT heading.
PLOTTER IS	A string containing numerals which specify the device selector of the current PLOTTER IS device or the path name of the current PLOTTER IS file. (Requires GRAPH)
PRINTALL IS	A string containing numerals which specify the device selector of the current PRINTALL IS device.
PRINTER IS	A string containing numerals which specify the device selector of the current PRINTER IS device or the path name of the current PRINTER IS file.
SERIAL NUMBER	If an ID PROM is present, this string contains bytes 4-14 of that PROM. Otherwise, a null string is returned.

Topic Specifier	Information Returned
SYSTEM ID	S300:20 on Series 300 computers with an MC68020 processor; or S300:10 on Series 300 computers with an MC68010 processor; or bytes 15 thru 21 of the ID PROM in a Series 200 computer (if present); or 9816, 9826A, or 9836A padded with trailing spaces to make a seven character string.
SYSTEM PRIORITY	A string containing numerals which specify the current system priority.
TRIG MODE	DEG or RAD
VERSION: option name	A string containing numerals which specify the revision number displayed at power up and displayed after LOAD BIN or LIST BIN.

TAB

See the PRINT and DISP statements.

TABXY

See the PRINT statement.

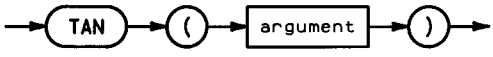
TALK

See the SEND statement.

TAN

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the tangent of the angle represented by the argument. Error 31 occurs when trying to compute the TAN of an odd multiple of 90 degrees.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 8.541 563 906 E+9 deg. or 1.490 784 13 E+8 rad.

Example Statements

```
Tangent=TAN(Angle)
PRINT "Tangent of";Z;"=";TAN(Z)
```

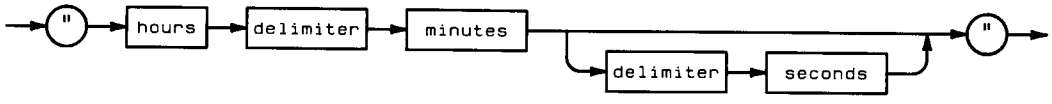
TIME

Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts the formatted time of day (HH:MM:SS), into the number of seconds past midnight. (Also see the OFF TIME, ON TIME, and SET TIME statements.)



literal form of time of day:



Item	Description/Default	Range Restrictions
time of day	string expression representing the time in 24-hour format	(see drawing)
hours	literal	0 thru 23
minutes	literal	0 thru 59
seconds	literal; default = 0	0 thru 59.99
delimiter	literal; single character	(see text)

Example Statements

```
Seconds=TIME(T$)
SET TIME TIME("8:37:30")
ON TIME TIME("12:12") GOSUB Food_food
```

Semantics

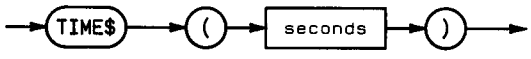
TIME returns a REAL whole number, in the range 0 thru 86 399, equivalent to the number of seconds past midnight.

While any number of non-numeric characters may be used as a delimiter, a single colon is recommended. Leading blanks and non-numeric characters are ignored.

TIME$

Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts the number of seconds past midnight into a string representing the time of day (HH:MM:SS).



Item	Description/Default	Range Restrictions	Recommended Range
seconds	numeric expression, truncated to the nearest second; seconds past midnight	−4.623 683 256 E+13 thru +4.653 426 335 039 9 E+13	0 thru 86 399

Example Statements

```
DISP "The time is: ";TIME$(TIMEDATE)
PRINT TIME$(45296)
```

Semantics

TIME$ takes time (in seconds) and returns the time of day in the form HH:MM:SS, where HH represents hours, MM represents minutes, and SS represents seconds. A modulo 86 400 is performed on the parameter before it is formatted as a time of day.

TIMEDATE

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the current value of the real-time clock. (Also see the SET TIMEDATE statement.)



Example Statements

```
Elapsed=TIMEDATE-T0
DISP TIMEDATE MOD 86400
```

Semantics

The value returned by TIMEDATE represents the sum of the last time setting and the number of seconds that have elapsed since that setting was made. The clock value set at power-on is 2.086 629 12 E+11, which represents midnight March 1, 1900. The time value accumulates from that setting unless it is changed by SET TIME or SET TIMEDATE.

The resolution of the TIMEDATE function is .01 seconds. If the clock is properly set, TIMEDATE MOD 86400 gives the number of seconds since midnight.

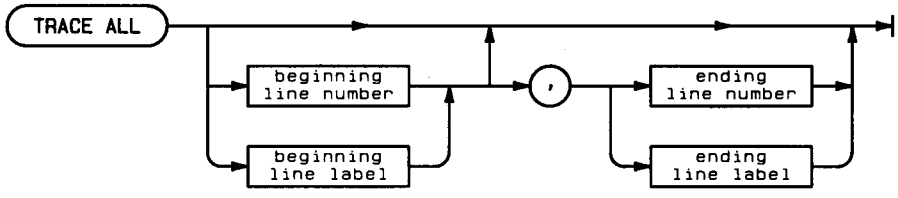
TIMEOUT

See the OFF TIMEOUT and ON TIMEOUT statements.

TRACE ALL

Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows tracing program flow and variable assignments during program execution.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
TRACE ALL Sort
TRACE ALL 1500,2450
```

Semantics

The entire program, or any part delimited by beginning and (if needed) ending line numbers or labels, may be traced.

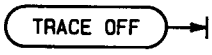
The ending line is not included in the trace output. The trace output stops immediately before the ending line is executed. When the program is traced, execution of the lines within the tracing range causes the line number and any variable which receives a new value to be output to the system message line of the CRT. Any type of variable (string, numeric or array) can be displayed. For simple string and numeric variables, the name and the new value are displayed. For arrays, a message is displayed stating that the array has a new value rather than outputting the entire array contents.

TRACE ALL output can also be printed on the PRINTALL printer, if PRINTALL is ON. TRACE ALL is disabled by TRACE OFF. The line numbers specified for TRACE ALL are not affected by REN.

TRACE OFF

Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

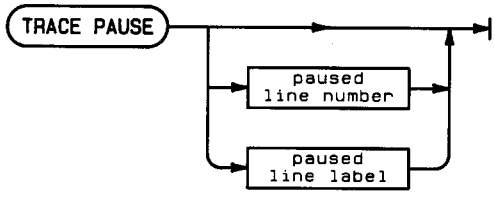
This statement turns off all tracing activity.



TRACE PAUSE

Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement causes program execution to pause before executing the specified line, and displays the next line to be executed on the CRT.



Item	Description/Default	Range Restrictions
paused line number	integer constant identifying a program line; Default = next program line	1 thru 32 766
paused line label	name of a program line	any valid name

Example Statements

```
TRACE PAUSE
TRACE PAUSE Loop_end
```

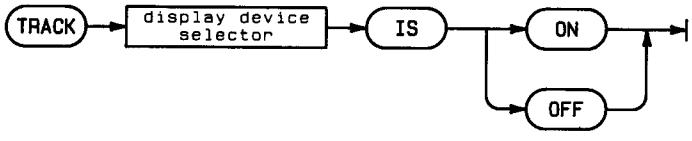
Semantics

Not specifying a line for TRACE PAUSE results in the pause occurring before the next line is executed. Only one TRACE PAUSE can be active at a time. TRACE PAUSE is cancelled by TRACE OFF.

TRACK

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement enables and disables tracking of the current locator position on the current display device.



Item	Description/Default	Range Restrictions
display device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
TRACK 709 IS ON
TRACK Plot IS OFF
```

Semantics

The current locator is defined by a GRAPHICS INPUT IS statement, and the current display device is defined by a PLOTTER IS statement. If TRACK...IS ON is executed, an echo on the current display device tracks the locator position during DIGITIZE statements. On a CRT, the echo is a 9-by-9-dot crosshair. On a plotter, the pen position tracks the locator. When a point is digitized, the echo is left at the location of the digitized point and tracking ceases.

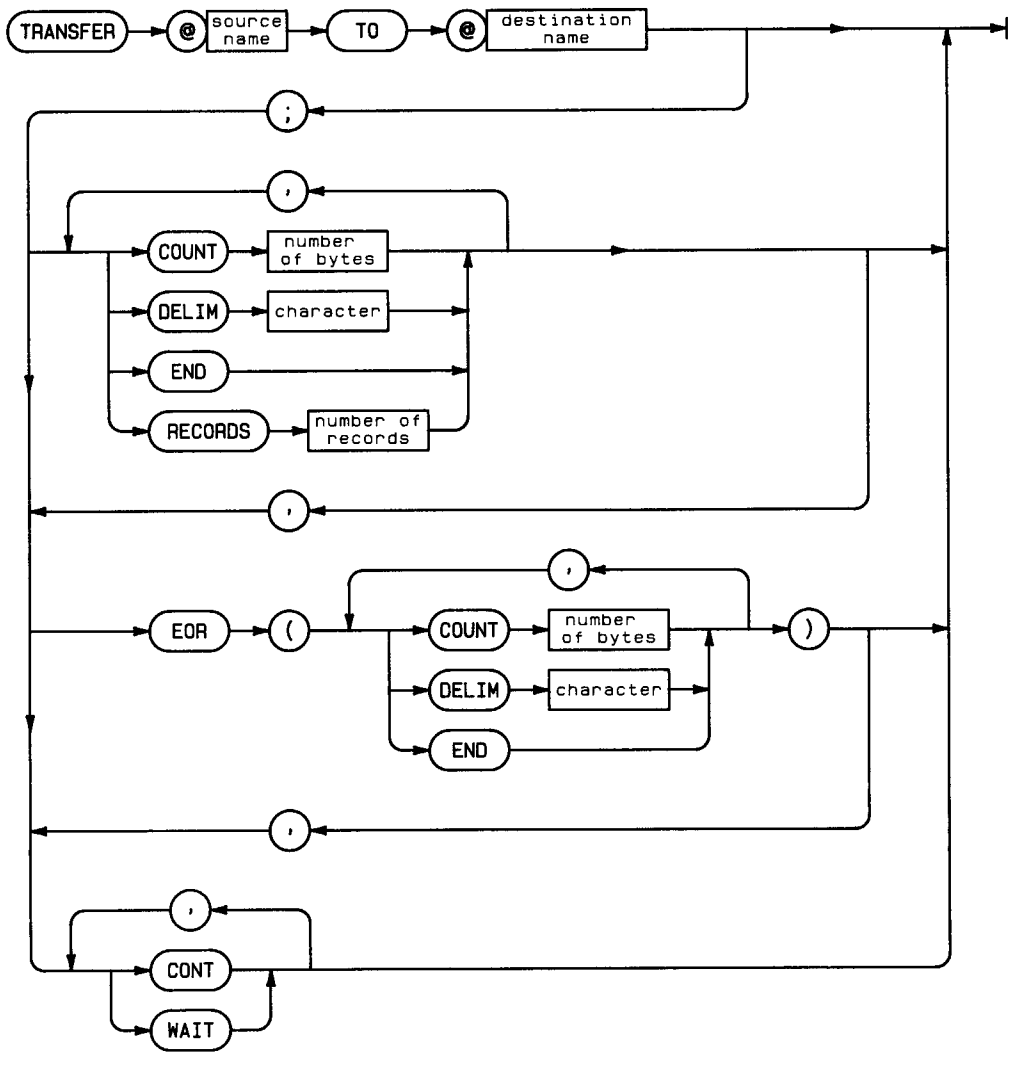
The display device selector must match that used in the most recently executed PLOTTER IS statement, or error 708 results.

Executing TRACK...IS OFF disables tracking.

TRANSFER

Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement initiates unformatted I/O transfers. (If using TRANSFER with SRM, also refer to the "SRM" section of this manual.)



Item	Description/Default	Range Restrictions
source name	I/O path name assigned to a device, a group of devices, a mass storage file, or a buffer	any valid name
destination name	I/O path name assigned to a device, a group of devices, a mass storage file, or a buffer	any valid name
number of bytes	numeric expression, rounded to an integer	1 thru $2^{31}-1$
character	string expression with a length of zero or one	—
number of records	numeric expression, rounded to an integer	1 thru $2^{31}-1$

Example Statements

```
TRANSFER @Device TO @Buff
TRANSFER @Buff TO @File;CONT
TRANSFER @Path TO @Destination;COUNT 256
TRANSFER @Source TO @Buffer;DELIM "/"
TRANSFER @Path TO @Buffer;RECORDS 12,EOR(COUNT 8)
```

Semantics

The TRANSFER statement allows unformatted data transfers between the computer and devices (mass storage drives are considered devices for this operation). Whenever possible, a TRANSFER takes place concurrently with continued program execution. Since no formatting is performed and the TRANSFER statement executes concurrently (overlapped) with regular program execution, the highest possible data transfer rate is achieved.

Before a data transfer can take place, a buffer must be declared. Every TRANSFER will need a buffer as either its source or its destination. An outbound TRANSFER empties the buffer (source) while an inbound TRANSFER fills the buffer (destination). Device to device transfers and buffer to buffer transfers are not allowed.

Two types of buffers are available; named and unnamed. A named buffer is a REAL array, INTEGER array, or a string scalar declared with the keyword BUFFER. See ASSIGN, COM, DIM, INTEGER, and REAL. Unnamed buffers are created in the ASSIGN statement by specifying the keyword BUFFER and the number of bytes to be reserved for the buffer. See ASSIGN.

Every buffer has two pointers associated with it. The fill pointer indicates the next available location in the buffer for data. The empty pointer indicates the next item to be removed from the buffer. This allows an inbound TRANSFER and an outbound TRANSFER to access the same buffer simultaneously.

BDAT is the only file type allowed in a TRANSFER. An end-of-file error will prematurely terminate a TRANSFER, thus triggering an end-of-transfer condition. If an end-of-record condition was satisfied when the end-of-file was reached, the EOR event will also be true.

I/O path names should be used to access the contents of the buffer. This ensures the automatic updating of the fill and empty pointers during a transfer. For named buffers, the contents of the buffer can also be accessed by the buffer's variable name. However, accessing the contents of the buffer by the variable name does not update the fill and empty pointers and is likely to corrupt the data in the buffer.

Transfer Parameters
When no parameters are specified for a TRANSFER, an inbound TRANSFER will fill the buffer with data and then terminate. An outbound transfer will empty the buffer and then terminate. Both inbound and outbound transfers execute in overlapped mode when possible.

The CONT parameter specifies that the TRANSFER is to continue indefinitely. Instead of terminating on buffer full or buffer empty conditions, the TRANSFER will be temporarily suspended until there is space available in the buffer (for inbound transfers) or until there is data available in the buffer (for outbound transfers).

The WAIT parameter specifies that the TRANSFER is to take place serially (non-overlapped). Program execution will not leave the TRANSFER statement until the data transfer is completed.

A TRANSFER can be specified to terminate when a device dependent signal is received (END), after a specified number of bytes has been transferred (COUNT), or after a specific character is detected (DELIM). The DELIM parameter can only be used with inbound transfers.

If `END` is included on a TRANSFER to a file, the end-of-file pointer is updated when the TRANSFER terminates; including `EOR(END)` causes the end-of-file pointer to be updated at the end of each record.

When the RECORD parameter is specified, the end-of-record parameter must also be specified (EOR). The end-of-record condition can be either COUNT, DELIM, END or any combination of conditions.

Overlapped execution of the TRANSFER statement can be deferred until a record has been transferred or until the entire TRANSFER has completed. See WAIT FOR EOR and WAIT FOR EOT.

Supported Devices
The TRANSFER statement supports data transfers to and from the following devices.

HP-IB	(HP 98624)
GPIO	(HP 98622)
Serial	(HP 98626)
Datacomm	(HP 98628)
HP-IL	(HP 98634)

TRANSFER can also be used with BDAT files on any of the mass storage devices supported by BASIC 4.0 except the 9144A and the 9122 formatted for 512-byte sectors (format option 2).

Transfer Method
The transfer method is device dependent and chosen by the computer. The three possible transfer modes are:

INT interrupt mode
FHS fast handshake
DMA direct memory access

The DMA mode will be used whenever possible. If the DMA mode cannot be used (DMA card is not installed, both channels are busy, DELIM is specified, or the interface does not support DMA) then the INT mode will be used. FHS is used with the HP-IB or GPIO interfaces only when DMA cannot be used and the WAIT parameter is specified.

Interactions
When the computer tries to move into the stopped state, it will wait for any transfer to complete. Therefore, operations which would cause a stopped state will make the computer unresponsive (or "hung") if a TRANSFER is in progress. Operations in this category include a programmed GET, modifying a paused program, and STOP. Also, the computer will not exit a context until any TRANSFER in that context is complete. This will cause the program to wait at a SUBEXIT, SUBEND, or RETURN <expression> statement while a TRANSFER is in progress.

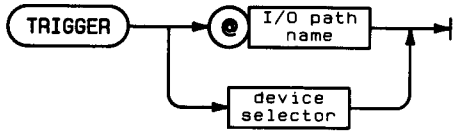
To terminate a transfer before it has finished (and free the computer), execute an ABORT IO (or, as a last resort, press (RESET)).

See also: ASSIGN, WAIT FOR EOT, WAIT FOR EOR, ABORTIO, RESET and the "Transfer" chapter of the *BASIC Interfacing Techniques* manual.

TRIGGER

Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends a trigger message to a selected device, or all devices addressed to listen, on the HP-IB.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
TRIGGER 712
TRIGGER @Hpib
```

Semantics

The computer must be the active controller to execute this statement.

If only the interface select code is specified, all devices on that interface which are addressed to listen are triggered. If a primary address is given, the bus is reconfigured and only the addressed device is triggered.

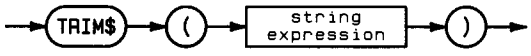
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN UNL LAG GET
Not Active Controller	Error			

TRIM$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the string stripped of all leading and trailing ASCII spaces.



Example Statements

```
Unjustify$=TRIM$("    center    ")
Clean$=TRIM$(Input$)
```

Semantics

Only leading and trailing ASCII spaces are removed. Embedded spaces are not effected.

TRN

See the MAT statement.

UNL

See the SEND statement.

UNT

See the SEND statement.

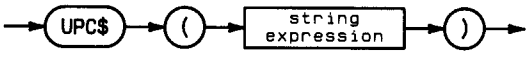
UNTIL

See the REPEAT...UNTIL construct.

UPC$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function replaces any lowercase characters with their corresponding uppercase characters.



Example Statements

```
Capital$=UPC$("lower")
IF UPC$(Name$)="TOM" THEN Equal_tom
```

Semantics

The corresponding characters for the Roman Extension alphabetic characters are determined by the current lexical order. When the lexical order is a user-defined table, the correspondence is determined by the STANDARD lexical order.

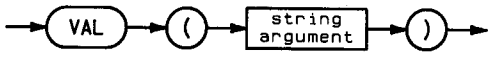
USING

See the DISP, ENTER, LABEL, OUTPUT, and PRINT statements.

VAL

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a string expression into a numeric value.



Item	Description/Default	Range Restrictions
string argument	string expression	numerals, decimal point, sign and exponent notation

Example Statements

```
Day=VAL(Date$)
IF VAL(Response$)<0 THEN Negative
```

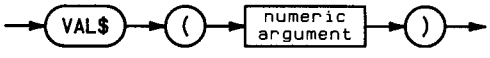
Semantics

The first non-blank character in the string must be a digit, a plus or minus sign, or a decimal point. The remaining characters may be digits, a decimal point, or an E, and must form a valid numeric constant. If an E is present, characters to the left of it must form a valid mantissa, and characters to the right must form a valid exponent. The string expression is evaluated when a non-numeric character is encountered or the characters are exhausted.

VAL$

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a string representation of the value of the argument. The returned string is in the default print format, except that the first character is not a blank for positive numbers. No trailing blanks are generated.



Item	Description/Default	Range Restrictions
numeric argument	numeric expression	—

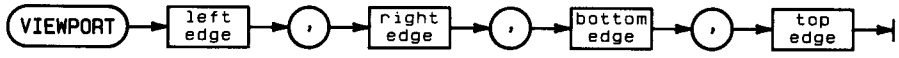
Example Statements

```
PRINT Esc$;VAL$(Cursor-1)
Special$=Text$&VAL$(Number)
```

VIEWPORT

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines an area onto which WINDOW and SHOW statements are mapped. It also sets the soft clip limits to the boundaries it defines.



Item	Description/Default	Range Restrictions
left edge	numeric expression	—
right edge	numeric expression	>left edge
bottom edge	numeric expression	—
top edge	numeric expression	>bottom edge

Example Statements

```
VIEWPORT 0,35,50,80
VIEWPORT Left,Right,Bottom,Top
```

Semantics

The parameters for VIEWPORT are in Graphic Display Units (GDUs). Graphic Display Units are 1/100 of the shorter axis of a plotting device. The units are isotropic (the same length in X and Y). The soft clip limits are set to the area specified, and the units defined by the last WINDOW or SHOW are mapped into the area.

For the plotter specifier "INTERNAL" (the CRT), the shorter axis is Y. The longer axis is X, which is 100 × RATIO GDUs long. For the plotter specifier "HPGL" (which deals with devices other than the CRT), the RATIO function may be used to determine the ratio of the length of the X axis to the length of the Y axis. If the ratio is greater than one, the Y axis is 100 GDUs long, and the length of the X axis is 100 × RATIO. If the ratio is less than one, then the length of the X axis is 100 GDUs and the length of the Y axis is 100 × RATIO.

A value of less than zero for the left edge or bottom is treated as zero. A value greater than the hard clip limit is treated as the hard clip limit for the right edge and the top. The left edge must be less than the right edge, and the bottom must be less than the top, or error 704 results.

WAIT

Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement will cause the computer to wait approximately the number of seconds specified before executing the next statement. Numbers less than 0.001 do not generate a WAIT interval.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest thousandth	less than 2 147 483.648

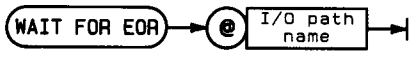
Example Statements

```
WAIT 3
WAIT Old_time/2
```

WAIT FOR EOR

Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement waits until an end-of-record event occurs in the TRANSFER on the specified I/O path.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name

Example Statements

```
WAIT FOR EOR @File
WAIT FOR EOR @Device
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the WAIT FOR EOR statement is executed.

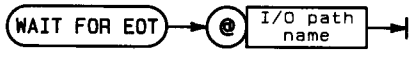
The WAIT FOR EOR statement prevents further program execution until an end-of-record event occurs in the TRANSFER whose I/O path name was specified. This allows ON EOR events, which might otherwise be missed, to be serviced. If the system priority prevents the servicing of an ON EOR event, the event will be logged.

The I/O path specified must be involved in an active TRANSFER for the statement to have any effect.

WAIT FOR EOT

Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement waits until the TRANSFER on the specified I/O path is completed.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name

Example Statements

```
WAIT FOR EOT @File
WAIT FOR EOT @Device
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the WAIT FOR EOT statement is executed.

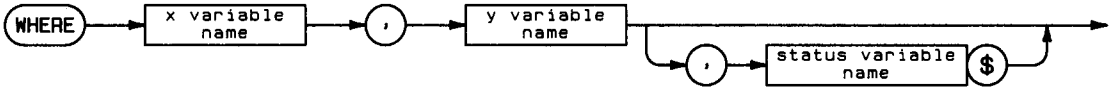
The WAIT FOR EOT statement prevents further program execution until the specified TRANSFER is completed. This allows ON EOT events, which might otherwise be missed, to be serviced. If the system priority prevents the servicing of an ON EOT event, the event will be logged.

The I/O path specified must be involved in an active TRANSFER for the statement to have any effect.

WHERE

Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement returns the current logical position of the pen and, optionally, pen status information.



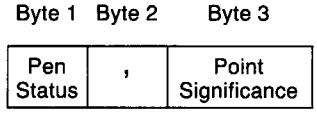
Item	Description/Default	Range Restrictions
x variable name	name of a numeric variable	any valid name
y variable name	name of a numeric variable	any valid name
status variable name	name of a string variable whose dimensioned length is at least 3	any valid name

Example Statements

```
WHERE X,Y
WHERE X_position,Y_position,Status$
```

Semantics

The characters in the status string may be interpreted as follows:

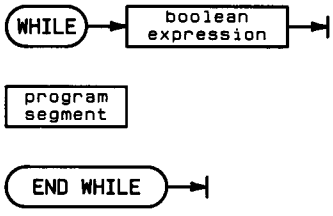


Byte	Value	Meaning
1	"0"	Pen is up
	"1"	Pen is down
2	comma	(delimiter)
3	"0"	Current position is outside hard clip limits.
	"1"	Current position is inside hard clip limits but outside viewport boundary.
	"2"	Current position is inside viewport boundary and hard clip limits.

WHILE

Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is executed as long as the boolean expression in the WHILE statement evaluates to true (evaluates to a non-zero value).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if zero.	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
840   WHILE Value<Min OR Value>Max
850     BEEP
860     INPUT "Out of range; RE-ENTER",Value
870   END WHILE


1220  WHILE P<=LEN(A$)
1230    IF NUM(A$[P])<32 THEN
1240      A$[P]=A$[P+1]  ! Remove control codes
1250    ELSE
1260      P=P+1          ! Go to next character
1270    END IF
1280  END WHILE
```

Semantics

The WHILE...END WHILE construct allows program execution dependent on the outcome of a relational test performed at the **start** of the loop. If the condition is true, the program segment between the WHILE and END WHILE statements is executed and a branch is made back to the WHILE statement. The program segment will be repeated until the test is false. When the relational test is false, the program segment is skipped and execution continues with the first program line after the END WHILE statement.

Branching into a WHILE...END WHILE construct (via a GOTO) results in normal execution up to the END WHILE statement, a branch back to the WHILE statement, and then execution as if the construct had been entered normally.

Nesting Constructs Properly

WHILE...END WHILE constructs may be nested within other constructs, provided the inner construct begins and ends before the outer construct can end.

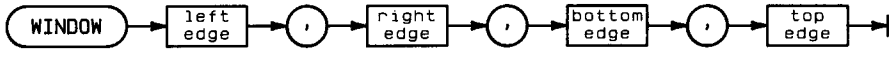
WIDTH

See the PRINTALL IS and PRINTER IS statements.

WINDOW

Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to define the current-unit-of-measure for graphics operations.



Item	Description/Default	Range Restrictions
left edge	numeric expression	—
right edge	numeric expression	≠ left edge
bottom edge	numeric expression	—
top edge	numeric expression	≠ bottom edge

Example Statements

```
WINDOW -5,5,0,100
WINDOW Left,Right,Bottom,Top
```

Semantics

WINDOW defines the values represented at the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. WINDOW may be used to create non-isotropic (not equal in X and Y) units. The direction of an axis may be reversed by specifying the left edge greater than the right edge, or the bottom edge greater than the top edge. (Also see SHOW.)

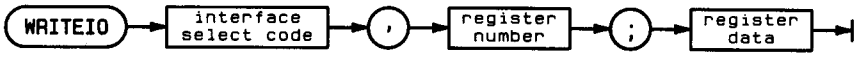
WORD

See the ASSIGN statement.

WRITEIO

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement writes an integer representation of the register-data to the specified hardware register on the specified interface. The actual action resulting from this operation depends on the interface and register selected.



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	numeric expression, rounded to an integer	1 thru 31	—
register number	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	interface dependent
register data	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	−32 768 thru +32 767

Note

Unexpected and possibly undesirable results may occur with select codes outside the given range.

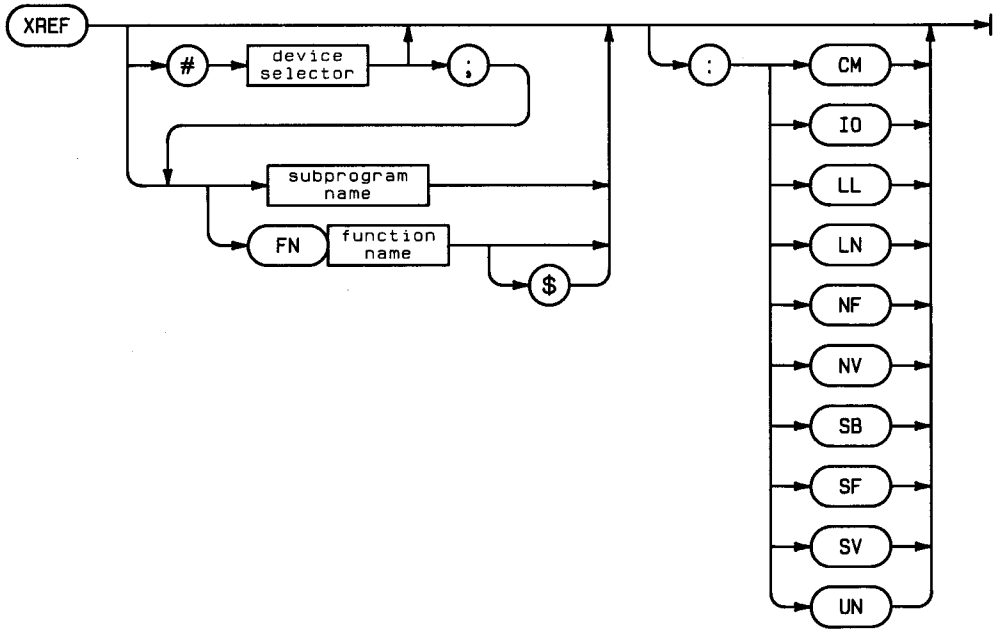
Example Statements

```
WRITEIO 12,0;Set_pctl
WRITEIO Hpib,23;12
```

XREF

Option Required	XREF
Keyboard Executable	Yes
Programmable	No
In an IF...THEN...	No

This command allows you to obtain a cross-reference listing of the identifiers in a program or subprogram.



Item	Description/Default	Range Restrictions
device selector	numeric expression; rounded to an integer Default = PRINTER IS device	(see Glossary)
subprogram name	name of a SUB subprogram or MAIN currently in memory	any valid name
function name	name of a user-defined function currently in memory	any valid name

Example Statements

```
XREF
XREF #705;FNUser$
XREF Print
XREF :NV
```

Semantics

The cross-reference listing is printed one context at a time, in the order that they occur in the program. The main program is listed first, followed by the subprograms.

The cross-reference listing starts with this line:

```
>>>> Cross Reference <<<<
```

Before each subsequent program segment, this line is printed:

```
>>>> Subprogram <<<<
```

followed by the line number of the first line in that context and the name of the context. If the subprogram is a user-defined function, an `FN` will precede the name, and if it is a string function, a `$` will follow its name.

Within each context, identifiers are listed by type. They occur in the following order:

- NV–Numeric Variables
- SV–String Variables
- IO–I/O Path Names
- LL–Line Labels
- LN–Line Numbers
- NF–Numeric Functions
- SF–String Functions
- SB–SUB Subprograms
- CM–Common Block Names
- UN–Unused Entries

If a type is specified in the command, only that type is printed. If there are no identifiers of a particular type in the context being cross-referenced, that heading is not printed.

Within each group (which is composed of a header telling what kind of entity follows, then the list of those entities), names are alphabetized according to the ASCII collating sequence, and line numbers are in numerical order. If a reference is a formal parameter in a SUB or DEF FN statement, declared in a COM, DIM, REAL, or INTEGER statement, or is a line label, the characters `<-DEF` will be printed immediately to the right of the line number containing the defining declaration. Note that variables declared by ALLOCATE are not given this marker. If unlabelled (blank) COM is used, it will have no name associated with it.

At the end of each context, a line is printed that begins with:

```
Unused entries =
```

This is a count of the symbol table entries which have been marked by a prerun as "unused." Unreferenced symbol table locations which have not yet been marked "unused" by the prerun processing will show up in the lists of identifiers with empty reference lists. Note that a subprogram that is not directly recursive will show up in its own cross-reference listing with an empty reference list.

If a subprogram name or MAIN is specified in the XREF command, the above rules are followed, but only the specified subprogram or the MAIN program is cross-referenced. If there are two or more subprograms of the same name in the computer, they will all be cross-referenced.

An XREF can be aborted by pressing (RESET), (CLR I/O) or (Break).

BASIC Language Reference for HP Series 200/300 SRM Workstation

This section lists all BASIC keywords either used exclusively with SRM or whose use with SRM differs from that described in the *BASIC Language Reference* manual.

Most keyword entries in this section describe only differences between the keyword's normal use and its use on SRM. The body of this manual provides full details of their use. SRM-specific keywords (CREATE DIR, LOCK, and UNLOCK) are listed in this section.

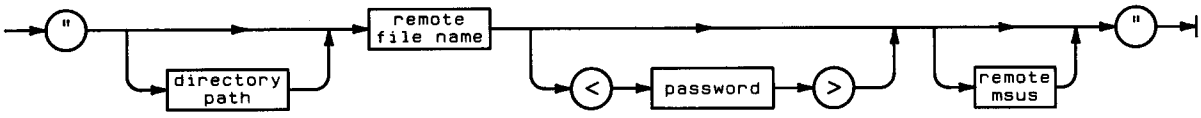
The primary difference in keyword syntax for SRM use is in file specification. Use of supported keywords on SRM requires you to supply a **remote file specifier** rather than the file specifier described for non-SRM uses of BASIC. Some keywords also involve a **directory specifier**, which is unique to SRM. Remote file specifiers and directory specifiers are described at the beginning ofthis section.

In addition, you must be aware of the **access capabilities** required on files and directories involved in the keyword's use. Access capability requirements are summarized in a table included in this section.

Syntax for Remote File and Directory Specification

The following syntax applies to remote file specification for BASIC keyword use on SRM. The semantics discussion applies to all remote file specification unless otherwise noted with a specific keyword's description.

Remote File Specifier



Item	Description/Default	Range Restrictions
directory path	literal	(see diagram)
remote file name	literal	any valid remote file name (see Semantics)
password	literal, first 16 non-blank characters are significant	any valid password (see Semantics)
remote msus	literal	(see diagram)

Semantics

A valid **remote file name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

A valid **password** consists of one to 16 characters, which may include any ASCII character except " > ". Spaces are ignored. Passwords are assigned by the PROTECT keyword.

If no directory path is included, the system assumes the file is in the current working directory (the directory specified in the latest MASS STORAGE IS statement). To specify a file in a directory other than the current working directory, specify the directory path to the desired file. (Refer to the syntax for directory path later in this section.) The directory path may begin at the current working directory or at the root.

The READ access capability for each directory included in the directory path must be public or the password that currently protects the READ capability must be included in the remote file specifier. A maximum of six identifiers can be included in a specifier -- five directories in the path and the target file. If the target file is more than five directories away from the current working directory, move closer by changing the working directory (with MSI).

Examples

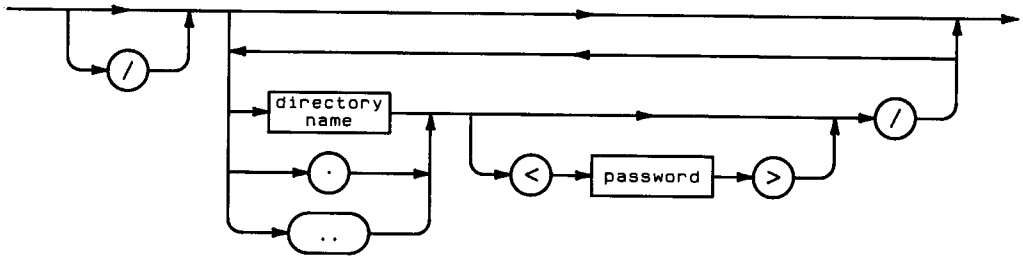
```
"PROJECTS/WRITERS/samples<wr_pass>:REMOTE 21,1;LABELVOL_TWO<master>"
```

illustrates the full remote file specifier syntax. For explanations of the directory path and remote msus portions of this illustration, see the examples with those components.

```
"thisfile"
```

specifies a file that is in the current working directory. This form assumes that the SRM (remote mass storage) has previously been "entered" via some form of the `MSI ":REMOTE"` statement.

Directory Path



Item	Description/Default	Range Restrictions
directory name	literal	any valid directory name (see Semantics)
password	literal, first 16 non-blank characters are significant	any valid password (see Semantics)

Semantics

A valid **directory name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

A valid **password** consists of one to 16 characters, which may include any ASCII character except " > ". Spaces are ignored. Passwords are assigned by the PROTECT.

A leading slash (/) in the directory path specifies that the path begins at the root. If you have not previously established the remote mass storage (using, for example, `MSI ":REMOTE"`), you must include some form of the remote msus with the file specifier. Including the remote msus also specifies that the directory path begins at the root. Remote msus is explained later in this section.

Subsequent slashes delimit individual names in the path.

Using " .. " in place of a directory name specifies the directory immediately superior to the current directory position. (Note that the root's superior directory is the root.) Using " . " in place of a directory name specifies the current directory position. To specify a file or directory subordinate to the current working directory, you do not include the current working directory in the directory path.

Examples

The directory path:

```
/USERS/BO/MANUAL_PLAN<mine*alone>
```

begins at the root.

The directory path:

```
../file1
```

begins at the directory immediately superior to the current working directory.

The directory path:

```
PROJECTS/WRITERS<writers_only>/samples:REMOTE
```

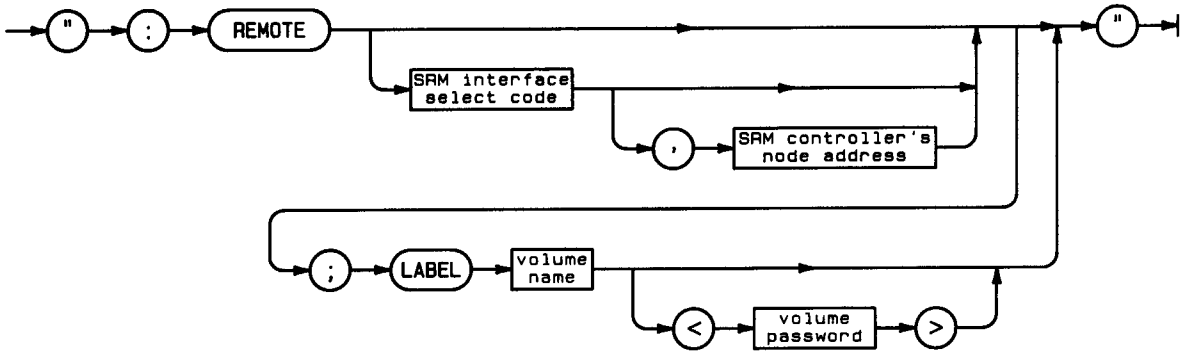
begins at the root.

The directory path:

```
dir_sub/file1
```

begins in the current working directory. In this example, `dir_sub` is immediately subordinate to the current working directory.

Remote msus



Item	Description/Default	Range Restrictions
SRM interface select code	integer constant	8 through 31
SRM controller's node address	integer constant	0 through 63
volume name	literal	any valid volume name (see Semantics)
volume password	literal	any valid password (see Semantics)

Semantics

The **volume name**, which is assigned at the volume's initialization, is used to identify a mass storage volume. Volume names consist of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)).

A valid **volume password** consists of one to 16 characters, which may include any ASCII character except " > ".

The volume password allows complete access to all files on a mass storage volume, and is assigned when the volume is initialized. The volume password supercedes all access restrictions placed on files and directories by the PROTECT statement.

You need supply the **SRM interface select code** only if you wish to specify an SRM interface in your Series 200/300 workstation other than that identified by the default select code. If your workstation boots from the SRM, the default is the select code of the interface through which the boot ROM activates your workstation. If your workstation boots from a source other than SRM, the default select code is the lowest available SRM interface select code in the workstation. (The factory-set default value for the HP 98629A interface's select code is 21.)

The **SRM controller's node address** is necessary only if the node address of the controller is other than the default controller's node address.

To determine the defaults for your workstation use the following command sequence:

```
MSI ":REMOTE"    EXECUTE or Return
CAT    EXECUTE or Return
```

The header of the resulting catalog listing shows the default values for your workstation's SRM interface select code and SRM controller's node address, and the name of the default SRM system volume.

If you include the controller's node address, you must also include the SRM interface select code.

The LABEL secondary keyword identifies a volume, and is used mainly when more than one shared volume is on the SRM system. You need supply the volume label only if you are identifying a volume other than the default SRM system volume (in an SRM system having more than one shared disc) or if your application requires that you specify the volume password.

The Generic Remote msus

The generic msus syntax (not indicated in the syntax diagram above) bypasses the need for all information required by the remote msus syntax except the workstation's SRM interface select code. An example of this msus syntax is:

```
:,21
```

Examples

The remote msus:

```
:REMOTE
```

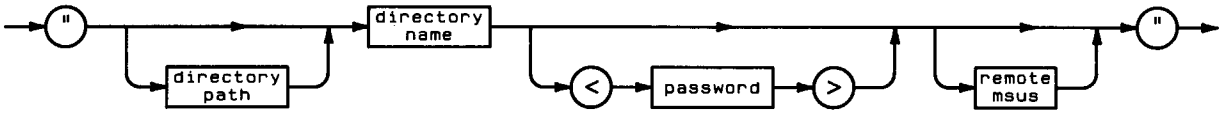
specifies the default SRM system volume.

The remote msus:

```
:REMOTE 21,1;LABEL VOL_TWO<secondpass>
```

specifies an SRM system volume. The LABEL syntax allows inclusion of the volume password in the remote msus. Note that, because the controller's node address is not the default and must be specified, the SRM interface select code must also be specified, even if that select code is the default.

Directory Specifier



Item	Description/Default	Range Restrictions
directory path	literal	(see diagram)
directory name	literal	any valid directory name (see Semantics)
remote msus	literal	(see diagram)

Semantics

A valid **directory name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

If no directory path is included, the current working directory (the directory specified in the latest MASS STORAGE IS statement) is assumed for the keyword's use. To specify a directory other than the current working directory, specify the directory path to the desired directory. (Refer to the syntax for directory path.) The directory path may begin at the current working directory or at the root.

The READ access capability for each directory included in the directory path must be public or the password that currently protects the READ capability must be included in the remote file specifier. A maximum of six directories may be included in the directory specifier. If the target directory is more than five directories away from the current working directory, move closer by changing the working directory (with MSI).

Examples

```
"/"
```

specifies the root. This form assumes that the SRM (remote mass storage) has previously been "entered" via some form of the `MSI ":REMOTE"` statement. (See directory path description.)

```
"../../../"
```

specifies the directory three levels superior to the current working directory. (See directory path description.)

```
".<MGR_pass>"
```

specifies the current working directory, with a password granting an access capability different from that currently in effect.

Access Capability Requirements

Because SRM allows password protection of files and directories, either certain access capabilities must be public or you must supply the password protecting those capabilities when you specify the file or directory in the keyword syntax. For more information on password protection and access capabilities, refer to the section on "Shared Access to Remote Directories and Files" earlier in this chapter and the PROTECT keyword entry in this reference.

The following chart lists BASIC keywords discussed in this section, indicating for each:

- whether the keyword is used with remote files, directories, or can be used with either;
- the access capabilities required on the directories superior to the specified directory or file;
- the access capabilities required on the specified directory or file itself.

Access requirements do not apply to the following keywords:

CHECKREAD
CONTROL
INITIALIZE
ON TIMEOUT
RESET
SCRATCH A
STATUS
UNLOCK
SYSTEM$

Note

For all keywords listed in the table, the READ capability must be public on all directories in the path to the target remote file or directory. Otherwise, you must supply the password protecting the READ capability on any such directory.

The entries in the following table indicate the access capabilities needed for use of the designated keyword. That is, the access capability listed must either be public (not protected with a password) or you must supply the password protecting the capability in the file or directory specifier included with the keyword.

For example, in an OUTPUT statement, if the WRITE capability on the file to which the data is to be written is not public, you must supply the password entitling you to write data to that file. (You would include the password as part of the remote file specifier in the statement assigning the I/O path name for the file to which the data is directed.) If the READ capability on the directory containing the remote file specified in the OUTPUT statement is not public, you must supply the appropriate password with the directory name in the directory path to the remote file.

Access Capabilities Required for Keyword Use

Keyword	Applies to	Access Capabilities Required	
		Directory/ File	Superior Directory
ASSIGN	file	at least 1	READ
CAT	either	READ	READ
COPY			
source	file	READ	READ
destination	file	–	READ & WRITE
CREATE ASCII	file	–	READ & WRITE
CREATE BDAT	file	–	READ & WRITE
CREATE DIR	directory	–	READ & WRITE
ENTER	file	READ	READ
GET	file	READ	READ
LOAD	file	READ	READ
LOADSUB	file	READ	READ
LOCK	file	at least 1	READ
MASS STORAGE IS	directory	–	READ
OUTPUT	file	WRITE	READ
PLOTTER IS	file	at least 1[1]	READ
PRINTER IS	file	at least 1[1]	READ
PROTECT	either	MANAGER	READ
PURGE	either	MANAGER	READ & WRITE
RENAME	either	MANAGER	READ & WRITE
RE-SAVE	file	READ & WRITE	READ & WRITE
RE-STORE	file	READ & WRITE	READ & WRITE
RE-STORE KEY	file	READ & WRITE	READ & WRITE
SAVE	file	–	READ & WRITE
STORE	file	–	READ & WRITE
STORE KEY	file	–	READ & WRITE
STORE SYSTEM	file	–	READ & WRITE
TRANSFER			
inbound	file	READ	READ
outbound	file	WRITE	READ

Dash (–) means "does not apply."

1 The statement, however, is not useful without WRITE access to the file.

Using Protected Files Created on a Pascal Workstation

The password protection assigned with the Pascal Filer's Access command imposes some restrictions on the use of BASIC keywords with a file protected with that command.

If a Pascal file's SEARCH capability alone is protected, the BASIC catalog listing will show the file's READ capability as public. The protection assigned for SEARCH, however, limits the types of BASIC read operations that can be performed on that file without the assigned password. For example, you can catalog a directory whose READ access capability is public and whose SEARCH access capability is not, but you cannot access any of the files or directories within that directory.

Similarly, the MANAGER access capability in BASIC encompasses the Pascal MANAGER, CREATELINK and PURGELINK capabilities.

BASIC vs. Pascal Protections

BASIC Access Capability	Equivalent Pascal Access Capability
MANAGER	MANAGER, CREATELINK, PURGELINK
READ	READ, SEARCH
WRITE	WRITE

Summary of BASIC Keyword Use on SRM

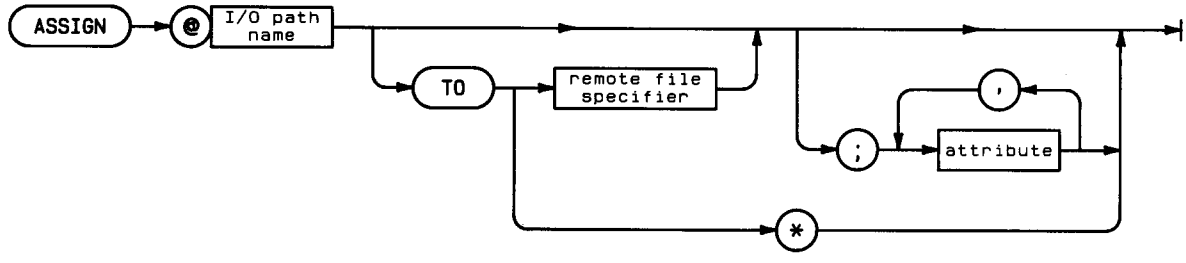
This section lists, in alphabetical order, the BASIC keywords that can be used with SRM and those that are unique to SRM (CREATE DIR, LOCK, the CAT PROTECT option, UNLOCK). Each keyword description in this section discusses only uses or features of the keyword that apply to its use on SRM.

Syntax diagrams appear only with those keywords requiring a different syntax for use with SRM. Where syntax diagrams are not included, you may follow the syntax described in the body of this manual, substituting **remote file specifier** syntax (described in the previous section) wherever "file specifier" is indicated in the keyword's syntax.

For access capability requirements, refer to the chart in the previous section.

ASSIGN

With SRM, I/O path names can be assigned to remote files, attributes can be assigned to the I/O path, and I/O paths can be closed. The following syntax and discussion describes only the use of ASSIGN with remote files. See the body of this manual for details of other uses of ASSIGN and the description of attributes associated with ASSIGN.



Example Statements

```
ASSIGN @Remote_file TO "DIR_JOHN/dir_proj/file1"
ASSIGN @File TO "P1/FredsData<pass>:REMOTE"
```

Semantics

Assigning an I/O path name to a remote file associates the I/O path with the file at the specified or default mass storage location.

ASSIGN opens any existing ASCII or BDAT file, regardless of protection on the file **except** when all access capabilities (MANAGER, READ and WRITE) are taken from the public. Attempts to use ASSIGN with a file whose capabilities are fully protected (without supplying the necessary passwords) result in Error 62.

In all other instances, a file's access capabilities are not checked at ASSIGN time. The specified operation on the file associated with the I/O path name is not executed, however, unless the file has the necessary access capability for that operation. For example, you may ASSIGN an I/O path name to a file that has only the READ capability public, but attempting to perform an OUTPUT operation without the password protecting the WRITE access capability generates Error 62.

ASSIGN does not create a file.

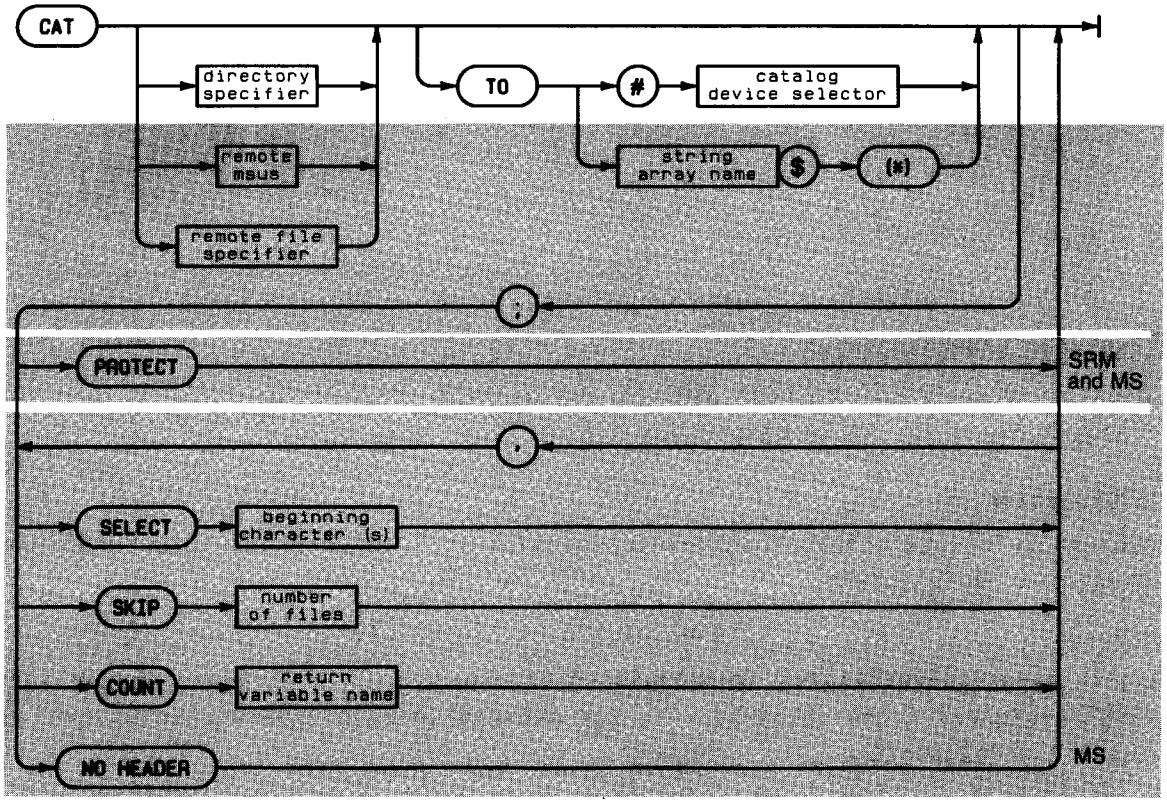
ASSIGN and Locked Files

Existing ASCII or BDAT files opened via ASSIGN are opened in shared mode, which means that several users can open a file at the same time. If you lock a file (refer to LOCK) and subsequently open that file via ASSIGN using the same @<*name*> (for example, to reset the file pointer), the ASSIGN automatically unlocks the file (refer to UNLOCK). To maintain sole access to the file, you must LOCK it again.

Closing an I/O path via ASSIGN (ASSIGN @...TO *) unlocks as well as closes the file (regardless of the number of LOCKs in effect for the file at the time).

CAT

With SRM, CAT lists all or specified portions of the contents of a directory or information regarding a specified PROG file. SRM adds the PROTECT option to the CAT statement. For a full description of the CAT statement syntax and CAT options, refer to the body of this manual.



Example Statements

```
CAT
CAT TO #701
CAT ":REMOTE"
CAT "../../.."
CAT "DIR1/DIR2"
CAT "A/B/C:REMOTE"
CAT "My_File";PROTECT
CAT ":REMOTE; LABEL Mastervol"
CAT;SELECT "D", SKIP Ten_files
CAT TO Directory$(*); NO HEADER
```

Semantics

To catalog remote directories, either you must include a remote msus in the CAT statement or the latest MASS STORAGE IS statement must have specified the desired remote msus. A catalog entry is listed for each file in the working or explicitly specified directory.

CAT to a Device

The catalog listing format used by the SRM system depends upon the line-width capacity of the device used for display.

When cataloging a remote directory on a 50-column display, the SRM system uses the following catalog format:

```
                                                        header
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0                   line 1
LABEL:     Disc1                                        line 2
FORMAT:    SDF                                          line 3
AVAILABLE SPACE:        54096                           line 4
                  PUB  FILE     NUMBER  RECORD  OPEN    line 5
FILE NAME         ACC  TYPE    RECORDS  LENGTH  STAT    line 6
================ ===  =====  =======  ======  ====
Common_data       MRW  ASCII       48     256  OPEN
Personal_data          BDAT       33     256  LOCK
Program_alpha      RW   PROG       44     256
HP9845_DATA         R   DATA?      22     256
HP9845_STORE      MRW  PROG?       9     256
Pascal_file,TEXT  MRW   TEXT      37     256
Program_500       MRW  PROG?      12     256
```

When cataloging a remote directory on an 80-column display, the SRM system uses the following catalog format:

```
                                                                                    header
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0                                               line 1
LABEL:     Disc1                                                                    line 2
FORMAT:    SDF                                                                      line 3
AVAILABLE SPACE:    54096                                                           line 4
                      SYS  FILE   NUMBER   RECORD     MODIFIED          PUB OPEN   line 5
FILE NAME         LEV TYPE  TYPE  RECORDS   LENGTH DATE        TIME    ACC STAT   line 6
================ === ==== ===== ======== ======== =============== ===  ====
Common_data         1      ASCII       48      256  2-Dec-83  13:20 MRW  OPEN
Personal_data       1 98X6  BDAT        33      256  2-Dec-83  13:20      LOCK
Program_alpha       1 98X6  PROG        44      256  3-Dec-83  15: 6  RW
HP9845_DATA         1 9845  DATA        22      256 10-Oct-83   8:45  R
HP9845_STORE        1 9845  PROG         9      256 10-Oct-83   8:47 MRW
Pascal_file,TEXT    1 PSCL  TEXT        37      256 11-Nov-83  12:25 MRW
Program_500         1 9000  PROG        12      256 13-Dec-83   9:54 MRW
```

The *header* gives you the following information:

line 1 Directory name and remote msus. The full path to the specified directory is displayed. Passwords used in the path are not displayed.

If the directory path specifier contains more characters than the display width, the last 49 or 79 characters (depending on catalog format) in the path specifier are shown. An asterisk (*) as the leftmost character in the path specifier indicates that leading characters were truncated for the display.

The system remembers a maximum of 160 characters for any directory path specifier at a single time. If a path specifier contains more than 160 characters, the excess characters are removed from the beginning of the specifier and are not retained. This restriction does not affect movement within the directory structure.

line 2 Volume label of the volume containing the directory.

line 3 Directory format, such as SDF (Structured Directory Format). See your disc's operating manual for details.

line 4 Number of bytes available on the volume (given in increments of 256 bytes).

lines 5 and 6 Labels for columns of information given for each file. The information provided is summarized below.

The `FILE NAME` column lists the names of the remote files and directories in the directory.

The `LEV` column (80-column format only) shows the level of the file relative to the current working directory or specified directory. The level is always shown as `1` in directory listings for Series 200/300 workstations.

The `PUB ACC` column lists the access capabilities available to all SRM system users. The three capabilities are READ, (`R`) WRITE (`W`) and MANAGER (`M`).

- Public MANAGER capability on a file or directory allows any user on the SRM system to PURGE that file or directory and to modify or add to its passwords (with PROTECT). Password-protected MANAGER capability gives users who supply the required password both READ and WRITE capabilities as well as MANAGER capability.
- READ capability on a directory allows you to access any file or directory in the directory. The READ capability on a file allows you to read the contents of the file.
- WRITE capability on a directory allows you to create or delete a file or directory in that directory. The WRITE capability on a file allows you to write information into that file.

The `SYS TYPE` column (80-column format only) shows the type of system used to create the file. The system type is not shown for ASCII files and directories. `98XG` denotes a Series 200/300 computer. If the SRM system does not recognize the system type, a coded identifier, obtained from the system being identified, appears in this column.

The `FILE TYPE` column indicates the file's type. Directories are indicated as type `DIR`. In the 50-column format, a question mark is appended to the file type if the file was not created on a Series 200/300 computer and was a type other than ASCII or DIR.

File types recognized by the BASIC system on SRM are: ASCII, BDAT, BIN, DIR, PROG, and SYSTM, as well as Series 200/300 Pascal and Series 500 file types.

If the system does not recognize a file's type, a coded file type identifier, obtained from the system originating the file, appears in the `FILE TYPE` column.

The NUMBER OF RECORDS column indicates the number of records in the file and the RECORD LENGTH column indicates the number of bytes constituting each of the file's records.

The MODIFIED columns (80-column format only) show the date and time the file's contents were last changed.

The OPEN STAT column shows whether the file is currently open (OPEN), locked (LOCK) or corrupt (CORR). OPEN indicates that the file has been opened, via ASSIGN, by a user. An open file is available for access from other workstations. LOCK means the file is accessible only from the workstation at which the file was locked. CORR indicates that the disc lost power while accessing the file, possibly altering the file's contents. If the entry is blank, the file is closed and available to any user.

Note

If a file's status is shown as corrupt (CORR), you should run the DSCK Utility program to check the directory structure and its integrity on the SRM system disc. Refer to the *SRM Operating System Manual* for details.

CAT to a String Array

Regardless of the workstation's display width, a CAT to a string array always produces the 80-column format.

The PROTECT Option

PROTECT is a CAT option provided by the SRM BIN file and available only on SRM. This option also requires the MS BIN file. The PROTECT option displays the password(s) and associated access capabilities for the specified file or directory.

For example, the statement:

```
CAT "Test_file<MPASS>:REMOTE";PROTECT
```

might produce the display:

```
PASSWORD             CAPABILITY
================ ==================
MPASS            MANAGER,READ,WRITE
WPASS            WRITE
RPASS            READ
PASSWORD         MANAGER
```

Use of this option requires MANAGER access capability on the file or directory. If the MANAGER capability is public, the PROTECT option may be used by any SRM user.

PROTECT must be specified separately from other CAT options, and is allowed only with SRM files and directories. Using PROTECT with media other than SRM results in ERROR 1 Configuration error.

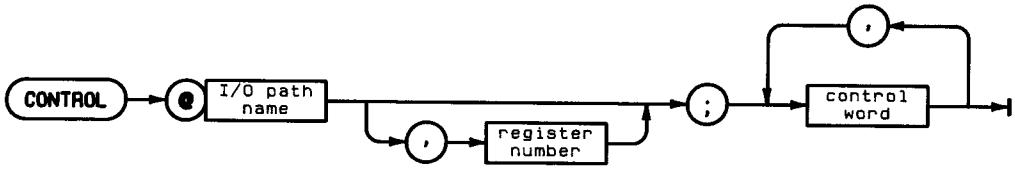
CHECKREAD

For SRM, CHECKREAD is implemented as a no-op, because the CHECKREAD function is already performed for every read and write statement on the SRM. Further checking places overhead on the system and doing so would not be accurate. With SRM, CHECKREAD **may or may not** cause a true write to the disc, while its read would probably only access the buffers in the SRM system. SRM's internal read and write checking and the automatic checking on the link make using CHECKREAD unnecessary.

CONTROL

With SRM, CONTROL sends control information to the internal table associated with an I/O path name assigned to an ASCII or BDAT file (see ASSIGN). Refer to the CONTROL keyword entry in the body of this manual for a full explanation of CONTROL syntax.

Control registers are listed in the "I/O Path Status and Control Registers" table in the Interface Registers section of this manual.

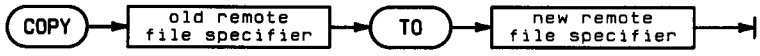


Example Statement

```
CONTROL @Rand_file,7;File_length
```

COPY

With SRM, COPY allows copying of individual remote files. Remote directories and volumes cannot be copied.



Example Statements

```
COPY "/Dir_1/File_1" TO "Dir_3/File_1"
COPY "File:INTERNAL" TO "File:REMOTE 21,0"
COPY Dir_path$&File$&Msus& TO "File:INTERNAL"
```

Semantics

The contents of the old remote file are copied to the new remote file and an entry is placed in the destination directory. The old and new remote files may be in the same directory, but the new remote file's name must be unique.

Although you may include a password in the new remote file specifier, the system ignores the password. If you wish to protect access to the new file, you must assign the password with PROTECT.

CREATE ASCII

With SRM, CREATE ASCII creates a new remote ASCII file, placing a corresponding directory entry in the current working directory or specified remote directory.

Example Statements

```
CREATE ASCII "Text03", 100
CREATE ASCII "/Dir1/Dir2/ASCIIFILE",25
```

Semantics

The name of the newly-created ASCII file must be unique within its containing directory.

CREATE ASCII **does not** open the file. Files are opened with the ASSIGN statement. If an error occurs during execution of CREATE ASCII, no directory entry is made and the file is not created.

The specified number of records determines the number of physical records for a remote ASCII file's initial space allocation. The physical records of an ASCII file have a fixed length of 256 bytes. (Logical records have variable lengths, determined automatically when an OUTPUT, SAVE or RE-SAVE statement is used.)

Storage space for subsequent saving of remote files is allocated only when needed. When data is added to a remote file such that saving the modified file would overflow the file's current space allocation, the SRM system adds another extent. An extent is a space allocation whose size is determined by multiplying the specified number of records by the record size.

When the remote file is created, all access capabilities are public. Including a password in the CREATE ASCII statement's remote file specifier does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but passwords in the CREATE ASCII statement are ignored.

CREATE BDAT

With SRM, CREATE BDAT creates a new remote BDAT file, placing a corresponding directory entry in the current working directory or specified remote directory.

Example Statements

```
CREATE BDAT "File",Records,Rec_size
CREATE BDAT "/Dir1/Dir2/BDATFILE",25,128
CREATE BDAT "Dir/File:REMOTE",10
```

Semantics

The name of the newly-created BDAT file must be unique within its containing directory.

CREATE BDAT **does not** open the file. Files are opened with the ASSIGN statement. If an error occurs during execution of CREATE BDAT, no directory entry is made and the file is not created.

The specified number of records determines the number of physical records for a remote BDAT file's initial space allocation. The length of a BDAT file's physical records is either specified by the record size parameter or set to 256 bytes if no record size is specified.

Storage space for subsequent saving of remote files is allocated only when needed. When data is added to a remote file such that saving the modified file would overflow the file's current space allocation, the SRM system adds another extent. An extent is a space allocation whose size is determined by multiplying the specified number of records by the record size. On SRM, CREATE BDAT does not allocate a sector for system use, as it does with local files.

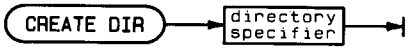
When the remote file is created, all access capabilities are public. Including a password in the CREATE BDAT statement's remote file specifier does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but passwords in the CREATE BDAT statement are ignored.

The data in remote BDAT files can be accessed both serially and randomly.

CREATE DIR

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a directory in either the current working directory or in the specified remote directory of an SRM mass storage device.



Example Statements

```
CREATE DIR "Under_work_dir"
CREATE DIR "Level1/Level2/New_dir:REMOTE 21,3"
CREATE DIR "/Level1/Level2/New_dir"
CREATE DIR "Level1<RWpassword>/New_dir"
```

Semantics

This statement creates a special 24-byte file of type DIR and a corresponding directory entry in the current working directory or specified remote directory. The DIR file, or directory, keeps information on files and directories immediately subordinate to itself.

The name of the newly-created directory must be unique within its containing directory.

Like remote data files, DIR files are extensible. Extents are added in 24-byte increments. As each directory or data file is created within a directory, a 24-byte record identifying the addition is added to the DIR file.

If no directory path is included in the directory specifier, the directory is created within the current working directory (the directory specified in the latest MASS STORAGE IS statement). To specify a target directory other than the current working directory, specify the directory path to the desired directory.

You cannot assign passwords to a directory when you create it. Passwords are assigned only via PROTECT. If an error occurs during execution of CREATE DIR, the directory entry in the superior directory is not made, and the directory is not created.

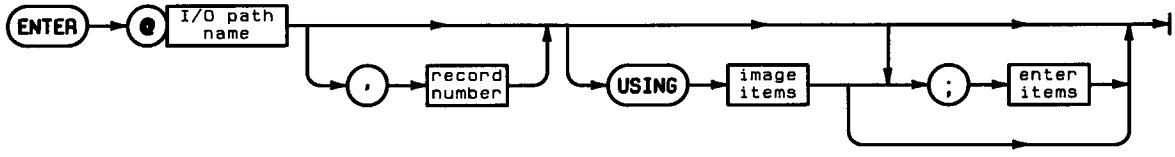
DIR files are opened with the MASS STORAGE IS (MSI) statement.

Refer to the section on "Syntax for Remote File and Directory Specification" earlier in this section for details on the semantics of directory specifiers.

ENTER

With SRM, ENTER is used to read data from a remote data file identified by an I/O path name and to assign the value(s) to variable(s). (See also ASSIGN.)

The capabilities available for using ENTER with remote files are the same as those for using ENTER with local files. Refer to the ENTER keyword entry in the body of this manual for a full explanation of ENTER syntax.



Example Statements

```
ENTER @Remote_file,REC;Alpha$,Beta$,Gamma$
ENTER @Name_of;A,B
```

Semantics

Entering data from remote files requires the READ access capability on the superior directory and on the file from which the data are to be read. If this capability is not public or if a password protecting this capability was not used at the time the file was ASSIGNed, an error is reported.

GET

With SRM, GET reads the specified remote ASCII file and attempts to store the data in memory as program lines.

Example Statements

```
GET "Filename:REMOTE"
GET "/Dir1/Dir2/Dir3/filename<READPass>"
```

Semantics

You may use GET with any ASCII file whose data is in the format of a BASIC program (that is, having numbered lines). Although you may also use GET with ASCII files created on non-Series 200/300 SRM workstations (HP 9835, HP 9845 or Model 520), any line that is not valid BASIC syntax for Series 200/300 computers is stored as a commented (!) program line.

When used on SRM, GET is executed in shared mode, which means that several users can get one file at the same time. Attempts to get a locked file (see LOCK) result in Error 453. Additionally, you cannot get a file while it is being saved. The SAVE and RE-SAVE operations open the file in exclusive mode (shown as LOCK in a CAT listing) and enforce that status until the SAVE or RE-SAVE is complete. While in exclusive mode, the file is accessible only to the SRM workstation executing the SAVE or RE-SAVE.

INITIALIZE

INITIALIZE can be used to initialize local mass storage media only. An error will result if you try to initialize a shared system volume.

LOAD

With SRM, LOAD loads the contents of remote PROG or BIN files into memory, or sets the typing-aid definitions of the softkeys according to the contents of a remote BDAT file.

Example Statements

```
LOAD "Program_z"
LOAD "/Dir1/Dir2/Prog2",500
LOAD "Dir3/Prog_1:REMOTE"

LOAD BIN Dir$&File$&Msus$
LOAD BIN "dir1/dir2/bin_file<Readpass>:REMOTE 21,5;LABEL Disc"

LOAD KEY "KEYS:REMOTE"
LOAD KEY "/Dir1/Dir2/Keyfile"
```

Semantics

LOAD

LOAD can be used with remote PROG files (created with the STORE statement). LOAD is executed in shared mode, which means that several users can load a file at the same time. Files being stored with the STORE or RE-STORE statements are locked during that operation and cannot be accessed for loading.

LOAD BIN

LOAD BIN can be used with remote BIN files. LOAD BIN is executed in shared mode, which means that several users can load a BIN file at the same time.

BIN files can be loaded into a workstation from the SRM without the SRM BIN file present in the workstation. Refer to the "Booting From the SRM" section of the SRM chapter in *BASIC Programming Techniques* for more details.

LOAD KEY

LOAD KEY can be used with remote BDAT files (created with the STORE KEY statement). LOAD KEY is executed in shared mode, which means that several users can perform a LOAD KEY from a BDAT file at the same time. Files being stored with the STORE KEY or RE-STORE KEY statements are locked during that operation and cannot be accessed for loading.

LOADSUB

With SRM, LOADSUB allows you to load subprograms from a remote PROG file into your workstation.

Example Statements

```
LOADSUB FROM "APSUBS"
LOADSUB FNReplace$ FROM "SUBFILE"
LOADSUB ALL FROM Subfile$
LOADSUB ALL FROM "Dir3/Progfile<Readpass>"
LOADSUB ALL FROM "/Dir1/Dir2/Prog23"
```

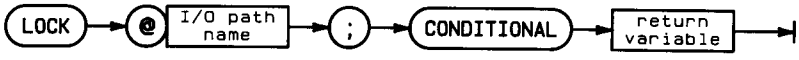
Semantics

With SRM, LOADSUB is executed in shared mode, which means that several workstations can perform a LOADSUB of a file at the same time. PROG files being stored with the STORE or RE-STORE statement are locked during that operation and cannot be accessed for loading.

LOCK

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement prevents other SRM workstations from accessing the remote file to which the I/O path name is currently assigned (see ASSIGN).



Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path	any valid name (See Glossary.)
return variable	name of a numeric variable	any valid name (See Glossary.)

Example Statements

```
LOCK @File;CONDITIONAL Result
LOCK @Ascii_1;CONDITIONAL Error_number
```

Semantics

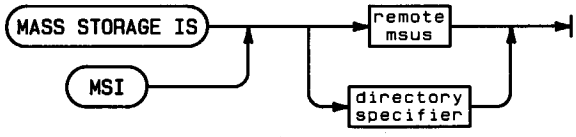
This statement establishes sole access to a file that has been opened with an ASSIGN statement. This exclusive access remains assigned to the workstation executing the LOCK statement until an UNLOCK statement is executed by that workstation. The UNLOCK function is also a result of SCRATCH A, (RESET) and ASSIGN...TO * (explicitly closing an I/O path).

A file may be locked several times. The system counts the number of LOCKs on a file, and an equal number of UNLOCKs must be executed to unlock the file. When an I/O path name is closed (for example, by ASSIGN...TO *), **all** LOCKs of that I/O path name are cleared.

If the LOCK is successful, the value of the return variable will be zero. Otherwise, the return variable's value will be the error number corresponding to the cause of the LOCK's failure.

MASS STORAGE IS

With SRM, MASS STORAGE IS specifies the SRM working directory.



Example Statements

```
MSI "Dir1/Dir2/Project_dir"
MSI ",,"
MASS STORAGE IS ",<password>"
MSI ":REMOTE"
```

Semantics

SRM allows directories or volumes to be assigned as system mass storage. If you specify the volume password in an MSI statement, that password is automatically applied to all accesses that use the default msus (that is, no remote msus is specified in the remote file specifier) until a remote msus is included in a subsequent MSI.

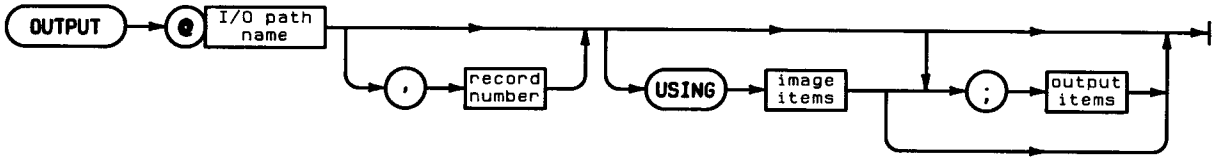
ON TIMEOUT

With SRM, ON TIMEOUT defines and enables a branch resulting from an I/O timeout on the specified SRM interface. Although ON TIMEOUT is supported on SRM, **its use should be avoided** because the asynchronous nature of the SRM system does not allow predictable results.

A TIMEOUT occurring during statements such as RE-SAVE and RE-STORE may leave a temporary file on the mass storage device. The file's name is a 10-character identifier (the first character is an alpha character, the rest are digits) derived from the value of the workstation's real-time clock when the TIMEOUT occurred. You may wish to check the contents of any such file before purging.

OUTPUT

With SRM, OUTPUT writes item(s) to the remote file to which the specified I/O path name is assigned (see ASSIGN). Refer to the OUTPUT keyword entry in the body of this manual for a full explanation of OUTPUT syntax.



Example Statement

```
OUTPUT @File;Array(*),END
```

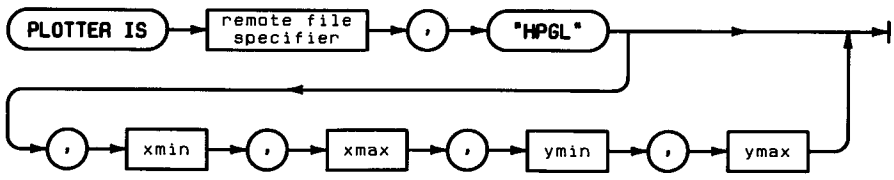
Semantics

You must have WRITE access capability on the remote file to output data to the file. If this capability is not public or if a password protecting this capability was not used at the time the file was ASSIGNed, Error 62 is reported.

If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the disc contains enough unused storage space). Refer to the "System Concepts" section of the SRM chapter in *BASIC Programming Techniques* for more details on the extensible nature of remote files.

PLOTTER IS

With SRM, PLOTTER IS causes all subsequent plotter output to go to the specified remote BDAT file. Refer to the PLOTTER IS keyword entry in the body of this manual for a full explanation of PLOTTER IS syntax.



Example Statements

```
PLOTTER IS "/PL/Plotfile"
PLOTTER IS "Plotfile:REMOTE","HPGL",6.25,256.25,6.975,186.975
```

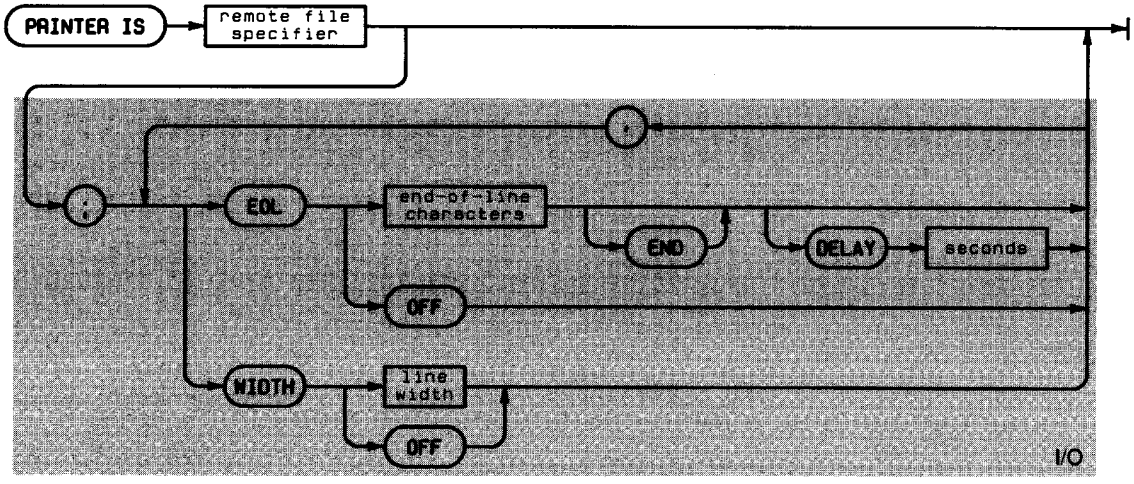
Semantics

If the specified remote file is in the SRM plotter spooler directory and the file contains data, when the file is closed the SRM system sends the data to the plotting device and then purges the file. You may close the file by executing another PLOTTER IS statement, SCRATCH A or SCRATCH BIN, or by pressing (RESET).

No end-of-file error occurs on SRM. If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the disc contains enough unused storage space). Refer to the "System Concepts" section of the SRM chapter in *BASIC Programming Techniques* for more details on the extensible nature of remote files.

PRINTER IS

With SRM, PRINTER IS specifies a remote BDAT file as the system printing file. Refer to the PRINTER IS keyword entry in the body of this manual for a full explanation of PRINTER IS syntax.



Example Statements

```
PRINTER IS "Spooler:REMOTE"
PRINTER IS "My_dir/Temp_print";WIDTH 80
```

Semantics

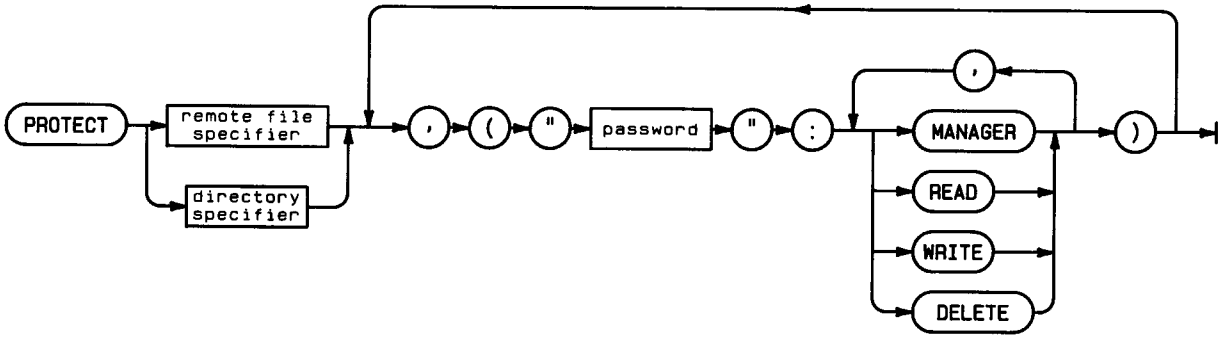
The system printing file receives all data sent by the PRINT statement, all data sent by CAT and LIST statements in which the destination is not explicitly specified, and other output generated by the BASIC system.

If the specified remote file is in the SRM printer spooler directory and the file contains data, when the file is closed, the SRM system sends the data to the printer and then purges the file. You may close the file by executing another PRINTER IS statement, or a SCRATCH A or SCRATCH BIN command.

No end-of-file error occurs on SRM. If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the media contains enough unused storage space). Refer to the System Concepts'' section of the SRM chapter in *BASIC Programming Techniques* for more details on the extensible nature of remote files.

PROTECT

With SRM, this statement protects access capabilities by assigning passwords to remote files and directories. The use of PROTECT with SRM is distinct from its use with local files (described in the body of this manual).



Example Statements

```
PROTECT "dir:REMOTE",("mgr":MANAGER),("rw":READ,WRITE)
PROTECT "File<rw>",("rw":DELETE)
```

Semantics

PROTECT allows you to control access to remote files and directories by protecting access capabilities with password(s). Access capabilities are either public (available to all SRM users) or password-protected (available only to users supplying the correct password with the file or directory specifier).

The three access capabilities – MANAGER, READ and WRITE – are public unless the PROTECT statement associates a password with one or more of those capabilities.

Once the capability on a given file or directory is password-protected, the capability can be exercised on the file or directory only if the correct password is included in the file or directory specifier. For instance, if a file's READ capabilities are protected, any user wishing to execute a command or statement that reads the file must supply a password protecting the file's READ capability.

MANAGER

Public MANAGER capability allows any SRM user to PROTECT, PURGE or RENAME a file or directory. Password-protected MANAGER capability provides READ and WRITE, as well as MANAGER, access capabilities to users who know the password.

You must have MANAGER capabilities on a file or directory to PROTECT the access capabilities on that file or directory. This includes adding, deleting and changing passwords.

READ

READ capability on a file allows use of commands and statements that read the contents of a file (for example: ENTER, LOAD, GET). READ capability on a directory allows you to read the files in the directory (CAT), or to "pass through" a directory by including the directory name (and password, if assigned) in a directory path.

WRITE
WRITE capability on a file allows use of commands and statements that write to the file (for example: OUTPUT, RE-SAVE, RE-STORE). WRITE capability on a directory allows use of commands that add or delete file names in the directory (for example: SAVE, STORE, PURGE, CREATE, RENAME).

Use of PROTECT
Each PROTECT statement allows up to six password/capability combinations per statement. The number of PROTECT statements that can be executed for each file or directory is unlimited, however, as long as each password is unique.

Successive associations of capabilities with the same password are not cumulative. To retain previous capability assignments for a file or directory, you must include those assignments in subsequent PROTECT statements designating the same password for that file or directory.

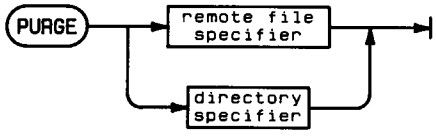
For example, say you protected the READ access capability on a file with the password *passme* then wanted to change that assignment so that *passme* would protect both the READ and WRITE access capabilities for that file. If you executed a second PROTECT statement associating *passme* with the WRITE capability only, *passme* would no longer protect the READ capability. Instead, you should specify the password and **both** the READ and WRITE capabilities in the second PROTECT statement.

To modify the access capabilities protected by a password, execute the PROTECT with the existing password and the new password/capability pair(s).

The secondary keyword DELETE is used to delete existing password assignments for a file or directory. To be effective, DELETE must be the only secondary keyword used with a password/capability pair in the PROTECT statement. Otherwise, DELETE is ignored. MANAGER capability is required to perform the DELETE. A DELETE executed without MANAGER capability results in a protect code violation error.

PURGE

With SRM, PURGE deletes a file entry from a directory or an empty remote directory from its superior directory.



Example Statements

```
PURGE "File"
PURGE "Dir_a<RWpass>/File<MGRpass>"
PURGE "Dir1/Dir2/Dir3"
```

Semantics

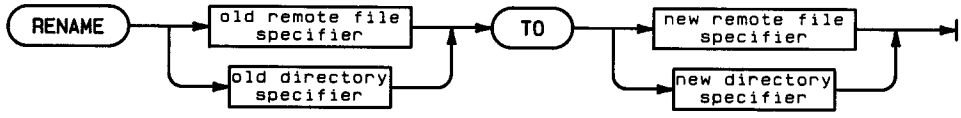
Only remote files and directories that are closed can be purged. Remote files are closed by ASSIGN...TO * (explicitly closes an I/O path). SCRATCH A closes both directories and remote files. All remote files except those opened with the PRINTER IS statement are also closed by (RESET). The current working directory is closed by an MSI to a different directory.

Once a file or directory is purged, its contents cannot be recovered.

To be purged, directories must be empty (must not contain any subordinate files or directories) as well as closed.

RENAME

With SRM, RENAME changes a remote file's name in a remote directory and performs limited file relocation.



Example Statements

```
RENAME "Old_name" TO "New_name"
RENAME "Dir1<RWpass>/F1<MGRpass>" TO "Dir2<RWpass>/F1"
RENAME "THIS:REMOTE" TO "THAT"
```

Semantics

RENAME can be used to change the name of remote files and directories or to move files within the directory structure. Directories cannot be moved with RENAME. Moving of files must occur within a single volume. If you move a file with RENAME, the original file ("old remote file specifier") is purged.

A maximum of nine names (file or directory) are allowed in the combined file/directory specifiers in the RENAME statement. No more than six names are allowed in either specifier individually. (The number of names in the old file/directory specifier plus the number of names in the new file/directory specifier must not exceed nine.)

Files and directories must be closed before being renamed. Remote files are closed by ASSIGN...TO * (explicitly closes an I/O path). SCRATCH A closes both directories and remote files. All remote files except those opened with the PRINTER IS statement are also closed by (RESET). The current working directory is closed by an MSI to a different directory.

Existing passwords are retained by the renamed file or directory. The new file name must not duplicate the name of any existing file in the destination directory.

RE-SAVE

With SRM, RE-SAVE creates a remote ASCII file and copies program lines as strings into that file.

Example Statements

```
RE-SAVE "File"
RE-SAVE "Dir<RWpass>/File<RWpass>"
```

Semantics

RE-SAVE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the RE-SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-SAVE.

If the file does not already exist, RE-SAVE performs the same action as SAVE. Including a password in the RE-SAVE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the specifier of a remote file that does not already exist, but the system ignores the password.

Passwords assigned to an existing file are retained when a RE-SAVE is performed on the file. If you specify the wrong password on a protected file, the system returns an error message.

Use of RE-SAVE on SRM may leave temporary files on the mass storage media if CLR I/O or RESET is pressed or a TIMEOUT occurs during the RE-SAVE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

RESET

With SRM, this statement resets the pointers of a remote file identified by an I/O path name (see ASSIGN).



Example Statement

```
RESET @Remote_file
```

RE-STORE

With SRM, RE-STORE creates a remote file and stores the BASIC program or typing-aid key definitions in that file.

Example Statements

```
RE-STORE "Prog_a"
RE-STORE "Dir<RWpass>/Prog_z<RWpass>"
RE-STORE KEY "KEYS:REMOTE"
RE-STORE KEY "TYPING"
```

Semantics

RE-STORE creates a remote PROG file, and RE-STORE KEY creates a remote BDAT file.

RE-STORE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the RE-STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-STORE.

If the file does not already exist, RE-STORE performs the same action as STORE. Including a password in the RE-STORE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the specifier of a remote file that does not already exist, but the system ignores the password.

Passwords assigned to an existing file are retained when a RE-STORE is performed on the file. If you specify the wrong password on a protected file, the system returns an error message.

Use of RE-STORE on SRM may leave temporary files on the mass storage media if (CLR I/O) or (RESET) is pressed or a TIMEOUT occurs during the RE-STORE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

SAVE

With SRM, SAVE creates a remote ASCII file and copies program lines as strings into the file.

Example Statements

```
SAVE "THE_WHALES"
SAVE "Dir<RWpass>/File"
SAVE "Ascii_file:REMOTE"
```

Semantics

SAVE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the SAVE.

Including a password in the SAVE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

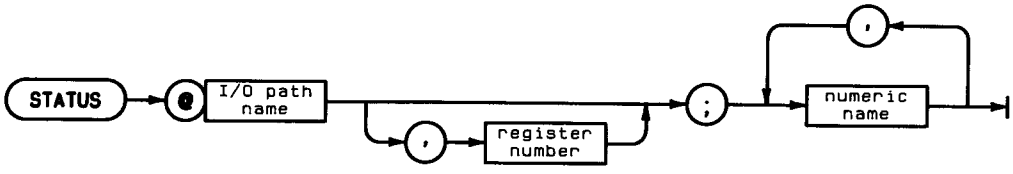
SCRATCH A

With SRM, SCRATCH A releases the system resources allocated to the workstation executing the SCRATCH A, making those resources available to other SRM workstations. SCRATCH A closes all files and directories, and resets the workstation's working directory to the default msus (the mass storage unit from which the workstation booted).

If the workstation has Boot ROM version 3.0 or later, and booted from the SRM, SCRATCH A resets the working directory to the root of the default system volume. If the workstation has an earlier version boot ROM or Boot ROM 3.0L, SCRATCH A resets the working directory to the device from which the workstation booted (for example, :INTERNAL if the workstation booted from a built-in drive).

STATUS

With SRM, STATUS returns the contents of I/O path name status registers (see ASSIGN). Refer to the STATUS keyword entry in the body of this manual for a full explanation of STATUS syntax. Status registers are listed in the "I/O Path Status and Control Registers" table in the Interface Registers section of this manual.



Example Statement

```
STATUS @File,5;Record
```

STORE

With SRM, STORE creates a remote file and stores a program or typing-aid key definitions into it.

Example Statements

```
STORE "Prog32"
STORE "Dir<RWpass>/Program"
STORE KEY "KEYS:REMOTE"
STORE KEY "/USERS/KRIS/TYPING"
```

Semantics

STORE creates a remote PROG file, and STORE KEY creates a remote BDAT file.

STORE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the STORE.

Including a password in the STORE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

STORE SYSTEM

With SRM, STORE SYSTEM stores the entire BASIC operating system currently in memory (including any BIN files) into the specified remote file.

Example Statements

```
STORE SYSTEM "SYSTEM_B1:REMOTE"
STORE SYSTEM "/SYSTEMS/SYSTEM_NEW"
```

Semantics

Including a password in the STORE SYSTEM statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

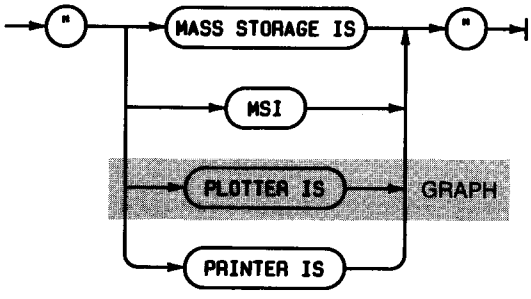
The READ access capability on the system file created with STORE SYSTEM must be public to allow use of the file for booting.

SYSTEM$

With SRM, this function returns a string containing system status and configuration information.



literal form of type of information:



Example Statement

```
SYSTEM$("MSI")
SYSTEM$("PRINTER IS")
SYSTEM$("PLOTTER IS")
```

Semantics

The system configuration information returned when SYSTEM$ is executed on SRM includes the full remote file specifier describing the file or directory about which the information is requested. Passwords in the specifier are not included.

The system remembers a maximum of 160 characters for any one specifier. If a specifier contains more than 160 characters, the excess characters are removed from the beginning of the specifier and are not retained. An asterisk (*) as the leftmost character in the specifier indicates that leading characters were truncated for the function.

TRANSFER

With SRM, this statement initiates unformatted data transfers between the workstation and remote mass storage devices. Either the source or destination of the transfer is specified as an I/O path name assigned to a remote BDAT file (see ASSIGN). Refer to the TRANSFER keyword entry in the body of this manual for a full explanation of TRANSFER syntax.

Example Statements

```
TRANSFER @Buffer TO @File;CONT
TRANSFER @Dir_Path TO @Destination;COUNT 256
TRANSFER @Source TO @Buffer;DELIM "/"
TRANSFER @Path TO @Buffer;RECORDS 12,EOR(COUNT 8)
```

Semantics

TRANSFER behaves the same on SRM as with local mass storage, except that inbound and outbound transfer execution is not overlapped. Whereas the discs on the SRM may be cabable of overlapped operation, the SRM system performs TRANSFERs serially. This difference only matters in applications, such as data logging, where you may want a program to be able to execute other statements before the transfer has completed. For further details, refer to the "Transfer Performance" section in the "Advanced Transfer Techniques" chapter of the *BASIC Interfacing Techniques* manual.

UNLOCK

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to remove exclusive access (placed by the LOCK statement) to a remote file identified by an I/O path name (see ASSIGN).



Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path to a remote file	any valid name (See Glossary.)

Example Statements

```
UNLOCK @File
IF Done THEN UNLOCK @File
```

Semantics

This statement unlocks a file previously locked with the LOCK statement. While a file is locked, other SRM workstations cannot access the file. After UNLOCK, other users may access the file provided they possess the proper access capability (or capabilities).

If multiple LOCKs were executed on the file, the same number of UNLOCKs must be executed to unlock the file.

UNLOCK is performed automatically by SCRATCH A, (RESET) and ASSIGN...TO * (explicit closing of an I/O path).

SRM BASIC Error Codes for HP Series 200/300 Computers

450	Volume not found
451	Volume labels do not match
453	File is use
454	Directory formats do not match
455	Possibly corrupt file
456	Unsupported directory operation
457	Passwords not supported
458	Unsupported directory format
459	Specified file is not a directory
460	Directory not empty
461	Duplicate passwords not allowed
462	Invalid password
465	Invalid rename across volumes
466	Duplicate volume entries
481	File locked or open exclusively
482	Cannot move a directory via a RENAME
483	System down
484	Password not found
485	Invalid volume copy

Glossary

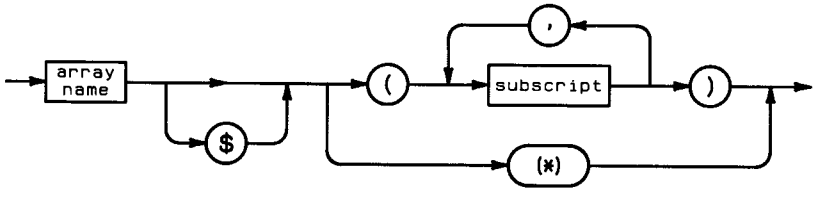
angle mode The current units used for expressing angles. Either degrees or radians may be specified, using the DEG or RAD statements, respectively. The default at power-on and SCRATCH A is radians.

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

array A structured data type that can be of type REAL, INTEGER, or string. Arrays are created with the DIM, REAL, INTEGER, ALLOCATE, or COM statements. Arrays have 1 to 6 dimensions; each dimension is allowed 32 767 elements. The lower and upper bounds for each dimension must fall in the range −32 767 (−32 768 for ALLOCATE) thru +32 767, and the lower bound must not exceed the upper bound. The default lower bound is the OPTION BASE value; the OPTION BASE statement can be used to specify 0 or 1 as the default lower bound. The default OPTION BASE in every environment is zero.

Each element in a string array is a string whose maximum length is specified in the declaring statement. The declared length of a string must be in the range 1 thru 32 767.

To specify an entire array, the characters (*) are placed after the array name. To specify a single element of an array, subscripts are placed in parentheses after the array name. Each subscript must not be less than the current lower bound or greater than the current upper bound of the corresponding dimension.



If an array is not explicitly dimensioned, it is implicitly given the number of dimensions used in its first occurrence, with an upper bound of 10. Undeclared strings have a default length of 18.

ASCII This is the acronym for "American Standard Code for Information Interchange". It is a commonly used code for representing letters, numerals, punctuation, special characters, and control characters. A table of the characters in the ASCII set and their code values can be found in the back of this manual.

bit This term comes from the words "binary digit". A bit is a single digit in base 2 that must be either a 1 or a 0.

byte A group of eight bits processed as a unit.

command A statement that can be typed on the input line and executed (see "statement").

context An instance of an environment. A context consists of a specific instance of all data types and system parameters that may be accessed by a program at a specific point in its execution. Context changes occur when subprograms are invoked or exited.

device selector A numeric expression used to specify the source or destination of an I/O operation. A device selector can be either an interface select code or a combination of an interface select code and a primary address. To construct a device selector with a primary address, multiply the interface select code by 100 and add the primary address.

Secondary addresses may be appended after a primary address by multiplying the device selector by 100 and adding the address. This may be repeated up to 6 times, adding a new secondary address each time. A device selector, once rounded, may contain a maximum of 15 digits.

When a device selector contains an odd number of digits, the leftmost digit is the interface select code. For an even number of digits, the leftmost two digits are the interface select code. For example, 70502 selects interface 7, primary address 05, and secondary address 02. Device selector 1516 selects interface 15 and primary address 16.

directory name A directory name (Shared Resource Management) is the same as a remote file name because a directory is a type of remote file. Directory names consist of from one to 16 characters, including uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

dyadic operator An operator that performs its operation with two expressions. It is placed between the two expressions. The following dyadic operators are available:

Operator	Operation
+	REAL or INTEGER addition
-	REAL or INTEGER subtraction
*	REAL or INTEGER multiplication
/	REAL division
^	Exponentiation
&	String concatenation
DIV	Gives the integer quotient of a division
MOD	Gives the remainder of a division
MODULO	Gives the modulo of a division
=	Comparison for equality
<>	Comparison for inequality
<	Comparison for less than
>	Comparison for greater than
<=	Comparison for less than or equal to
>=	Comparison for greater than or equal to
AND	Logical AND (Boolean >)
OR	Logical inclusive OR (Boolean <)
EXOR	Logical exclusive OR (Boolean <)

file name A file name consists of one to ten characters. Series 200/300 file names can contain uppercase letters, lowercase letters, numerals, the underbar (_), and CHR$(161) thru CHR$(254). LIF-compatible file names can contain only uppercase letters and numerals. The first character in a LIF-compatible file name must be a letter. (See "remote file name" for SRM.)

function A procedural call that returns a value. The call can be to a user-defined-function subprogram (such as FNInvert) or a machine-resident function (such as COS or EXP). The value returned by the function is used in place of the function call when evaluating the expression containing the function call.

graphic display unit This is 1/100 of the shortest axis on the plotting device. Graphic display units are the same size on both the X and Y axes. Abbreviated "GDU".

hard clip limits These are the physical limits of the plotting device.

hierarchy When a numeric or string expression contains more than one operation, the order of operations is determined by a precedence system. Operations with the highest precedence are performed first. Multiple operations with the same precedence are performed left to right. The following tables show the hierarchy for numeric and string operations.

Math Hierarchy

Precedence	Operator
Highest	Parentheses: (may be used to force any order of operations)
	Functions: user-defined and machine-resident
	Exponentiation: ^
	Multiplication and division: * / MOD DIV MODULO
	Addition, subtraction, monadic plus and minus: + -
	Relational operators: = <> < > <= >=
	NOT
	AND
Lowest	OR EXOR

String Hierarchy

Precedence	Operator
Highest	Parentheses
	Functions (user-defined and machine-resident) and substring operations
Lowest	Concatenation: &

I/O path A combination of firmware and hardware that can be used during the transfer of data to and from a BASIC program. Associated with an I/O path is a unique data type that describes the I/O path. This association table uses about 200 bytes and is referenced when an I/O path name is used. For further details, see the ASSIGN statement.

INTEGER A numeric data type stored internally in two bytes. Two's-complement representation is used, giving a range of −32 768 thru +32 767. If a numeric variable is not explicitly declared as an INTEGER, it is a REAL.

integer A number with no fractional part; a whole number.

interface select code A numeric expression that selects an interface for an I/O operation. Interface select codes 1 thru 7 are reserved for internal interfaces. Interface select codes 8 thru 31 are used for external interfaces. The internal HP-IB interface with select code 7 can be specified in statements that are restricted to external devices. (Also see "device selector".)

keyword A group of uppercase ASCII letters that has a predefined meaning to the computer. Keywords may be typed using all lowercase or all uppercase letters.

LIF This is the acronym for "Logical Interchange Format". This HP standard defines the format of mass storage files and directories. It allows the interchange of data between different machines. Series 200/300 files of type ASCII are LIF compatable. See "file name" for file name restrictions.

literal This is a string constant. When quote marks are used to delimit a literal, those quote marks are not part of the literal. To include a quote mark in a literal, type two consecutive quote marks (except in response to a LINPUT statement). The drawings showing literal forms of specifiers (such as file specifiers) show the quote marks required to delimit the literal.

logical pen See "pen".

monadic operator An operator that performs its operation with one expression. It is placed in front of the expression. The following monadic operators are available:

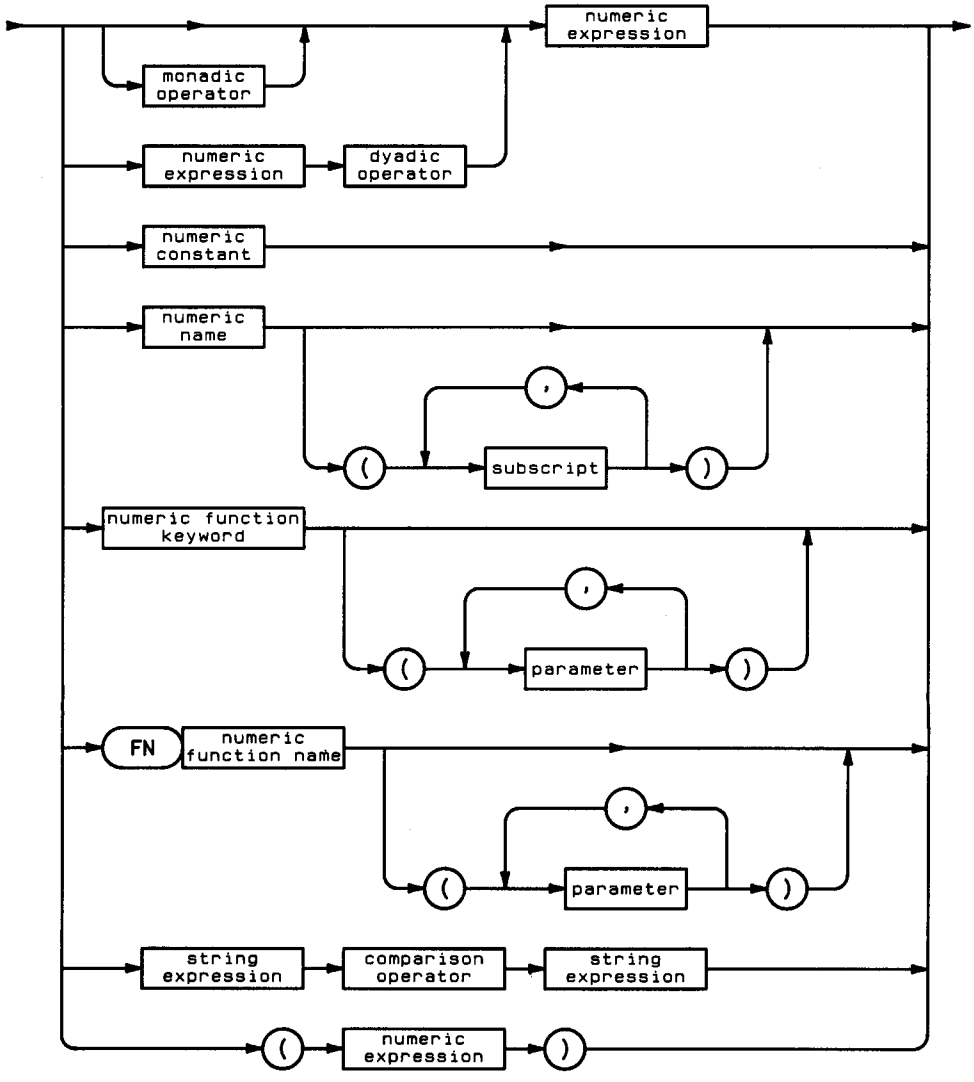
Operator	Operation
-	Reverses the sign of an expression
+	Identity operator
NOT	Logical complement (Boolean over-bar)

msus This is the acronym for "mass storage unit specifier". It is a string expression that specifies a device to be used for mass storage operations.

name A name consists of one to fifteen characters. The first character must be an uppercase ASCII letter or one of the characters from CHR$(161) thru CHR$(254). The remaining characters, if any, can be lowercase ASCII letters, numerals, the underbar (_), or CHR$(161) thru CHR$(254). Names may be typed using any combination of uppercase and lowercase letters, unless the name uses the same letters as a keyword. Conflicts with keywords are resolved by mixing the letter case in the name. (Also see "file name".)

node address An integer from 0 through 63 that identifies an SRM device (such as a workstation or controller).

numeric expression



Item	Description
monadic operator	An operator that performs its operation on the expression immediately to its right: + - NOT
dyadic operator	An operator that performs its operation on the two expressions it is between: ^ * / MOD DIV + - = <> < > <= >= AND OR EXOR MODULO
numeric constant	A numeric quantity whose value is expressed using numerals, decimal point, and exponent notation
numeric name	The name of a numeric variable or the name of a numeric array from which an element is extracted using subscripts
subscript	A numeric expression used to select an element of an array (see "array")
numeric function keyword	A keyword that invokes a machine-resident function that returns a numeric value
numeric function name	The name of a user-defined function that returns a numeric value
parameter	A numeric expression, string expression, or I/O path name that is passed to a function
comparison operator	An operator which returns a 1 (true) or a 0 (false) based on the the result of a relational test of the operands it separates: > < <= >= = <>

password Passwords are used to protect access to remote (SRM) files and directories. Passwords consist of one to 16 characters. All ASCII characters except ">" are allowed. Passwords are assigned by the PROTECT statement in BASIC or the Pascal Filer's Access command.

pen All graphical objects are "drawn" using mathematical representations in the computer's memory. This is done with the "logical pen". The logical pen creates five classes of objects: lines, polygons, labels, axes, and label locations (label locations are actually the position of an object, rather than an object).

Before these objects can be viewed, they are acted upon by various transformation matrixes, such as scaling and pivoting. No single transformation affects all the objects, and no object is effected by all the transformations.

The output of the transformations is used to control the "physical pen". The physical pen creates the image that you actually see on the plotter or CRT. Since the graphics statements used to create objects act directly upon the logical pen, and you can see only the output of the physical pen, the location of the logical pen may not always be readily discernable from what you see.

The following table shows which transformations act upon which objects.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			Note 4
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES & GRID)	X				
Location of Labels	Note 1	Note 3		Note 2	

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.
Note 4: RPLOT and IPLOT are affected by PDIR.

primary address A numeric expession in the range of 0 thru 31 that specifies an individual device on an interface which is capable of servicing more than one device. The HP-IB interface can service multiple devices. (Also see "device selector".)

program line A statement that is preceded by a line number (and an optional line label) and stored with the ENTER key into a program (see "statement").

protect code This is a non-listable, two-character code kept with a file description in the directory of a mass storage media. It guards against accidental changes to an individual file. When protect codes are specified, they may contain any number of characters. Blanks are trimmed from protect codes. When the result contains more than two characters, the first two are used as the actual protect code. A protect code that is the null string (or all blanks) is interpreted as no protect code. The character > is not allowed in a protect code.

REAL A numeric data type that is stored internally in eight bytes using sign-and-magnitude binary representation. One bit is used for the number's sign, 11 bits for a biased exponent (bias = 1023), and 52 bits for a mantissa. On all values except 0, there is an implied "1." preceding the mantissa (this can be thought of as the 53rd bit). The range of REAL numbers is approximately:

−1.797 693 134 862 32 E+308 thru −2.225 073 858 507 2 E−308, 0, and +2.225 073 858 507 2 E−308 thru +1.797 693 134 862 32 E+308.

If a numeric variable is not explicitly declared as an INTEGER, it is a REAL.

record The records referred to in this manual are defined records. Defined records are the smallest unit of storage directly accessible on the mass storage media. The length of a record is determined when a BDAT file is created by a CREATE BDAT statement. All records in a file are the same size.

There is another type of record called a "physical record" which is the unit of storage handled by the mass storage device and the operating system. Physical records contain 256 bytes and are not accessible to the user via standard BASIC statements.

recursive See "recursive".

remote file name A remote (SRM) file name consists of one to 16 characters. HP Series 200/300 remote file names can contain uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

secondary address A device-dependent command sent on HP-IB. It can be interpreted as a secondary address for the extended talker/listener functions or as part of a command sequence. (Also see "device selector".)

selector A numeric quantity used to identify or choose something from a number of possibilities. A selector is ususlly a numeric expression. For example: *device selector* is used to identify a device involved in a I/O operation, and *pen selector* is used to select a pen on a plotter.

soft clip limits These are plotter clipping limits that are defined by the programmer. Lines drawn on a plotting device are drawn only inside the clipping limits.

specifier A string used to identify a method for handling an I/O operation. A specifier is usually a string expression. For example: *mass storage unit specifier* selects the proper drivers for a mass storage unit, and *plotter specifier* chooses the protocol of a plotting device.

SRM The acronym for Shared Resource Management.

SRM controller The HP Series 200/300 computer that controls access to the shared resources of the Shared Resource Management system.

SRM controller's node address An integer in the range 0 through 63 that identifies the SRM controller.

SRM interface The term used to describe the HP 98629A Resource Management Interface resident in an SRM workstation computer (not the interface in the SRM controller).

statement A keyword combined with any additional items that are allowed or required with that keyword. If a statement is placed after a line number and stored, it becomes a program line. If a statement is typed without a line number and executed, it is called a command.

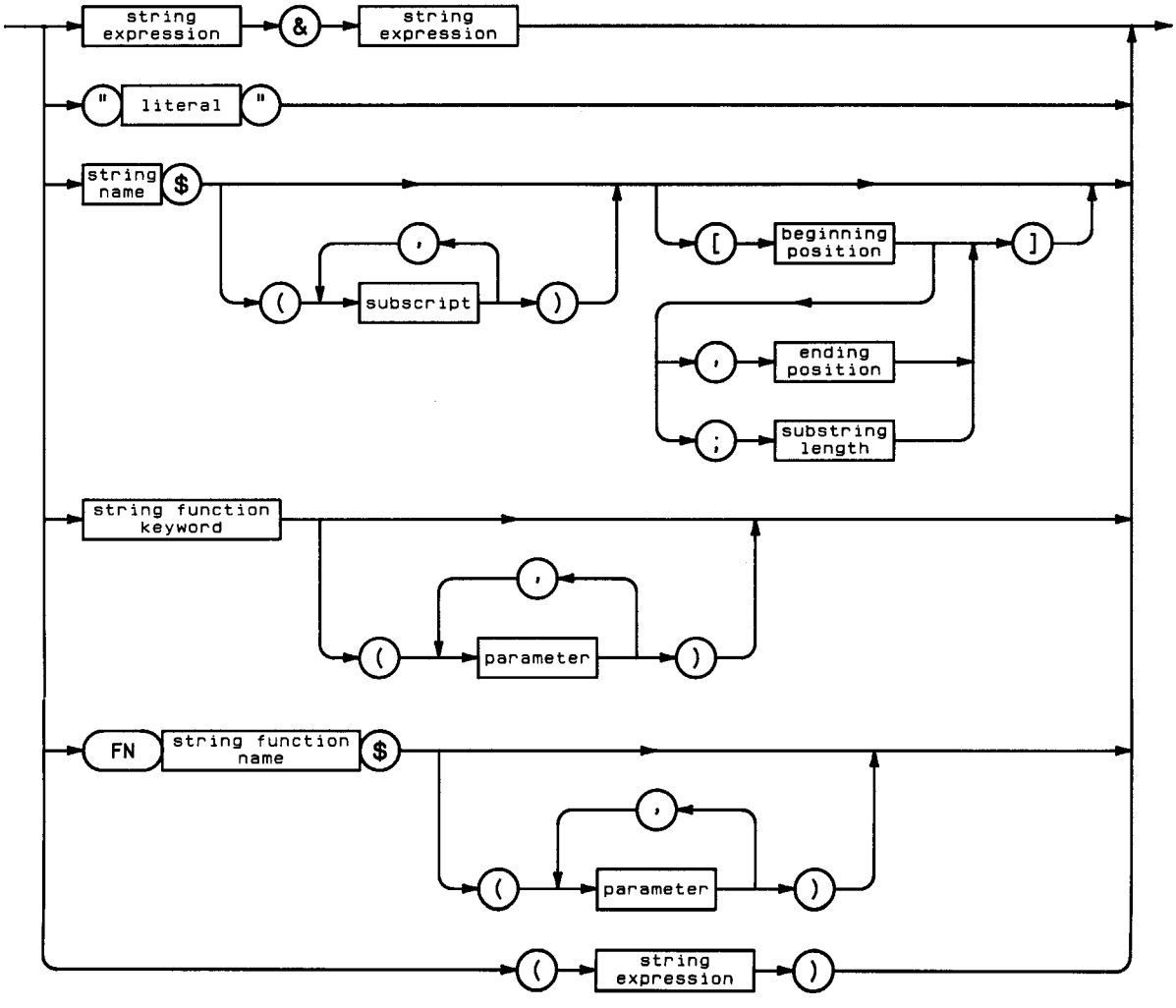
string A data type comprised of a contiguous series of characters. Strings require one byte of memory for each character of declared length, plus a two-byte length header. Characters are stored using an extended ASCII character set. The first character in a string is in position 1. The maximum length of a string is 32 767 characters. The current length of a string can never exceed the dimensioned length.

If a string is not explicitly dimensioned, it is implicitly dimensioned to 18 characters. Each element in an implicitly dimensioned string array is dimensioned to 18 characters.

When a string is empty, it has a current length of zero and is called a "null string". All strings are null strings when they are declared. A null string can be represented as an empty literal (for example: `A$=""`) or as one of three special cases of substring. The substrings that represent the null string are:

1. Beginning position one greater than current length
2. Ending position one less than beginning position
3. Maximum substring length of zero

string expression



Item	Description
literal	A string constant composed of any characters available on the keyboard, including those generated with the ANY CHAR key.
string name	The name of a string variable or the name of a string array from which a string is extracted using subscripts
subscript	A numeric expression used to select an element of an array (see "array")
beginning position	A numeric expression specifying the position of the first character in a substring (see "substring")
ending position	A numeric expression specifying the position of the last character in a substring (see "substring")
substring length	A numeric expression specifying the maximum number of characters to be included in a substring (see "substring")
string function keyword	A keyword that invokes a machine-resident function which returns a string value. String function keywords always end with a dollar sign.
string function name	The name of a user-defined function that returns a string value
parameter	A numeric expression, string expression, or I/O path name that is passed to a function

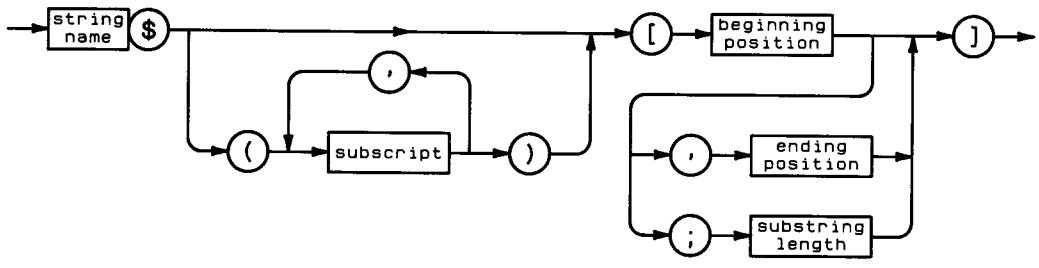
subprogram Can be a CSUB, a SUB subprogram or a user-defined-function subprogram (DEF FN). The first line in a SUB subprogram is a SUB statement. The last line in a SUB subprogram (except for comments) is a SUBEND statement. The first line in a function subprogram is a DEF FN statement. The last line in a function (except for comments) is an FNEND statement. Subprograms must follow the END statement of the main program.

SUB and CSUB subprograms are invoked by CALL. Function subprograms are invoked by an FN function occurring in an expression. A function subprogram returns a value that replaces the occurrence of the FN function when the expression is evaluated. Subprograms may alter the values of parameters passed by reference or variables in COM. It is recommended that you do not let function subprograms alter values in that way.

Invoking a subprogram establishes a new context. The new context remains in existence until the subprogram is properly exited or program execution is stopped. Subprograms can be recursive.

subroutine A program segment accessed by a GOSUB statement and ended with a RETURN statement.

substring



A substring is a contiguous series of characters that comprises all or part of a string. Substrings may be accessed by specifying a beginning position, or a beginning position and an ending position, or a beginning position and a maximum substring length.

The beginning position must be at least one and no greater than the current length plus one. When only the beginning position is specified, the substring includes all characters from that position to the current end of the string.

The ending position must be no less than the beginning position minus one and no greater than the dimensioned length of the string. When both beginning and ending positions are specified, the substring includes all characters from the beginning position to the ending position or current end of the string, whichever is less.

The maximum substring length must be at least zero and no greater than one plus the dimensioned length of the string minus the beginning position. When a beginning position and substring length are specified, the substring starts at the beginning position and includes the number of characters specified by the substring length. If there are not enough characters available, the substring includes only the characters from the beginning position to the current end of the string.

volume A named portion of mass storage media, which may contain several files. Disc drives supported by HP Series 200/300 mass storage operations contain only one volume per disc.

volume name A name used to identify a mass storage volume. The volume name is assigned to the volume at initialization. Volume names consist of one to 16 characters including uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

volume password A "master" password, assigned at initialization, that allows complete access to all files on a mass storage volume. Volume passwords consist of one to 16 characters. All ASCII characters except ">" are allowed. The volume password supercedes all access restrictions placed on files by the PROTECT statement in BASIC or the Pascal Filer's Access command.

Interface Registers

I/O Path Status and Control Registers

For All I/O Path Names:

	Returned Value	Meaning
Status Register 0	0	Invalid I/O path name
	1	I/O path name assigned to a device
	2	I/O path name assigned to a data file
	3	I/O path name assigned to a buffer

I/O Path Names Assigned to a Device:

Status Register 1	Interface select code
Status Register 2	Number of devices
Status Register 3	1st device selector

If assigned to more than one device, the other device selectors are available starting in Status Register 4.

I/O Path Names Assigned to an ASCII File:

Status Register 1	File type = 3
Status Register 2	Device selector of mass storage device
Status Register 3	Number of records
Status Register 4	Bytes per record = 256
Status Register 5	Current record
Status Register 6	Current byte within record

I/O Path Names Assigned to a BDAT File:

Status Register 1	File type = 2
Status Register 2	Device selector of mass storage device
Status Register 3	Number of defined records
Status Register 4	Defined record length (in bytes)
Status Register 5	Current record
Control Register 5	Set current record
Status Register 6	Current byte within record
Control Register 6	Set current byte within record
Status Register 7	EOF record
Control Register 7	Set EOF record
Status Register 8	Byte within EOF record
Control Register 8	Set byte within EOF record

I/O Path Names Assigned to a Buffer:

Status Register 1 Buffer type (1 = named, 2 = unnamed)

Status Register 2 Buffer size in bytes

Status Register 3 Current fill pointer
Control Register 3 Set fill pointer

Status Register 4 Current number of bytes in buffer
Control Register 4 Set number of bytes

Status Register 5 Current empty pointer
Control Register 5 Set empty pointer

Status Register 6 Interface select code of inbound TRANSFER

Status Register 7 Interface select code of outbound TRANSFER

Status Register 8 If non-zero, inbound TRANSFER is continuous
Control Register 8 Cancel continuous mode inbound TRANSFER if zero

Status Register 9 If non-zero, outbound TRANSFER is continuous
Control Register 9 Cancel continuous mode outbound TRANSFER if zero

Status Register 10 Termination status for inbound TRANSFER

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANSFER Active	TRANSFER Aborted	TRANSFER Error	Device Termination	Byte Count	Record Count	Match Character
Value = 0	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 11 Termination status for outbound TRANSFER

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANSFER Active	TRANSFER Aborted	TRANSFER Error	Device Termination	Byte Count	Record Count	0
Value = 0	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 0

Status Register 12 Total number of bytes transferred by last inbound TRANSFER

Status Register 13 Total number of bytes transferred by last outbound TRANSFER

CRT Status and Control Registers

Status Register 0 Current print position (column)
Control Register 0 Set print position (column)

Status Register 1 Current print position (line)
Control Register 1 Set print position (line)

Status Register 2 Insert-character mode
Control Register 2 Set insert character mode if non-0

Status Register 3 Number of lines "above screen"
Control Register 3 Undefined

Status Register 4 Display functions mode
Control Register 4 Set display functions mode if non-0

Status Register 5 Returns the CRT alpha color value set (or default). This does not reflect changes due to printing `CHR$(x)`, where $136 \leq x \leq 143$.

Control Register 5 Set default alpha color:
For Alpha Displays:

Value	Result
< 128	Error
128-135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144-159	Ignored
> 159	Error

For Bit-Mapped Displays:
Values 0 thru 255 which correspond to the graphics pens.

For multi-plane bit-mapped displays, the graphics pen to be used for alpha.

`CONTROL CRT,5;`*n* sets the values of the CRT registers 15, 16, and 17, but the converse is not true. That is, `STATUS CRT,5` may not accurately reflect the CRT state if `CONTROL 15, 16`, and/or `17` have been executed.

Status Register 6 ALPHA ON flag
Control Register 6 Undefined

Status Register 7 GRAPHICS ON flag
Control Register 7 Undefined

CRT Status and Control Registers (cont.)

Status Register 8
Control Register 8
Display line position (column)
Set display line position (column)

Status Register 9
Control Register 9
Screenwidth (number of characters)
Undefined

Status Register 10
Control Register 10
Cursor-enable flag
Cursor-enable: 0 = cursor invisible
non-0 = cursor visible

Status Register 11
Control Register 11
CRT character mapping flag
Disable CRT character mapping if non-0

Status Register 12
Control Register 12
Key labels display mode
Set key labels display mode: 0 = typing-aid key labels displayed unless program is in the RUN state
1 = key labels always off
2 = key labels displayed at all times

Status Register 13
Control Register 13
CRT height
CRT height; number of lines in Alpha display must be greater than 8.

Status Register 14
Control Register 14
Display replacement
Display replacement
0–0
1–source AND old
2–source AND NOT old
3–source;default
4–NOT source AND old
5–old
6–source EXOR old
7–source OR old
8–source NOR old
9–source EXNOR old
10–NOT old
11–source OR NOT old
12–NOT source
13–NOT source OR old
14–source NAND old
15–1

CRT Status and Control Registers (cont.)

Status Register 15 Return the value set (or the default) for the color in the PRINT/DISP area. This does not reflect changes due to printing CHR$(*x*), where $136 \leq x \leq 143$.

Control Register 15 Set PRINT/DISP color. Similar to CRT control register 5 but specific to CRT PRINT/DISP areas; that is, it does not affect the areas covered by CRT registers 16 and 17.

Status Register 16 Return the value set (or the default) for the softkey label color.

Control Register 16 Set key labels color. Similar to CRT control register 5 but only affects the softkey labels. Does not affect the areas covered by CRT registers 15 and 17.

Status Register 17 Return the value set (or the default) for the color of the "non-enhance" area. This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen.

Control Register 17 Set "non-enhance" color. This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen. Similar to CRT control register 5 but does not affect the areas covered by CRT control registers 15 and 16.

Status Register 18 Read the alpha write-enable mask.
Control Register 18 Set alpha write-enable mask to a bit pattern.
Status Register 19 Return number of planes in alpha CRT.
Control Register 19 Undefined.

Status Register 20 Read the alpha display-enable mask.
Control Register 20 Set alpha display-enable mask to a bit pattern.

Status Register 21 Return compatibility mode (0 or 1).

Control Register 21 Switch between the CRT compatibility mode (value $\neq 0$) and the native bit-mapped mode (value $= 0$). That is, switch both alpha and graphics to non-bit-mapped display (if value $\neq 0$) or bit-mapped display (if value $= 0$). It effectively initializes the alpha display and executes a GINIT and a PLOTTER IS CRT,"INTERNAL".

Keyboard Status and Control Registers

Status Register 0 CAPS LOCK flag
Control Register 0 Set CAPS LOCK if non-0

Status Register 1 PRINTALL flag
Control Register 1 Set PRINTALL if non-0

Status Register 2 Function key menu.
Control Register 2 Function key menu. 0 = System menu
1-3 = User menu 1 thru 3

Status Register 3 Undefined

Control Register 3 Set auto-repeat interval. If 1 thru 255, repeat rate in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at power-on or SCRATCH A is 80 ms.)

Status Register 4 Undefined

Control Register 4 Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at power-on or SCRATCH A is 700 ms.)

Status Register 5 KBD$ buffer overflow register. 1 = overflow.
Register is reset when read.
Control Register 5 Undefined

Status Register 6 Typing aid expansion overflow register.
1 = overflow. Register is reset when read.
Control Register 6 Undefined

Status Register 7 **Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	INITIALIZE Timeout Interrupt Disabled	Reserved For Future Use	Reserved For Future Use	RESET Key Interrupt Disabled	Keyboard and Knob Interrupt Disabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 7 (Set bit to disable) **Interrupt Disable Mask**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			INITIALIZE Timeout	Reserved For Future Use	Reserved For Future Use	RESET Key	Keyboard and Knob
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Keyboard Status and Control Registers (cont.)

Status Register 8 Keyboard language jumper

0 – US ASCII	7 – United Kingdom	13 – Swiss German
1 - French	8 – Canadian French	14 – Latin(Spanish)
2 – German	9 – Swiss French	15 – Danish
3 – Swedish	10 – Italian	16 – Finnish
4 – Spanish	11 – Belgian	17 – Norwegian
5 – Katakana	12 – Dutch	18 – Swiss French*
6 – Canadian English		19 – Swiss German*

Control Register 8 Undefined

Status Register 9

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Internal Use	Internal Use	1 = HP46020A Keyboard 0 = Other Keyboard	1 = No Keyboard 0 = Keyboard Present	1 = n-Key Rollover 0 = 2 or less Key Rollover	0	0	1 = HP98203A Keyboard 0 = Other Keyboard
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 9 Undefined

Status Register 10 **State at Last Knob Interrupt**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	CTRL Key Pressed	SHIFT Key Pressed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Keyboard Status and Control Registers (cont.)

Control Register 10	Undefined
Status Register 11	0 = horizontal-pulse mode; 1 = all-pulse mode
Control Register 11	(default is 0 without KNB2_0 loaded, 1 with KNB2_0 loaded). Refer to the Knob section in Chapter 15 of the *BASIC Programming Techniques* manual for more information.
Status Register 12	"Pseudo-EOI forCTRL-E" flag
Control Register 12	Enable pseudo-EOI for CTRL-E if non-0
Status Register 13	Katakana flag
Control Register 13	Set Katakana if non-0
Status Register 14	Function keys on HP 46020A software key numbers shifted.
Control Register 14	Function keys on HP 46020A software key numbers shifted. 0 = [f1] is Key 1; default 1 = [f1] is Key 0
Status Register 15	Return keyboard compatibility mode (0→off, 1→on).
Control Register 15	Turns Model 236 keyboard compatibility mode on (≠0) and off (=0). (See the chapter "Porting to Series 300" in the *Programming Techniques* manual.)

HP-IB Status and Control Registers

Status Register 0 Card identification = 1

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrrupts Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 **Serial Poll Response Byte**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	SRQ 1 = I did it 0 = I didn't	Device Dependent Status					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 2 **Busy Bits**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Reserved For Future Use	Handshake In Progress	Interrupts Enabled	TRANSFER In Progress
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 **Parallel Poll Response Byte**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1 = True	DIO7 1 = True	DIO6 1 = True	DIO5 1 = True	DIO4 1 = True	DIO3 1 = True	DIO2 1 = True	DIO1 1 = True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

HP-IB Status and Control Registers (cont.)

Status Register 3

Controller Status and Address

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 3

Set My Addrress

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 4

Interrupt Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 4 Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

HP-IB Status and Control Registers (cont.)

Status Register 5 — Interrupt Enable Mask

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 5 — Parallel Poll Response Mask

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Uncon- figure	Logic Sense	Data Bit Used For Response		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

HP-IB Status and Control Registers (cont.)

Status Register 6

Interface Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Least-significant bit of last address recognized

Status Register 7

Bus Control and Data Lines

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Only if addressed to TALK, else not valid.
** Only if Active Controller, else not valid.

HP-IB Status and Control Registers (cont.)

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

RS-232 Status and Control Registers

Status Register 0 — **Card Identification**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 = Future Use Jumper Installed	0	0	0	0	0	1	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 0 Reset interface if non-zero

Status Register 1 — **Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Send break if non-zero

Status Register 2

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Progress	Interrupts Enabled	TRANSFER In Progress
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 Undefined

Status Register 3 Baud Rate

Control Register 3 Set Baud Rate

RS-232 Status and Control Registers (cont.)

Status Register 4

Character Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved for Future Use		00 = Odd Parity 01 = Even Parity 10 = Parity Bit "1" 11 = Parity Bit "0"		Parity Enabled	0 = One Stop Bit 1 = Two Stop Bits*	Character Length (add this value to 5)	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*1.5 stop bit if character length is 5.

Control Register 4

Character Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used		00 = Odd parity 01 = Even Parity 10 = Parity Bit "1" 11 = Parity Bit "0"		1 = Enable parity	0 = One Stop Bit 1 = Two Stop Bits*	Character Length (add this value to 5)	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*1.5 stop bits if character length is 5.

Status Register 5

Modem Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	Loop Back Mode	Secondary Request To Send	Data Rate Select	Request To Send	Data Terminal Ready
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 5

Modem Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			1 = Set Loopback Mode	1 = Set Secondary Request To Send	1 = Set Data Rate Select	RTS* 1 = Set 0 = Handshake	DTR** 1 = Set 0 = Handshake
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* 0 = Set only during an OUTPUT statement.

** 0 = Set only during an OUTPUT or ENTER statement.

RS-232 Status and Control Registers (cont.)

Status Register 6 Data In (8 bits)

Control Register 6 Data Out (8 bits)

Status Register 7 — **Optional Circuits**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Optional Driver 3	Optional Driver 4	Optional Receiver 2	Optional Receiver 3
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 7 — **Optional Circuits**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Optional Driver 3	Optional Driver 4	Not Used	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 8 — **Interrupt Enable Mask**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 9 — **Interrupt Cause**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	11 = Receiver Line Status 10 = Receiver Buffer Full 01 = Transmitter Holding Register Empty 00 = Modem Status Change		0 = UART Requesting Interrupt
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

RS-232 Status and Control Registers (cont.)

Status Register 10

UART Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Transmit Shift Register Empty	Transmit Holding Register Empty	Break Received	Framing Error	Parity Error	Overrun Error	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 11

Modem Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect	Ring Indicator	Data Set Ready	Clear To Send	Change In Carrier Detect	Ring Indicator Changed To False	Change In Data Set Ready	Change In Clear To Send
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 12

Modem Handshake Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable*	0	Data Set Ready Disable**	Clear to Send Disable***	0	0	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* 0 = Wait for Carrier Detect on Enter Operations; 1 = Don't wait
** 0 = Wait for Data Set Ready on Enter and Output Operations; 1 = Don't wait
*** 0 = Wait for Clear to Send on Output Operations; 1 = Don't wait

RS-232 Status and Control Registers (cont.)

Control Register 12 — Modem Handshake Control

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable*	Not Used	Data Set Ready Disable**	Clear to Send Disable***	Not Used	Not Used	Not Used	Not Used
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* 0 = Wait for Carrier Detect on Enter Operations; 1 = Don't wait
** 0 = Wait for Data Set Ready on Enter and Output Operations; 1 = Don't wait
*** 0 = Wait for Clear to Send on Output Operations; 1 = Don't wait

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 13

Read the SCRATCH A default baud rate. The interface is set to this default when SCRATCH A occurs.

Control Register 13

Set the SCRATCH A default baud rate. The values accepted are the same as for CONTROL/STATUS register 3. The default power-up value is 9600 baud.

Status Register 14

Read the SCRATCH A default character format. The interface is set to this default when SCRATCH A occurs.

Control Register 14

Set the "SCRATCH A" default character format. The values accepted are the same as for CONTROL/STATUS register 4. The default values are: 8 bits/character, 1 stop bit, no parity.

GPIO Status and Control Registers

Status Register 0 Card identification = 3

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	Interrupt Requested	Hardware Interrupt Level Switches		Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Set PCTL if non-zero

Status Register 2

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Process	Interrupts Are Enabled	TRANSFER In Progress
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 **Peripheral Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					PSTS Error (1 = Report; 0 = Ignore)	Set CTL1 (1 = Low; 0 = High)	Set CTL0 (1 = Low; 0 = High)
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 Data In (16 bits)

Control Register 3 Data Out (16 bits)

Status Register 4 Interface is Ready for a subsequent data transfer; 1 = Ready, 0 = Busy.

GPIO Status and Control Registers (cont.)

Status Register 5

Peripheral Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS OK	EIR Line Low	STI1 Line Low	STI0 Line Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

BCD Status and Control Registers

Status Register 0 Card Identification = 4.
Control Register 0 Reset Interface (if non-zero value sent).

Status Register 1 **Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Reset driver pointer (if non-zero value sent).

Status Register 2 **Busy Bit**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Progress	Interrupts Enabled	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 Request data by Setting CTLA and CTLB (if a non-zero value is sent); this operation also clears an Interrupt Request (clears bit 6 of Status Register 1).

Status Register 3 Binary Mode: 1 if the interface is currently operating in Binary mode, and 0 if in BCD mode.

Control Register 3 Set Binary Mode: set Binary Mode if non-zero value sent, and BCD Mode if zero sent.

BCD Status and Control Registers (cont.)

Status Register 4

Switch and Line States

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OF Switch Is ON	DATA Switch Is ON	SGN1 Switch Is ON	SGN2 Switch Is ON	OVLD Switch Is ON	SGN1 Input Is True	SGN2 Input Is True	OVLD Input Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 4

Data Out Lines

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set DO-7 True	Set DO-6 True	Set DO-5 True	Set DO-4 True	Set DO-3 True	Set DO-2 True	Set DO-1 True	Set DO-0 True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 5

BCD Digits D1 and D2

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI1-8 Is True	DI1-4 Is True	DI1-2 Is True	DI1-1 Is True	DI2-8 Is True	DI2-4 Is True	DI2-2 Is True	DI2-1 Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 6

BCD Digits D3 and D4

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI3-8 Is True	DI3-4 Is True	DI3-2 Is True	DI3-1 Is True	DI4-8 Is True	DI4-4 Is True	DI4-2 Is True	DI4-1 Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

BCD Status and Control Registers (cont.)

Status Register 7

BCD Digits D5 and D6

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI5-8 Is True	DI5-4 Is True	DI5-2 Is True	DI5-1 Is True	DI6-8 Is True	DI6-4 Is True	DI6-2 Is True	DI6-1 Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 8

BCD Digits D7 and D8

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7-8 Is True	DI7-4 Is True	DI7-2 Is True	DI7-1 Is True	DI8-8 Is True	DI8-4 Is True	DI8-2 Is True	DI8-1 Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 9

BCD Digits D9 and D10

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI9-8 Is True	DI9-4 Is True	DI9-2 Is True	DI9-1 Is True	DI10-8 Is True	DI10-4 Is True	DI10-2 Is True	DI10-1 Is True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Data Communications Status and Control Registers

General Notes: Control registers accept values in the range of zero through 255. Some registers require specified values, as indicated. Illegal values or values less than zero or greater than 255, cause ERROR 327.

Reset value, shown for various Control Registers, is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

Status 0 Card Identification
Value returned: 52 (if 180 is returned, check select code switch cluster and make sure switch R is ON).

Control 0 Card Reset
Any value, 1 thru 255, resets the card. Immediate execution. Data in queues is destroyed.

Status 1 Hardware Interrupt Status (not used in most applications)
1 = Enabled 0 = Disabled

Status 2 Datacomm Activity
0 = No activity pending on this select code.
Bit 0 set: ENTER in process.
Bit 1 set: OUTPUT in process.
Bit 2 set: inbound TRANSFER in process
Bit 3 set: outbound TRANSFER in process

Status 3 Current Protocol Identification:
1 = Async, 2 = Data Link Protocol

Control 3 Protocol to be used after next card reset `(CONTROL Sc,0;1)`
1 = Async Protocol 2 = Data Link Protocol
This register overrides default switch configuration.

Status 4 Cause of ON INTR program branch.

Bit	Function: Async Protocol	Function: Data Link Protocol
0	Data and/or Control Block available	Data Block Available
1	Prompt received	Space available for a new transmission block
2	Framing and/or parity error	Receive or transmit error
3	Modem line change	Modem line change
4	No Activity timeout (forces a disconnect)	No Activity timeout (forces a disconnect)
5	Lost carrier or connection timeout (forces a disconnect)	Lost carrier or connection timeout (forces a disconnect)
6	End-of-line received	Not Used
7	Break received	Not Used

Contents of this register are cleared when a STATUS statement is executed to it.

Datacomm Status and Control Registers (cont.)

Status 5 Inbound queue status

Value	Interpretation
0	Queue is empty
1	Queue contains data but no control blocks
2	Queue contains one or more control blocks but no data
3	Queue contains both data and one or more control blocks

Control 5 Terminate Transmission

`OUTPUT S,5;0` is equivalent to `OUTPUT S;END`

Data Link: Sends previous data as a single block with an ETX terminator, then idles the line with an EOT.

Async: Tells card to turn half-duplex line around. Does nothing when line is full-duplex. The next data OUTPUT automatically regains control of the line by raising the RTS (request-to-send) modem line.

Status 6 Break status: 1 = BREAK transmission pending, 0 = no BREAK pending.

Control 6 Send Break; causes a Break to be sent as follows:

Data Link Protocol: Send Reverse Interrupt (RVI) reply to inbound block, or CN character instead of data in next outbound block.

Async Protocol: Transmit Break. Length is defined by Control Register 39.

Note that the value sent to the register is arbitrary.

Status 7 Modem receiver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Data Mode (Data Set Ready) line
Bit 1: Receive ready (Data Carrier Detect line)
Bit 2: Clear-to-send (CTS) line
Bit 3: Incoming call (Ring Indicator line)
Bit 4: Depends on cable option or adapter used

Status 8 Returns modem driver line states.

Control 8 Sets modem driver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Request-to-send (RS or RTS) line — 1 = line set (active)
Bit 1: Data Terminal Ready (DTR) line — 0 = line clear (inactive)
Bit 2: Driver 1: Data Rate Select
Bit 3: Driver 2: Depends on cable option or adapter used
Bit 4: Driver 3: Depends on cable option or adapter used
Bit 5: Driver 4: Depends on cable option or adapter used
Bits 6,7: Not used

Reset value = 0 prior to connect. Post-connect value is handshake dependent.

Note that RTS line cannot be altered (except by OUTPUT or OUTPUT...END) for half-duplex modem connections.

Datacomm Status and Control Registers (cont.)

Status 9 Returns control block TYPE if last ENTER terminated on a control block. See Status Register 10 for values.

Status 10 Returns control block MODE if last ENTER terminated on a control block.

Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (Channel A)
251	1[1]	Framing error in the following character
251	2[1]	Parity error in the following character
251	3[1]	Parity and framing errors in the following character
252	1	End-of-line terminator detected
253	1	Prompt received from remote
0	0	No Control Block encountered

Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character
254	2	Preceding block terminated by ETX character
253[2]	—	(see following table for Mode interpretation)
0	0	No Control Block encountered.

Mode Bit(s)	Interpretation
0	1 = Transparent data in following block
	0 = Normal data in following block
2,1	00 = Device select
	01 = Group select
	10 = Line select
3	1 = Command channel
	2 = Data channel

Status 11 Returns available outbound queue space (in bytes), provided there is sufficient space for at least three control blocks. If not, value is zero.

Control 12 Datacomm Line connection control

Value	Action
0	Disconnect
1	Begin connection sequence
2	Begin autodial sequence

1 Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (_) character.

2 This type is used primarity in specialized applications.

Datacomm Status and Control Registers (cont.)

Status 12 Datacomm Line connection status

Value	Interpretation
0	Disconnected
1	Attempting Connection
2	Dialing
3	Connected[1]
4	Suspended
5	Currently receiving data (Data Link only)
6	Currently transmitting data (Data Link only)

Note

When the datacomm line is suspended, CLEAR, ABORT, or RESET must be executed before the line can be reconnected.

Reset value – 0 if $\bar{R}$ on interface select code switch cluster is ON (1).

Status 13 Returns current ON INTR mask

Control 13 Sets ON INTR mask[2]

Data Link Protocol:

Bit	Value	Enables interrupt when:
0	1	A full block is available in receive queue
1	2	Transmit queue is empty
2	4	Receive or transmit error detected
3	8	A modem line changed
4	16[3]	No Activity timeout forced a disconnection
5	32[3]	Lost Carrier or Connection timeout caused a disconnection

Async Protocol:

Bit	Value	Enables interrupt when:
0	1	Data or control block available in receive queue
1	2	Prompt received from remote device
2	4	Framing or parity error detected in incoming data
3	8	A modem line changed
4	16[3]	No Activity timeout forced a disconnection
5	32[3]	Lost Carrier or Connection timeout caused a disconnection
6	64	End-of-line received
7	128	Break received

Reset value = 0

1 When using Data Link: Connected - datacomm idle

2 If a CONTROL statement is used to access this register, the control block is placed in the outbound queue. If the ENABLE INTR... statement is used with a mask, the mask value is placed directly in the control register, bypassing any queue delays.

3 If bits 4 and 5 are not set, the corresponding errors can be trapped by using an ON ERROR statement.

Datacomm Status and Control Registers (cont.)

Status 14 Returns current Control Block mask.

Control 14 Sets Control Block mask. Control block information is queued sequentially with incoming data as follows:

Bit	Value	Async Control Block Passed	Data Link Control Block Passed
0	1	Prompt position	Transparent/Normal Mode[1]
1	2	End-of-line position	ETX Block Terminator[2]
2	4	Framing and/or Parity error[3]	ETB Block Terminator[2]
3	8	Break received	
Reset Value:		**0** (Control Blocks disabled)	**6** (ETX/ETB Enabled)

Bits 4, 5, 6, and 7 are not used.

Status 15 Returns current modem line interrupt mask.

Control 15 Sets modem line interrupt mask. Enables an interrupt to ON INTR when Bit 3 of Control Register 13 is set as follows:

Bit	Value	Modem Line to Cause Interrupt
0	1	Data Mode (Data Set Ready)
1	2	Receive Ready (Data Carrier Detect)
2	4	Clear-to-send
3	8	OCR1, Incoming Call (Ring Indicator)
4	16	OCR2, Cable or adapter dependent

Reset Value = 0

Note that bit functions are the same as for STATUS register 7. Functions shown are for male connector cable option for modem connections.

Status 16 Returns current connection timeout limit.

Control 16 Sets Attempted Connection timeout limit.
Acceptable values: 1 thru 255 seconds. 0 = timeout disabled.
Reset Value = 25 seconds

Status 17 Returns current No Activity timeout limit.

Control 17 Sets No Activity timeout limit.
Acceptable values: 1 thru 255 minutes. 0 = timeout disabled.
Reset Value = 10 minutes (disabled if Async, non-modem handshake).

Status 18 Returns current Lost Carrier timeout limit.

Control 18 Sets Lost Carrier timeout limit in units of 10 ms.
Acceptable values: 1 thru 255. 0 = timeout disabled.
Reset Value = 40 (400 milliseconds)

1 Transparent/Normal format identification control block occurs at the BEGINNING of a given block of data in the receive queue.
2 ETX and ETB Block Termination identification control blocks occur at the END of a given block of data in the receive queue.
3 This control block precedes each character containing a parity or framing error.

Datacomm Status and Control Registers (cont.)

Status 19 Returns current Transmit timeout limit.

Control 19 Sets Transmit timeout limit (loss of clock or CTS not returned by modem when transmission is attempted).
Acceptable values: 1 thru 255.0 = timeout disabled.
Reset Value = 10 seconds

Status 20 Returns current transmission speed (baud rate). See table for values.

Control 20 Sets transmission speed (baud rate) as follows:

Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock	8	600
*1	50	9	1200
*2	75	10	1800
*3	110	11	2400
*4	134.5	12	3600
*5	150	13	4800
*6	200	14	9600
7	300	15	19200

* Async only. These values cannot be used with Data Link. These values set transmit speed ONLY for Async; transmit AND receive speed for Data Link. Default value is defined by the interface card configuration switches.

Status 21 Protocol dependent. Returns receive speed (Async) or GID address (Data Link) as specified by Control Register 21.

Control 21 Protocol dependent. Functions are as follows:

Data Link: Sets Group IDentifier (GID) for terminal. Values 0 thru 26 correspond to identifiers @, A, B,...Y, Z, respectively. Other values cause an error. Default value is 1 ("A").

Async: Sets datacomm receiver speed (baud rate). Values and defaults are the same as for Control Register 20.

Status 22 Protocol dependent. Returns DID (Data Link) or protocol handshake type (Async) as specified by Control Register 22.

Control 22 Protocol dependent. Functions are as follows:

Data Link: Sets Device IDentifier (DID) for terminal. Values are the same as for Control Register 21. Default is determined by interface card configuration switches.

Async: Defines protocol handshake type that is to be used.

Value	Handshake type
0	Protocol handshake disabled
1	ENQ/ACK with desktop computer as the host
2	ENQ/ACK, desktop computer as a terminal
3	DC1/DC3, desktop computer as host
4	DC1/DC3, desktop computer as a terminal
5	DC1/DC3, desktop computer as both host and terminal

Datacomm Status and Control Registers (cont.)

Status 23 Returns current hardware handshake type.

Control 23 Sets hardware handshake type as follows:

0 = Handshake OFF, non-modem connection.
1 = FULL-DUPLEX modem connection.
2 = HALF-DUPLEX modem connection.
3 = Handshake ON, non-modem connection.
Reset Value is determined by interface configuration switches.

Status 24 Protocol dependent. Returns value set by preceding CONTROL statement to Control Register 24.

Control 24 Protocol dependent. Functions as follows:
Data Link protocol: Set outbound block size limit.

Value	Block size	Value	Block size
0	512 bytes	4	8 bytes
1	2 bytes	.	.
2	4 bytes	.	.
3	6 bytes	255	510 bytes

Reset outbound block size limit = 512 bytes

Async Protocol: Set mask for control characters included in receive data message queue.
Bit set: transfer character(s).
Bit cleared: delete character(s).

Bit set	Value	Character(s) passed to receive queue
0	1	Handshake characters (ENQ, ACK, DC1, DC3)
1	2	Inbound End-of-line character(s)
2	4	Inbound Prompt character(s)
3	8	NUL (CHR$(0))
4	16	DEL (CHR$(127))
5	32	CHR$(255)
6	64	Change parity/framing errors to underscores (_) if bit is set.
7	128	Not used

Reset value = 127 (bits 0 thru 6 set)

Status 25 Returns number of received errors since power up or reset.

Note

Control Registers 26 through 35, Status Registers 27 through 35, and Control and Status Registers 37 and 39 are used for ASYNC protocol ONLY. They are not available during Data Link operation.

Datacomm Status and Control Registers (cont.)

Status 26 Protocol dependent
Data Link protocol: Returns number of transmit errors (NAKs received) since last interface reset.
Async protocol: Returns first protocol handshake character (ACK or DC1).

Control 26 Sets first protocol handshake character as follows:
(Async only) 6 = ACK, 17 = DC1. Other values used for special applications only. Use ACK when Control Register 22 is set to 1 or 2. Use DC1 when Control Register 22 is set to 3, 4, or 5.
Reset value = 17 (DC1)

Status 27 Returns second protocol handshake character.
(Async only)

Control 27 Sets second protocol handshake character as follows:
(Async only) 5 = ENQ, 19 = DC3. Other values used for special applications only. Use ENQ when Control Register 22 is set to 1 or 2. Use DC3 when Control Register 22 is set to 3, 4, or 5.
Reset value = 19 (DC3)

Status 28 Returns number of characters in inbound end-of-line delimiter sequence.
(Async only)

Control 28 Sets number of characters in end-of-line delimiter sequence
(Async only) Acceptable values are 0 (no EOL delimiter), 1, or 2.
Reset Value = 2

Status 29 Returns first end-of-line character.
(Async only)

Control 29 Sets first end-of-line character.
(Async only) **Reset Value = 13** (carriage return)

Status 30 Returns second end-of-line character.
(Async only)

Control 30 Sets second end-of-line character.
(Async only) **Reset Value = 10** (line feed)

Status 31 Returns number of characters in Prompt sequence.
(Async only)

Control 31 Sets number of characters in Prompt sequence.
(Async only) Acceptable values are 0 (Prompt disabled), 1 or 2.
Reset Value = 1

Status 32 Returns first character in Prompt sequence.
(Async only)

Control 32 Sets first character in Prompt sequence.
(Async only) **Reset Value = 17** (DC1)

Status 33 Returns second character in Prompt sequence.
(Async only)

Control 33 Sets second character in Prompt sequence.
(Async only) **Reset Value = 0** (null)

Datacomm Status and Control Registers (cont.)

Status 34 Returns the number of bits per character.
(Async only)
Control 34 Sets the number of bits per character as follows:
(Async only) 0 = 5 bits/character 2 = 7 bits/character
1 = 6 bits/character 3 = 8 bits/character)
When 8 bits/char, parity must be NONE, ODD, or EVEN.
Reset Value is determined by interface card default switches.

Status 35 Returns the number of stop bits per character.
(Async only)
Control 35 Sets the number of stop bits per character as follows:
(Async only) 0 = 1 stop bit 1 = 1.5 stop bits 2 = 2 stop bits
Reset Value: 2 stop bits if 150 baud or less, otherwise 1 stop bit.
Reset Value is determined by interface configuration switch settings.

Status 36 Returns current Parity setting.
Control 36 Sets Parity for transmitting and receiving as follows:

Data Link Protocol: 0 = NO Parity; Network host is HP 1000 Computer.
1 = ODD Parity; Network host is HP 3000 Computer.
Reset Value = 0

Async Protocol : 0 = NONE; no parity bit is included with any characters.
1 = ODD; Parity bit SET if there is an EVEN number of "1"s in the character body.
2 = EVEN; Parity bit OFF if there is an ODD number of "1"s in the character body.
3 = "0"; Parity bit is always ZERO, but parity is not checked.
4 = "1"; Parity bit is always SET, but parity is not checked.

Default is determined by interface configuration switches. If 8 bits per character, parity must be NONE, ODD, or EVEN.

Status 37 Returns inter-character time gap in character times.
(Async only)
Control 37 Sets inter-character time gap in character times.
(Async only) Acceptable values: 1 thru 255 character times.
0 = No gap between characters.
Reset Value = 0

Status 38 Returns Transmit queue status.
If returned value = 1, queue is empty, and there are no pending transmissions.

Status 39 Returns current Break time (in character times).
(Async only)
Control 39 Sets Break time in character times.
(Async only) Acceptable values are: 2 thru 255.
Reset Value = 4.

Powerfail Status and Control Registers

Status Register 0 Card Identification is always 5.

Control Register 0 Shut Down. Any non-zero value written to this register will turn off both battery and ac-line power to the computer, which conserves battery power after the service routine has finished responding to the powerfail. If ac-line power is on when this statement is executed, the computer will be turned back on in the normal powerup sequence.

Status Register 1 **Powerfail Interrupt Cause**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					One Second Left	Power Is Back	Power Has Failed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Undefined

Status Register 2 Interrupt Mask has bit definitions identical to the preceding register.

Control Register 2 Undefined

Status Register 3 **Powerfail Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Failed Self Test	Not Used			One Second Left	Currently Using Battery	Ac Is Down	In the Powerfail State
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 3 Undefined.

Powerfail Status and Control Registers (cont.)

Status Register 4	Overheat Protection Timer contains the amount of battery time used during this Powerfail State (in tens of milliseconds). For every second the power is down, it must be back for two seconds to ensure adequate cooling for the machine. Thus, the value of this register bounds the maximum amount of time that can be obtained from the battery, even though 60 seconds may have been specified as the protection time (CONTROL Register 6).
Control Register 4	Undefined.
Status Register 5	Power Back Timer contains the time elapsed since power was restored after the last powerfail (in tens of milliseconds).
Control Register 5	Power Back Delay. The value of this register determines the amount of time (in tens of milliseconds) that the computer will delay, after power is back, before leaving the powerfail state (i.e., before generating a "Power Is Back" interrupt). The power-on default value is 50 (500 milliseconds).
Status Register 6	Powerfail Timer contains the time elapsed since the last powerfail (in tens of milliseconds).
Control Register 6	Protection Time. The value of register determines the maximum amount of time (in tens of milliseconds) that the computer is to have battery backup. Power-on default is 6000 (60 seconds).
Status Register 7	Undefined.
Control Register 7	Powerfail Delay Timer. The contents of this register determine the amount of time (in tens of milliseconds) that the Powerfail-Protection Interface will wait, after a powerfail, before generating a "Power Has Failed" interrupt. Power-on default is 10 (100 milliseconds).
Status Registers 8 thru 71	Continuous-Memory Registers contain the 64 bytes of data written by the last CONTROL statement directed to these registers.
Control Registers 8 thru 71	Continuous-Memory Registers. These sixty-four, single-byte registers can be filled with any desired data, one byte (ASCII character) per register.

EPROM Programmer Status and Control Registers

Status Register 0 **ID Register**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	1	1	0	1	1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register contains a value of 27 (decimal) which is the ID of an EPROM Programmer card.

Control Register 0

Interface Reset
Writing any non-zero value into this register resets the card; writing a value of zero causes no action.

Status Register 1

Read Program Time
A value of 0 indicates that the program time is 52.5 milliseconds for each 16-bit word (default); a non-zero value indicates that the program time is 13.1 milliseconds.

Control Register 1

Set Program Time
Writing a value of 0 into this register sets the program time to 52.5 milliseconds for each 16-bit word; any non-zero value sets program time to 13.1 milliseconds.

Status Register 2

Read Target Address
This register contains the offset address (relative to the card's base address) at which the next word of data will be read (via Status Register 3) or written (via Control Register 3). The default address is 0, which is the address of the first byte on the card.

Control Register 2

Set Target Address
Writing to this register sets the offset address at which the next word of data will be read (via Status Register 3) or written (via Control Register 3). The target address must always be an *even* number.

Status Register 3

Read Word at Target Address
This register contains the 16-bit word at the current target address.

Control Register 3

Write Word at Target Address
Writing a data word to this register programs a 16-bit word at the current target address. The target address must be set (via Control register 2) before every word is written. Automatic verification is also performed after the word is programmed.

Status Register 4

Current Memory Card Capacity (in bytes)
This register contains the current capacity of a *fully loaded* card in bytes; it also indirectly indicates which type of EPROM devices are being used on the card. If 262 144 is returned, then 27128 EPROMs are being used; if 131 072 is returned, then 2764 devices are being used. A 0 is returned if the programmer card is not currently connected to any EPROM memory card.

Control Register 4

Undefined.

Status Register 5

Number of Contiguous, Erased Bytes
Reading this register causes the system to begin counting the number of subsequent bytes, beginning at the current target address, that are erased (or are empty sockets). The counting is stopped when a programmed byte (i.e., one containing at least one logical 0) is found or when the end of the card is reached. If the byte at the current target address is not FF, then a count of 0 is returned. Error 84 is reported if the programmer card is not currently connected to any EPROM card.

Control Register 5

Undefined.

Status Register 6

Base Address of EPROM Memory Card
This register contains the (absolute) base address of the EPROM memory card to which the programmer card is currently connected; this base address is also the absolute address of the first word on the card. Error 84 is reported if the programmer card is not currently connected to any EPROM memory card.

Control Register 6

Undefined.

Parity, Cache and Float Status and Control Registers (Pseudo Select Code 32)

Status Register 0	Parity checking 0 = off, 1 = on
Control Register 0	Sets parity checking 0 = off, 1 = on
Status Register 1	Cache 0 = off, 1 = on
Control Register 1	Sets cache 0 = off, 1 = on
Status Register 2	HP 98635 floating-point math card/MC68881 floating-point math co-processor 0 = off, 1 = on
Control Register 2	Sets HP 98635 floating-point math card/MC68881 floating-point math co-processor 0 = off, 1 = on
Status Register 3	Cache 0 = off, 1 = on
Control Register 3	Sets cache 0 = off, non-0 = on

Summary of SRM Status Registers

Status Register 0 Card Identification

52 if the Remote Control switch (R) is set to 0 (closed); 180 if switch is set to 1 (open).

Status Register 1 Interface Interrupts

1 = interrupts enabled; 0 = interrupts disabled.

Status Register 2 Interface Busy

1 = busy; 0 = not busy.

Status Register 3 Interface Firmware ID

Always 3 (the firmware ID of the HP 98629A interface).

Status Register 4 Not Implemented

Status Register 5 Data Availability

0 = receiver buffer empty;
1 = receiver data available but no control blocks buffered:
2 = receiver control blocks available but no data buffered;
3 = both control blocks and data available.

Status Register 6 Node Address of the interface

Node address of the HP 98629A interface installed in **this** computer which is set to the specified select code. The range of node addresses is 0 through 63.

Status Register 7 CRC Errors

Total number of cyclic redundancy check (CRC) errors detected by the interface since powerup or (RESET).

Status Register 8 Buffer Overflows

Total number of times the receive buffer has overflowed since powerup or (RESET).

Status Register 11 Amount of available space (number of bytes) in the transmit-data buffer.

Status Register 12 Number of transmission retries performed since powerup or (RESET).

Useful Tables

Option Numbers

1	BASIC Main	19	ERR
2	GRAPH	20	DISC
3	GRAPHX	21	CS80
4	IO	22	BUBBLE
5	BASIC Main	23	EPROM
6	TRANS	24	HP 9885
7	MAT	25	HPIB
8	PDEV	26	FHPIB
9	XREF	27	SERIAL
10	KBD	28	GPIO
11	CLOCK	29	BCD
12	LEX	30	DCOMM
13	BASIC Main	31-40	Reserved
14	MS	41	"PHYREC"
15	SRM	42	CRTB
16-17	Reserved	43	CRTA
18	KNB2–0		

Interface Select Codes

Internal Select Codes

1 Display (alpha)
2 Keyboard
3 Display (graphics)
4 Internal floppy-disc drive
5 Optional powerfail protection interface
6 Display (Graphics for bit mapped)
7 HP-IB interface (built-in)

Factory Presets for External Interfaces

8 HP-IB
9 RS-232
10 (not used)
11 BCD
12 GPIO
14 HP-IB Disc Interface
20 Data Communications
21 Shared Resource Management
27 EPROM Programmer
28 Color Output
30 Bubble Memory
32 Parity, Cache, Float (Pseudo Select Code)

Display-Enhancement Characters

Alpha Displays:

Character Code	Action Resulting from Displaying the Character	
128	All enhancements off	
129	Inverse mode on	
130	Blinking mode on	
131	Inverse and Blinking modes on	
132	Underline mode on	
133	Underline and Inverse modes on	
134	Underline and Blinking modes on	
135	Underline, Inverse, and Blinking modes on	
136	White	Model 236C alpha colors. (CRT control registers 5 and 15 through 17 also provide a method of changing the alpha color.)
137	Red	
138	Yellow	
139	Green	
140	Cyan	
141	Blue	
142	Magenta	
143	Black	

Bit-Mapped Displays:

Character Code	Action Resulting from Displaying the Character	
128	All enhancements off	
129	Inverse mode on	
130	No action	
131	Inverse mode on	
132	Underline mode on	
133	Underline and Inverse modes on	
134	Underline mode on	
135	Underline and Inverse modes on	
136	White (pen 1)	Default color map of displays with at least three color planes. (CRT control registers 5 and 15 through 17 also provide a method of changing alpha color.)
137	Red (pen 2)	
138	Yellow (pen 3)	
139	Green (pen 4)	
140	Cyan (pen 5)	
141	Blue (pen 6)	
142	Magenta (pen 7)	
143	Black (pen 0)	

PRINTing CHR$(*x*), where 136≤*x*≤143, will provide the same colors as on the Model 236C as long as the color map contains default values and the alpha write-enable mask includes planes 0 through 2. A user-defined color map which changes the values of pens 0 to 7 will change the meaning of CHR$(*x*).

US ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
NUL	0	00000000	000	00	
SOH	1	00000001	001	01	GTL
STX	2	00000010	002	02	
ETX	3	00000011	003	03	
EOT	4	00000100	004	04	SDC
ENQ	5	00000101	005	05	PPC
ACK	6	00000110	006	06	
BEL	7	00000111	007	07	
BS	8	00001000	010	08	GET
HT	9	00001001	011	09	TCT
LF	10	00001010	012	0A	
VT	11	00001011	013	0B	
FF	12	00001100	014	0C	
CR	13	00001101	015	0D	
SO	14	00001110	016	0E	
SI	15	00001111	017	0F	
DLE	16	00010000	020	10	
DC1	17	00010001	021	11	LLO
DC2	18	00010010	022	12	
DC3	19	00010011	023	13	
DC4	20	00010100	024	14	DCL
NAK	21	00010101	025	15	PPU
SYNC	22	00010110	026	16	
ETB	23	00010111	027	17	
CAN	24	00011000	030	18	SPE
EM	25	00011001	031	19	SPD
SUB	26	00011010	032	1A	
ESC	27	00011011	033	1B	
FS	28	00011100	034	1C	
GS	29	00011101	035	1D	
RS	30	00011110	036	1E	
US	31	00011111	037	1F	

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
space	32	00100000	040	20	LA0
!	33	00100001	041	21	LA1
"	34	00100010	042	22	LA2
#	35	00100011	043	23	LA3
$	36	00100100	044	24	LA4
%	37	00100101	045	25	LA5
&	38	00100110	046	26	LA6
'	39	00100111	047	27	LA7
(	40	00101000	050	28	LA8
)	41	00101001	051	29	LA9
*	42	00101010	052	2A	LA10
+	43	00101011	053	2B	LA11
,	44	00101100	054	2C	LA12
-	45	00101101	055	2D	LA13
.	46	00101110	056	2E	LA14
/	47	00101111	057	2F	LA15
0	48	00110000	060	30	LA16
1	49	00110001	061	31	LA17
2	50	00110010	062	32	LA18
3	51	00110011	063	33	LA19
4	52	00110100	064	34	LA20
5	53	00110101	065	35	LA21
6	54	00110110	066	36	LA22
7	55	00110111	067	37	LA23
8	56	00111000	070	38	LA24
9	57	00111001	071	39	LA25
:	58	00111010	072	3A	LA26
;	59	00111011	073	3B	LA27
<	60	00111100	074	3C	LA28
=	61	00111101	075	3D	LA29
>	62	00111110	076	3E	LA30
?	63	00111111	077	3F	UNL

US ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
@	64	01000000	100	40	TA0
A	65	01000001	101	41	TA1
B	66	01000010	102	42	TA2
C	67	01000011	103	43	TA3
D	68	01000100	104	44	TA4
E	69	01000101	105	45	TA5
F	70	01000110	106	46	TA6
G	71	01000111	107	47	TA7
H	72	01001000	110	48	TA8
I	73	01001001	111	49	TA9
J	74	01001010	112	4A	TA10
K	75	01001011	113	4B	TA11
L	76	01001100	114	4C	TA12
M	77	01001101	115	4D	TA13
N	78	01001110	116	4E	TA14
O	79	01001111	117	4F	TA15
P	80	01010000	120	50	TA16
Q	81	01010001	121	51	TA17
R	82	01010010	122	52	TA18
S	83	01010011	123	53	TA19
T	84	01010100	124	54	TA20
U	85	01010101	125	55	TA21
V	86	01010110	126	56	TA22
W	87	01010111	127	57	TA23
X	88	01011000	130	58	TA24
Y	89	01011001	131	59	TA25
Z	90	01011010	132	5A	TA26
[	91	01011011	133	5B	TA27
\	92	01011100	134	5C	TA28
]	93	01011101	135	5D	TA29
^	94	01011110	136	5E	TA30
_	95	01011111	137	5F	UNT

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
`	96	01100000	140	60	SC0
a	97	01100001	141	61	SC1
b	98	01100010	142	62	SC2
c	99	01100011	143	63	SC3
d	100	01100100	144	64	SC4
e	101	01100101	145	65	SC5
f	102	01100110	146	66	SC6
g	103	01100111	147	67	SC7
h	104	01101000	150	68	SC8
i	105	01101001	151	69	SC9
j	106	01101010	152	6A	SC10
k	107	01101011	153	6B	SC11
l	108	01101100	154	6C	SC12
m	109	01101101	155	6D	SC13
n	110	01101110	156	6E	SC14
o	111	01101111	157	6F	SC15
p	112	01110000	160	70	SC16
q	113	01110001	161	71	SC17
r	114	01110010	162	72	SC18
s	115	01110011	163	73	SC19
t	116	01110100	164	74	SC20
u	117	01110101	165	75	SC21
v	118	01110110	166	76	SC22
w	119	01110111	167	77	SC23
x	120	01111000	170	78	SC24
y	121	01111001	171	79	SC25
z	122	01111010	172	7A	SC26
{	123	01111011	173	7B	SC27
|	124	01111100	174	7C	SC28
}	125	01111101	175	7D	SC29
~	126	01111110	176	7E	SC30
DEL	127	01111111	177	7F	SC31

U.S./European Display Characters

These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
^@	0	00000000
^A	1	00000001
^B	2	00000010
^C	3	00000011
^D	4	00000100
^E	5	00000101
^F	6	00000110
^G	7	00000111
^H	8	00001000
^I	9	00001001
^J	10	00001010
^K	11	00001011
^L	12	00001100
^M	13	00001101
^N	14	00001110
^O	15	00001111
^P	16	00010000
^Q	17	00010001
^R	18	00010010
^S	19	00010011
^T	20	00010100
^U	21	00010101
^V	22	00010110
^W	23	00010111
^X	24	00011000
^Y	25	00011001
^Z	26	00011010
^[	27	00011011
^\	28	00011100
^]	29	00011101
^^	30	00011110
^_	31	00011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	32	00100000
!	33	00100001
"	34	00100010
#	35	00100011
$	36	00100100
%	37	00100101
&	38	00100110
'	39	00100111
(	40	00101000
)	41	00101001
*	42	00101010
+	43	00101011
,	44	00101100
-	45	00101101
.	46	00101110
/	47	00101111
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110101
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010
[	91	01011011
\	92	01011100
]	93	01011101
^	94	01011110
_	95	01011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
`	96	01100000
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010
{	123	01111011
|	124	01111100
}	125	01111101
~	126	01111110
▒	127	01111111

STD-LL-80182

U.S./European Display Characters

These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	128	10000000
	129	10000001
	130	10000010
	131	10000011
	132	10000100
	133	10000101
	134	10000110
	135	10000111
	136	10001000
	137	10001001
	138	10001010
	139	10001011
	140	10001100
	141	10001101
	142	10001110
	143	10001111
	144	10010000
	145	10010001
	146	10010010
	147	10010011
	148	10010100
	149	10010101
	150	10010110
	151	10010111
	152	10011000
	153	10011001
	154	10011010
	155	10011011
	156	10011100
	157	10011101
	158	10011110
	159	10011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	160	10100000
	161	10100001
	162	10100010
	163	10100011
	164	10100100
	165	10100101
	166	10100110
	167	10100111
	168	10101000
	169	10101001
	170	10101010
	171	10101011
	172	10101100
	173	10101101
	174	10101110
	175	10101111
	176	10110000
	177	10110001
	178	10110010
	179	10110011
	180	10110100
	181	10110101
	182	10110110
	183	10110111
	184	10111000
	185	10111001
	186	10111010
	187	10111011
	188	10111100
	189	10111101
	190	10111110
	191	10111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	192	11000000
	193	11000001
	194	11000010
	195	11000011
	196	11000100
	197	11000101
	198	11000110
	199	11000111
	200	11001000
	201	11001001
	202	11001010
	203	11001011
	204	11001100
	205	11001101
	206	11001110
	207	11001111
	208	11010000
	209	11010001
	210	11010010
	211	11010011
	212	11010100
	213	11010101
	214	11010110
	215	11010111
	216	11011000
	217	11011001
	218	11011010
	219	11011011
	220	11011100
	221	11011101
	222	11011110
	223	11011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	224	11100000
	225	11100001
	226	11100010
	227	11100011
	228	11100100
	229	11100101
	230	11100110
	231	11100111
	232	11101000
	233	11101001
	234	11101010
	235	11101011
	236	11101100
	237	11101101
	238	11101110
	239	11101111
	240	11110000
	241	11110001
	242	11110010
	243	11110011
	244	11110100
	245	11110101
	246	11110110
	247	11110111
	248	11111000
	249	11111001
	250	11111010
	251	11111011
	252	11111100
	253	11111101
	254	11111110
	255	11111111

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	NU	32		64	@	96	`
1	SH	33	!	65	A	97	a
2	SX	34	"	66	B	98	b
3	EX	35	#	67	C	99	c
4	ET	36	$	68	D	100	d
5	EQ	37	%	69	E	101	e
6	AK	38	&	70	F	102	f
7	Δ	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DL	48	0	80	P	112	p
17	D1	49	1	81	Q	113	q
18	D2	50	2	82	R	114	r
19	D3	51	3	83	S	115	s
20	D4	52	4	84	T	116	t
21	NK	53	5	85	U	117	u
22	SY	54	6	86	V	118	v
23	EB	55	7	87	W	119	w
24	CN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SB	58	:	90	Z	122	z
27	EC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	■

U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	CL	160		192	â	224	Á
129	IV	161	À	193	ê	225	Â
130	BG	162	Â	194	ô	226	ã
131	IB	163	È	195	û	227	Ð
132	UL	164	Ê	196	á	228	đ
133	IV	165	Ë	197	é	229	Í
134	BG	166	î	198	ó	230	Ì
135	IV	167	Ï	199	ú	231	ó
136	HH	168	´	200	à	232	ò
137	RD	169	`	201	è	233	õ
138	YE	170	^	202	ò	234	ð
139	GR	171	¨	203	ù	235	š
140	CY	172	~	204	ä	236	š
141	BU	173	Ù	205	ë	237	Ú
142	MG	174	Û	206	ö	238	Ÿ
143	BK	175	£	207	ü	239	ÿ
144	S0	176	¯	208	Ä	240	Þ
145	S1	177	B1	209	î	241	þ
146	S2	178	B2	210	Ø	242	F2
147	S3	179	·	211	Æ	243	F3
148	S4	180	Ç	212	à	244	F4
149	S5	181	ç	213	í	245	I0
150	S6	182	Ñ	214	ø	246	–
151	S7	183	ñ	215	æ	247	¼
152	S8	184	¡	216	Ä	248	½
153	S9	185	¿	217	ì	249	ª
154	SA	186	¤	218	Ö	250	º
155	SB	187	£	219	Ü	251	«
156	SC	188	¥	220	é	252	■
157	SD	189	§	221	ï	253	»
158	SE	190	ƒ	222	ß	254	±
159	SF	191	¢	223	Ô	255	K

U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	NU	32		64	@	96	`
1	SH	33	!	65	A	97	a
2	SX	34	"	66	B	98	b
3	EX	35	#	67	C	99	c
4	ET	36	$	68	D	100	d
5	EQ	37	%	69	E	101	e
6	AK	38	&	70	F	102	f
7	BL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DL	48	0	80	P	112	p
17	D1	49	1	81	Q	113	q
18	D2	50	2	82	R	114	r
19	D3	51	3	83	S	115	s
20	D4	52	4	84	T	116	t
21	NK	53	5	85	U	117	u
22	SY	54	6	86	V	118	v
23	EB	55	7	87	W	119	w
24	CN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SB	58	:	90	Z	122	z
27	EC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	CL	160		192	â	224	Á
129	IV	161	À	193	ê	225	Â
130	BG	162	Â	194	ô	226	ã
131	IB	163	È	195	û	227	Ð
132	UL	164	ê	196	á	228	ð
133	IL	165	Ë	197	é	229	Ë
134	BG	166	î	198	ó	230	ë
135	IL	167	Ï	199	ú	231	Ó
136	WH	168	´	200	à	232	Ò
137	RD	169	`	201	è	233	Õ
138	YE	170	^	202	ò	234	õ
139	GR	171	¨	203	ù	235	Š
140	CY	172	~	204	ä	236	š
141	BU	173	Ù	205	ë	237	Ú
142	MG	174	Ô	206	ö	238	Ÿ
143	BK	175	£	207	ü	239	ÿ
144	90	176	¯	208	Å	240	Þ
145	91	177	Ý	209	î	241	þ
146	92	178	ý	210	Ø	242	·
147	93	179	·	211	Æ	243	µ
148	94	180	Ç	212	å	244	¶
149	95	181	ç	213	Í	245	IO
150	96	182	Ñ	214	ø	246	–
151	97	183	ñ	215	æ	247	¼
152	98	184	¡	216	Ä	248	½
153	99	185	¿	217	ì	249	Ä
154	9A	186	¤	218	Ö	250	¤
155	9B	187	£	219	Ü	251	«
156	9C	188	¥	220	É	252	■
157	9D	189	§	221	ï	253	»
158	9E	190	ƒ	222	ß	254	±
159	9F	191	¢	223	Ô	255	■

Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	0	00000000
	1	00000001
	2	00000010
	3	00000011
	4	00000100
	5	00000101
	6	00000110
	7	00000111
	8	00001000
	9	00001001
	10	00001010
	11	00001011
	12	00001100
	13	00001101
	14	00001110
	15	00001111
	16	00010000
	17	00010001
	18	00010010
	19	00010011
	20	00010100
	21	00010101
	22	00010110
	23	00010111
	24	00011000
	25	00011001
	26	00011010
	27	00011011
	28	00011100
	29	00011101
	30	00011110
	31	00011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	32	00100000
!	33	00100001
"	34	00100010
#	35	00100011
$	36	00100100
%	37	00100101
&	38	00100110
'	39	00100111
(	40	00101000
)	41	00101001
*	42	00101010
+	43	00101011
,	44	00101100
-	45	00101101
.	46	00101110
/	47	00101111
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110101
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010
[	91	01011011
¥	92	01011100
]	93	01011101
^	94	01011110
_	95	01011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
`	96	01100000
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010
{	123	01111011
|	124	01111100
}	125	01111101
~	126	01111110
	127	01111111

Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ｦ	128	10000000
ｧ	129	10000001
ｨ	130	10000010
ｩ	131	10000011
ｪ	132	10000100
ｫ	133	10000101
ｬ	134	10000110
ｭ	135	10000111
ｮ	136	10001000
ｯ	137	10001001
ｰ	138	10001010
ｱ	139	10001011
ｲ	140	10001100
ｳ	141	10001101
ｴ	142	10001110
ｵ	143	10001111
ｶ	144	10010000
ｷ	145	10010001
ｸ	146	10010010
ｹ	147	10010011
ｺ	148	10010100
ｻ	149	10010101
ｼ	150	10010110
ｽ	151	10010111
ｾ	152	10011000
ｿ	153	10011001
ﾀ	154	10011010
ﾁ	155	10011011
ﾂ	156	10011100
ﾃ	157	10011101
ﾄ	158	10011110
ﾅ	159	10011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ﾆ	160	10100000
ﾇ	161	10100001
ﾈ	162	10100010
ﾉ	163	10100011
ﾊ	164	10100100
ﾋ	165	10100101
ﾌ	166	10100110
ﾍ	167	10100111
ﾎ	168	10101000
ﾏ	169	10101001
ﾐ	170	10101010
ﾑ	171	10101011
ﾒ	172	10101100
ﾓ	173	10101101
ﾔ	174	10101110
ﾕ	175	10101111
ﾖ	176	10110000
ﾗ	177	10110001
ﾘ	178	10110010
ﾙ	179	10110011
ﾚ	180	10110100
ﾛ	181	10110101
ﾜ	182	10110110
ﾝ	183	10110111
ﾞ	184	10111000
ﾟ	185	10111001
｡	186	10111010
｢	187	10111011
｣	188	10111100
､	189	10111101
･	190	10111110
ｰ	191	10111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ｦ	192	11000000
ｧ	193	11000001
ｨ	194	11000010
ｩ	195	11000011
ｪ	196	11000100
ｫ	197	11000101
ｬ	198	11000110
ｭ	199	11000111
ｮ	200	11001000
ｯ	201	11001001
ｰ	202	11001010
ｱ	203	11001011
ｲ	204	11001100
ｳ	205	11001101
ｴ	206	11001110
ｵ	207	11001111
ｶ	208	11010000
ｷ	209	11010001
ｸ	210	11010010
ｹ	211	11010011
ｺ	212	11010100
ｻ	213	11010101
ｼ	214	11010110
ｽ	215	11010111
ｾ	216	11011000
ｿ	217	11011001
ﾀ	218	11011010
ﾁ	219	11011011
ﾂ	220	11011100
ﾃ	221	11011101
ﾄ	222	11011110
ﾅ	223	11011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ﾆ	224	11100000
ﾇ	225	11100001
ﾈ	226	11100010
ﾉ	227	11100011
ﾊ	228	11100100
ﾋ	229	11100101
ﾌ	230	11100110
ﾍ	231	11100111
ﾎ	232	11101000
ﾏ	233	11101001
ﾐ	234	11101010
ﾑ	235	11101011
ﾒ	236	11101100
ﾓ	237	11101101
ﾔ	238	11101110
ﾕ	239	11101111
ﾖ	240	11110000
ﾗ	241	11110001
ﾘ	242	11110010
ﾙ	243	11110011
ﾚ	244	11110100
ﾛ	245	11110101
ﾜ	246	11110110
ﾝ	247	11110111
ﾞ	248	11111000
ﾟ	249	11111001
｡	250	11111010
｢	251	11111011
｣	252	11111100
､	253	11111101
･	254	11111110
ｰ	255	11111111

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	0	00000000
	1	00000001
	2	00000010
	3	00000011
	4	00000100
	5	00000101
	6	00000110
	7	00000111
	8	00001000
	9	00001001
	10	00001010
	11	00001011
	12	00001100
	13	00001101
	14	00001110
	15	00001111
	16	00010000
	17	00010001
	18	00010010
	19	00010011
	20	00010100
	21	00010101
	22	00010110
	23	00010111
	24	00011000
	25	00011001
	26	00011010
	27	00011011
	28	00011100
	29	00011101
	30	00011110
	31	00011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
	32	00100000
!	33	00100001
"	34	00100010
#	35	00100011
$	36	00100100
%	37	00100101
&	38	00100110
'	39	00100111
(	40	00101000
)	41	00101001
*	42	00101010
+	43	00101011
,	44	00101100
-	45	00101101
.	46	00101110
/	47	00101111
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110101
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
:	58	00111010
;	59	00111011
<	60	00111100
=	61	00111101
>	62	00111110
?	63	00111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
@	64	01000000
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010
[	91	01011011
¥	92	01011100
]	93	01011101
^	94	01011110
_	95	01011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
`	96	01100000
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010
{	123	01111011
|	124	01111100
}	125	01111101
~	126	01111110
▒	127	01111111

STD-LL-60182

Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
CL	128	10000000
IV	129	10000001
BG	130	10000010
IB	131	10000011
UL	132	10000100
IU	133	10000101
BS	134	10000110
IK	135	10000111
WH	136	10001000
RD	137	10001001
YE	138	10001010
GR	139	10001011
CY	140	10001100
BU	141	10001101
MG	142	10001110
BK	143	10001111
S0	144	10010000
S1	145	10010001
S2	146	10010010
S3	147	10010011
S4	148	10010100
S5	149	10010101
S6	150	10010110
S7	151	10010111
S8	152	10011000
S9	153	10011001
SA	154	10011010
SB	155	10011011
SC	156	10011100
SD	157	10011101
SE	158	10011110
SF	159	10011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
｠	160	10100000
｡	161	10100001
｢	162	10100010
｣	163	10100011
､	164	10100100
･	165	10100101
ｦ	166	10100110
ｧ	167	10100111
ｨ	168	10101000
ｩ	169	10101001
ｪ	170	10101010
ｫ	171	10101011
ｬ	172	10101100
ｭ	173	10101101
ｮ	174	10101110
ｯ	175	10101111
ｰ	176	10110000
ｱ	177	10110001
ｲ	178	10110010
ｳ	179	10110011
ｴ	180	10110100
ｵ	181	10110101
ｶ	182	10110110
ｷ	183	10110111
ｸ	184	10111000
ｹ	185	10111001
ｺ	186	10111010
ｻ	187	10111011
ｼ	188	10111100
ｽ	189	10111101
ｾ	190	10111110
ｿ	191	10111111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
ﾀ	192	11000000
ﾁ	193	11000001
ﾂ	194	11000010
ﾃ	195	11000011
ﾄ	196	11000100
ﾅ	197	11000101
ﾆ	198	11000110
ﾇ	199	11000111
ﾈ	200	11001000
ﾉ	201	11001001
ﾊ	202	11001010
ﾋ	203	11001011
ﾌ	204	11001100
ﾍ	205	11001101
ﾎ	206	11001110
ﾏ	207	11001111
ﾐ	208	11010000
ﾑ	209	11010001
ﾒ	210	11010010
ﾓ	211	11010011
ﾔ	212	11010100
ﾕ	213	11010101
ﾖ	214	11010110
ﾗ	215	11010111
ﾘ	216	11011000
ﾙ	217	11011001
ﾚ	218	11011010
ﾛ	219	11011011
ﾜ	220	11011100
ﾝ	221	11011101
ﾞ	222	11011110
ﾟ	223	11011111

ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary
E0	224	11100000
E1	225	11100001
E2	226	11100010
E3	227	11100011
E4	228	11100100
E5	229	11100101
E6	230	11100110
E7	231	11100111
E8	232	11101000
E9	233	11101001
EA	234	11101010
EB	235	11101011
EC	236	11101100
ED	237	11101101
EE	238	11101110
EF	239	11101111
F0	240	11110000
F1	241	11110001
F2	242	11110010
F3	243	11110011
F4	244	11110100
I0	245	11110101
F6	246	11110110
F7	247	11110111
F8	248	11111000
F9	249	11111001
FA	250	11111010
FB	251	11111011
■	252	11111100
FD	253	11111101
FE	254	11111110
▒	255	11111111

STD-TL-9082

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

Master Reset Table

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
CRT														
CRT DISP Line	Clear	Clear	—	—	Clear	—	—	—	—	—	—	—	—	—
CRT Display Functions	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
CRT Message Line	Ready	Clear	Clear	Clear	Reset	—	—	—	—	—	—	Clear	—	—
CRT Input Line (Note 6)	Clear	Clear	Clear	—	Clear	—	—	—	—	—	—	—	—	—
CRT Printout Area	Clear	Clear	—	—	—	—	—	—	—	—	—	—	—	—
CRT Print Position (TABXY)	1,1	1,1	—	—	Note 15	—	—	—	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—	—	—	—	—	—	—
KEYBOARD														
Keyboard Recall Buffer	Clear	—	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Result Buffer	Empty	Empty	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Knob Mode	↕	↕	↕	↕	↕	—	—	—	—	—	—	—	—	—
Tabs On Input Line	None	None	—	—	—	—	—	—	—	—	—	—	—	—
Typing Aid Labels	Note 16	Note 16	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Katakana Mode	Off	Off	Off	—	Off	—	—	—	—	—	—	—	—	—
SUSPEND INTERACTIVE	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	—	Off	—	—
PRINTING														
Print column	1	1	—	—	1	—	—	—	—	—	—	—	—	—
PRINTALL	Off	Off	—	—	Off	—	—	—	—	—	—	—	—	—
PRINTALL IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
PRINTER IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
ENVIRONMENTS & VARIABLES														
Allocated Variables	None	None	None	None	Note 1	Note 1	None	None	None	None	—	None	None	Pre-ent
Normal Variables	None	None	None	None	—	—	None	None	None	None	—	Note 11	Note 11	Pre-ent
COM Variables	None	None	—	None	—	—	—	Note 9	—	Note 9	—	—	—	—
OPTION BASE	0	0	0	—	—	—	—	Note 9	—	Note 9	—	Note 9	Note 9	Pre-ent
I/O Path Names	None	Closed	Closed	Closed	None	Closed	Closed	Closed	Closed	Closed	—	Closed	—	sub clsd
I/O Path Names in COM	None	Closed	—	Closed	None	—	Note 10	Note 10	Note 10	Note 10	—	—	—	—
Keyboard Variable Access	No	No	No	No	Main	Main	No	In cnt.	No	In cnt.	In cnt.	Main	SUB	Pre-ent
BASIC Program Lines	None	None	None	—	—	—	Note 4	Note 4	Note 4	Note 4	Note 4	—	—	—
BASIC Program Environment	Main	Main	Main	Main	Main	Main	Main	Main	Main	Main	—	Main	SUB	Pre-ent
Normal Binary Programs	None	None	—	—	—	—	Note 5	Note 5	—	—	—	—	—	—
SUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Push	Pop
NPAR	0	0	0	0	0	0	0	0	0	0	—	0	Actual	Pre-ent
CONTINUE Allowed	No	No	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
ON <event> ACTIONS														
ON <event> Log	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	Empty	Note 8	Note 8
System Priority	0	0	0	0	0	0	0	0	0	0	—	0	Note 7	Pre-ent
ON KEY Labels	None	None	None	None	None	None	None	None	None	None	—	None	—	Pre-ent
ENABLE/DISABLE	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	—	Enable	—	—
KNOBX & KNOBY	0	0	0	0	0	0	0	0	0	0	—	0	—	—

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
MISC.														
GOSUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Local	Pre-ent
TIMEDATE	Note 14	—	—	—	—	—	—	—	—	—	—	—	—	—
ERRL, ERRN, and ERRDS	0	0	—	—	—	—	—	0	—	0	—	0	—	—
ERRM$	Null	Null	—	—	—	—	—	Null	—	Null	—	Null	—	—
DATA Pointer	None	None	None	None	None	None	None	1st main	None	1st main	—	1st main	1st sub	Pre-ent
LEXICAL ORDER IS	Stand.	Stand.	—	—	—	—	—	—	—	—	—	—	—	—
MASS STORAGE IS	Note 12	Note 12	—	—	—	—	—	—	—	—	—	—	—	—
CHECKREAD ON/OFF	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
Angle Mode	RAD	RAD	RAD	RAD	—	—	RAD	RAD	RAD	RAD	—	RAD	—	Pre-ent
Random Number Seed	Note 13	Note 13	Note 13	—	—	—	—	Note 13	—	Note 13	—	Note 13	—	—
DET	0	0	0	—	—	—	—	—	—	—	—	0	—	—
TRANSFER	None	Aborts	Note 17	Waits	Aborts	Waits	None	Note 18	None	Waits	—	None	—	Note 19
TRACE ALL	Off	Off	Off	—	—	—	—	—	—	—	—	—	—	—

— = Unchanged
Pre-ent = As existed previous to entry into the subprogram.
In cnt. = Access to variables in current context only.
1st main = Pointer set to first DATA statement in main program.
1st sub = Pointer set to first DATA statement in subprogram.
sub clsd = All local I/O path names are closed at subexit.
Waits = Operation waits until TRANSFER completes.

Note 1: Only those allocated in the main program are available.
Note 2: Pressing the STOP key is identical in function to executing STOP. Editing or altering a paused program causes the program to go into the stopped state.
Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.
Note 4: Modified according to the statement or command parameters and file contents.
Note 5: Any new binary programs in the file are loaded.
Note 6: Includes cursor position, INS CHR mode, ANY CHAR sequence state, but not tabs, auto-repeat rate, and auto-repeat delay. (These last three are defaulted only at SCRATCH A and Power On.)
Note 7: The system priority changes at SUB entry if the subroutine was invoked by an ON <event> CALL.
Note 8: See the appropriate keyword.
Note 9: As specified by the new environment or program.
Note 10: A COM mismatch between programs will close I/O path names. If I/O path names exist in a labeled COM, and a LOAD or GET brings in a program which does not contain that labeled COM, those I/O path names are closed.
Note 11: Numeric variables are set to 0, and string lengths are set to 0.
Note 12: The default mass storage device is INTERNAL (the right-hand drive) on the 9826 and 9836. See the 9816 Installation Manual for information on its default mass storage device.
Note 13: The default random number seed is $INT(PI \times (2^{31} - 2)/180)$. This is equal to 37 480 660.
Note 14: The default TIMEDATE is 2.086 629 12 E+11 (midnight March 1, 1900, Julian time).
Note 15: After a RESET, the CRT print position is in column one of the next line below the print position before the RESET.
Note 16: Typing aid labels are displayed unless a program is in the RUN state.
Note 17: Operation waits until TRANSFER completes unless both I/O path names are in COM.
Note 18: Operation waits until TRANSFER completes unless both I/O path names are in a COM area preserved during the LOAD.
Note 19: Operation waits until TRANSFER completes if the TRANSFER uses a local I/O path name.

Further Comments

The PAUSE key, the programmed PAUSE statement, and executing PAUSE from the keyboard all have identical effects. The only permanent effects of the sequence "PAUSE...CONTINUE" on a running program are:

1. Delay in execution.
2. Second and subsequent interrupt events of a given type are ignored.
3. INPUT, LINPUT, and ENTER 2 statements will be restarted.
4. ON KEY and ON KNOB are temporarily deactivated (i.e. not logged or executed) during the pause.
5. A TRANSFER may complete during the pause, causing ON EOT to be serviced at the next end-of-line.

Fatal program errors (i.e. those not trapped by ON ERROR) have the following effects:

— a PAUSE
— a beep
— an error message in the message line
— setting the values of the ERRL, the ERRN, and possibly the ERRDS functions
— setting the default EDIT line number to the number of the line in which the error occurred.

Autostart is equivalent to: Power On, LOAD "AUTOST", RUN.

CLR IO terminates ENTER and OUTPUT on all interfaces, handshake setup operations, HP-IB control operations, DISP, ENTER from CRT or keyboard, CAT, LIST, external plotter output, and output to the PRINTER IS, PRINTALL IS, and DUMP DEVICE IS devices when they are external. CLR IO does not terminate CONTROL, STATUS, READIO, WRITEIO, TRANSFER, real-time clock operations, mass storage operations (other than CAT), OUTPUT 2 (keyboard), or message line output.

CLR IO clears any pending closure key action.

If CLR IO is used to abort a DUMP GRAPHICS to an external device, the external device may be in the middle of an escape-code sequence. Thus, it might be counting characters to determine when to return to normal mode (from graphics mode). This means that a subsequent I/O operation to the same device may yield "strange" results. Handling this situation is the responsibility of the user and is beyond the scope of the firmware provided with the product. Sending 75 ASCII nulls is one way to "clear" the 9876 Graphics Printer.

Graphic Reset Table

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	GINIT	Main Prerun
PLOTTER IS	CRT	CRT	—	—	CRT	—	CRT	—
Graphics Memory	Clear	Clear	—	—	Note 1	—	Note 1	—
VIEWPORT	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
X and Y Scaling (unit of measure)	GDU	GDU	—	—	GDU	—	GDU	—
Soft Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
Current Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
CLIP ON/OFF	Off	Off	—	—	Off	—	Off	—
PIVOT	0	0	—	—	0	—	0	—
AREA PEN	1	1	—	—	1	—	1	—
PEN	1	1	—	—	1	—	1	—
LINE TYPE	1,5	1,5	—	—	1,5	—	1,5	—
Pen Position	0,0	0,0	—	—	0,0	—	0,0	—
LORG	1	1	—	—	1	—	1	—
CSIZE	5,.6	5,.6	—	—	5,.6	—	5,.6	—
LDIR	0	0	—	—	0	—	0	—
PDIR	0	0	—	—	0	—	0	—
GRAPHICS ON/OFF	Off	Off	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—
DUMP DEVICE IS	701	701	—	—	—	—	—	—
GRAPHICS INPUT IS	None	None	—	—	None	—	None	—
TRACK ... ON/OFF	Off	Off	—	—	Off	—	Off	—
Color Map (Note 4)	Off	Off	—	—	Note 5	—	Note 5	—
Drawing Mode	Norm	Norm	—	—	Norm	—	Norm	—

— = Unchanged
hrd clip = The default hard clip boundaries of the CRT.

Note 1: Although RESET leaves the graphics memory unchanged, it will be cleared upon execution of the next graphics statement that sets a default plotter following the RESET.
Note 2: Pressing the STOP key is identical to executing STOP. Altering a paused program causes the program to go into the stopped state.
Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.
Note 4: With color map off, 8 standard colors are available. With color map on, 16 user-defined colors are available. See PLOTTER IS.
Note 5: Although the color map remains unchanged, it is changed if a graphics statement selects the device as a default plotter.

Interface Reset Table

	Power On	SCRATCH A	SCRATCH	BASIC RESET	Note 5 END/ STOP	LOAD	GET	Note 6 Reset Cmd	Main Prerun	SUB Entry	SUB Exit	CLR I/O
GPIO Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card (PRESET)	Reset	Note 1	Note 1	Reset	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
PSTS Error Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
RS-232 Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Data Rate/Character Format	Swtch	Swtch	—	—	—	—	—	—	—	—	—	—
RTS-DTR Latch	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Request to Send Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Data Terminal Ready Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Line Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Modem Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Data-In Buffer	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	—	Empty
Error-Pend. Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
HP-IB												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
User Interrupt Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Serial Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
Parallel Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
My Address Register	Note 4	Note 4	—	—	—	—	—	—	—	—	—	—
IFC Sent	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
REN Set True	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
Data Communications												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 7	—	Reset	—	—	—	Note 7	—	—	—	—
Line State	Dscon	Dscon	—	Dscon	—	—	—	Dscon	—	—	—	—
Data Buffers	Empty	Empty	—	Empty	—	—	—	Empty	—	—	—	—
Protocol Selection (Async or Data Link)	Swtch	Note 8	—	Swtch	—	—	—	Note 8	—	—	—	—
Protocol Options	Swtch	Swtch	—	Swtch	—	—	—	Swtch	—	—	—	—
BCD Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 1	Note 1	Note 1	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
Rewind Driver	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	—	—	Rwd
BCD/Binary Mode	Swtch	Swtch	—	—	—	—	—	—	—	—	—	—

	Power On	SCRATCH A	SCRATCH	BASIC RESET	Note 5 END/ STOP	LOAD	GET	Note 6 Reset Cmd	Main Prerun	SUB Entry	SUB Exit	CLR I/O
EPROM Programmer												
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Programming Time Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Target Address Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—

— = Unchanged
Swtch = Set according to the switches on the interface card
Dscon = A disconnect is performed

Note 1: Reset only if card is not ready.
Note 2: Cleared only if corresponding modem control line is not set.
Note 3: Sent only if System Controller.
Note 4: If System Controller and Active Controller, address is set to 21. Otherwise, it is set to 20.
Note 5: Pressing the STOP key is identical in function to executing STOP or END. Editing or altering a paused program causes the program to go into the stopped state.
Note 6: Caused by sending a non-zero value to CONTROL register 0.
Note 7: This is a "soft reset," which does not include an interface self-test or a reconfiguration of protocol.
Note 8: Set according to the value used in the most recent CONTROL statement directed to Register 3. If there has been no CONTROL 3 statement, the switch settings are used.

Second Byte of Non-ASCII Key Sequences (String)

Holding the CTRL key and pressing a non-ASCII key generates a two-character sequence on the CRT. The first character is an "inverse-video" K. This table can be used to look up the key that corresponds to the second character of the sequence. (On the small keyboard of the Model 216, some non-ASCII keys generate ASCII characters when they are pressed while holding the CTRL key.)

Character	Value	Key
space		[1]
!	33	STOP
"		[1]
#	35	CLR LN
$	36	ANY CHAR
%	37	CLR→END
&	38	Select
'	39	Prev
(	40	SHIFT-TAB
)	41	TAB
*	42	INS LN
+	43	INS CHR
,	44	Next
-	45	DEL CHR
.	46	Ignored
/	47	DEL LN
0	48	k0
1	49	k1
2	50	k2
3	51	k3
4	52	k4
5	53	k5
6	54	k6
7	55	k7
8	56	k8
9	57	k9
:	58	SHIFT-system f6 [2]
;	59	SHIFT-system f7 [2]
<	60	←
=	61	RESULT
>	62	→
?	63	RECALL
@	64	SHIFT-RECALL
A	65	PRT ALL
B	66	BACK SPACE
C	67	CONTINUE
D	68	EDIT
E	69	ENTER
F	70	DISPLAY FCTNS
G	71	SHIFT-→
H	72	SHIFT-←
I	73	CLR I/O
J	74	Katakana Mode
K	75	CLR SCR
L	76	GRAPHICS
M	77	ALPHA
N	78	DUMP GRAPHICS
O	79	DUMP ALPHA

Character	Value	Key
P	80	PAUSE
Q		[1]
R	82	RUN
S	83	STEP
T	84	SHIFT-↓
U	85	CAPS LOCK
V	86	↓
W	87	SHIFT-↑
X	88	EXECUTE
Y	89	Roman Mode
Z		[1]
[	91	CLR TAB
\	92	▶
]	93	SET TAB
^	94	↑
_	95	SHIFT-▶
`		[1]
a	97	k10
b	98	k11
c	99	k12
d	100	k13
e	101	k14
f	102	k15
g	103	k16
h	104	k17
i	105	k18
j	106	k19
k	107	k20
l	108	k21
m	109	k22
n	110	k23
o	111	SHIFT-system f1 [2]
p	112	SHIFT-system f2 [2]
q	113	SHIFT-system f3 [2]
r	114	SHIFT-system f4 [2]
s	115	SHIFT-user f1 [2]
t	116	SHIFT-user f2 [2]
u	117	SHIFT-user f3 [2]
v	118	SHIFT-user f4 [2]
w	119	SHIFT-user f5 [2]
x	120	SHIFT-user f6 [2]
y	121	SHIFT-user f7 [2]
z	122	SHIFT-user f8 [2]
}	123	System
|	124	Menu
{	125	User
~	126	SHIFT-Menu
▒		[1]

1 These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR$(255) in an output to the keyboard, an error is reported (`Error 131 Bad non-alphanumeric keycode.`).

2 System and user refer to the softkey menu which is currently active.

Selected High-Precision Metric Conversion Factors

English Units	Metric Units	To convert from English to Metric, multiply by:	To convert from Metric to English, multiply by:
Length			
mil	micrometre (micron)	2.54×10^{1}☆	$3.937\ 007\ 874 \times 10^{-2}$
inch	millimetre	2.54×10^{1}☆	$3.937\ 007\ 874 \times 10^{-2}$
foot	metre †	3.048×10^{-1}☆	$3.280\ 839\ 895$
mile (intl.)	kilometre	$1.609\ 344$☆	$6.213\ 711\ 922 \times 10^{-1}$
Area			
$inch^2$	$millimetre^2$	$6.451\ 6 \times 10^{2}$☆	$1.550\ 003\ 100 \times 10^{-3}$
$foot^2$	$metre^2$	$9.290\ 304 \times 10^{-2}$☆	$1.076\ 391\ 042 \times 10^{1}$
$mile^2$	$kilometre^2$	$2.589\ 988\ 110$	$3.861\ 021\ 585 \times 10^{-1}$
acre (U.S. survey)	hectare	$4.046\ 873 \times 10^{-1}$	$2.471\ 044$
Volume			
$inches^3$	$millimetres^3$	$1.638\ 706\ 4 \times 10^{4}$☆	$6.102\ 374\ 409 \times 10^{-5}$
$feet^3$	$metres^3$	$2.831\ 684\ 659 \times 10^{-2}$	$3.531\ 466\ 672 \times 10^{1}$
ounces (U.S. fluid)	$centimetres^3$	$2.957\ 353 \times 10^{1}$	$3.381\ 402 \times 10^{-2}$
gallon (U.S. fluid)	litre ‡	$3.785\ 412$	$2.641\ 721 \times 10^{-1}$
Mass			
pound (avdp.)	kilogram	$4.535\ 923\ 7 \times 10^{-1}$☆	$2.204\ 622\ 622$
ton (short)	ton (metric)	$9.071\ 847\ 4 \times 10^{-1}$☆	$1.102\ 311\ 311$
Force[1]			
ounce (force)	dyne	$2.780\ 138\ 510 \times 10^{4}$	$3.596\ 943\ 090 \times 10^{-5}$
pound (force)	newton	$4.448\ 221\ 615$	$2.248\ 089\ 431 \times 10^{-1}$
Pressure			
psi	pascal	$6.894\ 757\ 293 \times 10^{3}$	$1.450\ 377\ 377 \times 10^{-4}$
inches of Hg (at 32°F)	millibar	$3.386\ 4 \times 10^{1}$	$2.952\ 9 \times 10^{-2}$
Energy			
BTU (IST)	Calorie (kg, thermochem.)	$2.521\ 644\ 007 \times 10^{-1}$	$3.965\ 666\ 831$
BTU (IST)	watt-hour	$2.930\ 710\ 702 \times 10^{-1}$	$3.412\ 141\ 633$
BTU (IST)	joule §	$1.055\ 055\ 853 \times 10^{3}$	$9.478\ 171\ 203 \times 10^{-4}$
ft•lb	joule	$1.355\ 817\ 948$	$7.375\ 621\ 493 \times 10^{-1}$
Power			
BTU (IST)/hr	watt	$2.930\ 710\ 702 \times 10^{-1}$	$3.412\ 141\ 633$
horsepower (mechanical)	watt	$7.456\ 998\ 716 \times 10^{2}$	$1.341\ 022\ 090 \times 10^{-3}$
horsepower (electric)	watt	7.46×10^{2}☆	$1.340\ 482\ 574 \times 10^{-3}$
ft•lb/s	watt	$1.355\ 817\ 948$	$7.375\ 621\ 493 \times 10^{-1}$
Temperature			
°Rankine	kelvin	1.8☆	$5.555\ 555\ 556 \times 10^{-1}$
°Fahrenheit	°Celsius	°C=(°F−32) /1.8☆	°F = (°C x 1.8)+32☆

☆ Exact conversion
† Conversion redefined in 1959
‡ Conversion redefined in 1964
§ Conversion redefined in 1956

Note: The preferred metric unit for force is the newton; for pressure, the pascal; and for energy, the joule.

Prefix	Symbol	Multiplier
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	k	10^{3}
hecto	h	10^{2}
deka	da	10^{1}

Prefix	Symbol	Multiplier
deci	d	10^{-1}
centi	c	10^{-2}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}
atto	a	10^{-18}

Sources

American Society for Testing and Materials (ASTM), "Standard for Metric Practice". Reprinted from Annual Book of ASTM Standards.

U.S. Department of Commerce, National Bureau of Standards, "NBS Guidelines for the Use of the Metric System". Reprinted from Dimensions/NBS. (October 1977).

Error Messages

1 Missing option or configuration error. If a statement requires an option which is not loaded, the option number or option name is given along with error 1. These numbers are listed in the Useful Tables section. Error 1 without an option number indicates other configuration errors.

2 Memory overflow. If you get this error while loading a file, the program is too large for the computer's memory. If the program loads, but you get this error when you press RUN, then the overflow was caused by the variable declarations. Either way, you need to modify the program or add more read/write memory.

3 Line not found in current context. Could be a GOTO or GOSUB that references a non-existent (or deleted) line, or an EDIT command that refers to a non-existent line label.

4 Improper RETURN. Executing a RETURN statement without previously executing an appropriate GOSUB or function call. Also, a RETURN statement in a user-defined function with no value specified.

5 Improper context terminator. You forgot to put an END statement in the program. Also applies to SUBEND and FNEND.

6 Improper FOR...NEXT matching. Executing a NEXT statement without previously executing the matching FOR statement. Indicates improper nesting or overlapping of the loops.

7 Undefined function or subprogram. Attempt to call a SUB or user-defined function that is not in memory. Look out for program lines that assumed an optional CALL.

8 Improper parameter matching. A type mismatch between a pass parameter and a formal parameter of a subprogram.

9 Improper number of parameters. Passing either too few or too many parameters to a subprogram. Applies only to non-optional parameters.

10 String type required. Attempting to return a numeric from a user-defined string function.

11 Numeric type required. Attempting to return a string from a user-defined numeric function.

12 Attempt to redeclare variable. Including the same variable name twice in declarative statements such as DIM or INTEGER.

13 Array dimensions not specified. Using the (*) symbol after a variable name when that variable has never been declared as an array.

14 OPTION BASE not allowed here. The OPTION BASE statement must appear before any declarative statements such as DIM or INTEGER. Only one OPTION BASE statement is allowed in one context.

15 Invalid bounds. Attempt to declare an array with more than 32 767 elements or with upper bound less than lower bound.

16 Improper or inconsistent dimensions. Using the wrong number of subscripts when referencing an array element.

17 Subscript out of range. A subscript in an array reference is outside the current bounds of the array.

18 String overflow or substring error. String overflow is an attempt to put too many characters into a string (exceeding dimensioned length). This can happen in an assignment, an ENTER an INPUT, or a READ. A substring error is an attempted violation of the rules for substrings. Watch out for null strings where you weren't expecting them.

19 Improper value or out of range. A value is too large or too small. Applies to items found in a variety of statements. Often occurs when the number builder overflows (or underflows) during an I/O operation.

20 INTEGER overflow. An assignment or result exceeds the range allowed for INTEGER variables. Must be −32 768 thru 32 767.

22 REAL overflow. An assignment or result exceeds the range allowed for REAL variables.

24 Trig argument too large for accurate evaluation. Out-of-range argument for a function such as TAN or LDIR.

25 Magnitude of ASN or ACS argument is greater than 1. Arguments to these functions must be in the range −1 thru +1.

26 Zero to non-positive power. Exponentiation error.

27 Negative base to non-integer power. Exponentiation error.

28 LOG or LGT of a non-positive number.

29 Illegal floating point number. Does not occur as a result of any calculations, but is possible when a FORMAT OFF I/O operation fills a REAL variable with something other than a REAL number.

30 SQR of a negative number.

31 Division (or MOD) by zero.

32 String does not represent a valid number. Attempt to use "non-numeric" characters as an argument for VAL, data for a READ, or in response to an INPUT statement requesting a number.

33 Improper argument for NUM or RPT$. Null string not allowed.

34 Referenced line not an IMAGE statement. A USING clause contains a line identifier, and the line referred to is not an IMAGE statement.

35 Improper image. See IMAGE or the appropriate keyword in the *BASIC Language Reference.*

36 Out of data in READ. A READ statement is expecting more data than is available in the referenced DATA statements. Check for deleted lines, proper OPTION BASE, proper use of RESTORE, or typing errors.

38 TAB or TABXY not allowed here. The tab functions are not allowed in statements that contain a USING clause. TABXY is allowed only in a PRINT statement.

40 Improper REN, COPYLINES, or MOVELINES command. Line numbers must be whole numbers from 1 to 32 766. This may also result from a COPYLINES or MOVELINES statement whose destination line numbers lie within the source range.

41 First line number greater than second line number. Parameters out of order in a statement like SAVE, LIST, or DEL.

43 Matrix must be square. The MAT functions: IDN, INV, and DET require the array to have equal numbers of rows and columns.

44 Result cannot be an operand. Attempt to use a matrix as both result and argument in a MAT TRN or matrix multiplication.

46 Attempting a SAVE when there is no program in memory.

47 COM declarations are inconsistent or incorrect. Includes such things as mismatched dimensions, unspecified dimensions, and blank COM occurring for the first time in a subprogram.

49 Branch destination not found. A statement such as ON ERROR or ON KEY refers to a line that does not exist. Branch destinations must be in the same context as the ON...statement.

51 File not currently assigned. Attempting an ON/OFF END statement with an unassigned I/O path name.

52 Improper mass storage unit specifier. The characters used for a msus do not form a valid specifier. This could be a missing colon, too many parameters, illegal characters, etc.

53 Improper file name. File names are limited to 10 characters. Foreign characters are allowed, but punctuation is not.

54 Duplicate file name. The specified file name already exists in directory. It is illegal to have two files with the same name on one volume.

55 Directory overflow. Although there may be room on the media for the file, there is no room in the directory for another file name. Discs initialized by BASIC have room for over 100 entries in the directory, but other systems may make a directory of a different size.

56 File name is undefined. The specified file name does not exist in the directory. Check the contents of the disc with a CAT command.

58 Improper file type. Many mass storage operations are limited to certain file types. For example, LOAD is limited to PROG files and ASSIGN is limited to ASCII and BDAT files.

59 End of file or buffer found. For files: No data left when reading a file, or no space left when writing a file. For buffers: No data left for an ENTER, or no buffer space left for an OUTPUT. Also, WORD-mode TRANSFER terminated with odd number of bytes.

60 End of record found in random mode. Attempt to ENTER a field that is larger than a defined record.

62 Protect code violation. Failure to specify the protect code of a protected file, or attempting to protect a file of the wrong type.

64 Mass storage media overflow. There is not enough contiguous free space for the specified file size. The disc is full.

65 Incorrect data type. The array used in a graphics operation, such as GLOAD, is the wrong type (INTEGER or REAL).

66 INITIALIZE failed. Too many bad tracks found. The disc is defective, damaged, or dirty.

67 Illegal mass storage parameter. A mass storage statement contains a parameter that is out of range, such as a negative record number or an out of range number of records.

68 Syntax error occurred during GET. One or more lines in the file could not be stored as valid program lines. The offending lines are usually listed on the system printer. Also occurs if the first line in the file does not start with a valid line number.

72 Disc controller not found or bad controller address. The msus contains an improper device selector, or no external disc is connected.

73 Improper device type in mass storage unit specifier. The msus has the correct general form, but the characters used for a device type are not recognized.

76 Incorrect unit number in mass storage unit specifier. The msus contains a unit number that does not exist on the specified device.

77 Attempt to purge an open file. The specified file is assigned to an I/O path name which has not been closed.

78 Invalid mass storage volume label. Usually indicates that the media has not been initialized on a compatible system. Could also be a bad disc.

79 File open on target device. Attempt to copy an entire volume with a file open on the destination disc.

80 Disc changed or not in drive. Either there is no disc in the drive or the drive door was opened while a file was assigned.

81 Mass storage hardware failure. Also occurs when the disc is pinched and not turning. Try reinserting the disc.

82 Mass storage unit not present. Hardware problem or an attempt to access a left-hand drive on the Model 226.

83 Write protected. Attempting to write to a write_protected disc. This includes many operations such as PURGE, INITIALIZE, CREATE, SAVE, OUTPUT, etc.

84 Record not found. Usually indicates that the media has not been initialized.

85 Media not initialized. (Usually not produced by the internal drive.)

87 Record address error. Usually indicates a problem with the media.

88 Read data error. The media is physically or magnetically damaged, and the data cannot be read.

89 Checkread error. An error was detected when reading the data just written. The media is probably damaged.

90 Mass storage system error. Usually a problem with the hardware or the media.

93 Incorrect volume code in MSUS. The MSUS contains a volume number that does not exist on the specified device.

100 Numeric IMAGE for string item.

101 String IMAGE for numeric item.

102 Numeric field specifier is too large. Specifying more than 256 characters in a numeric field.

103 Item has no corresponding IMAGE. The image specifier has no fields that are used for item processing. Specifiers such as # X / are not used to process the data for the item list. Item-processing specifiers include things like K D B A.

105 Numeric IMAGE field too small. Not enough characters are specified to represent the number.

106 IMAGE exponent field too small. Not enough exponent characters are specified to represent the number.

107 IMAGE sign specifier missing. Not enough characters are specified to represent the number. Number would fit except for the minus sign.

117 Too many nested structures. The nesting level is too deep for such structures as FOR, SELECT, IF, LOOP, etc.

118 Too many structures in context. Refers to such structures as FOR/NEXT, IF/THEN/ELSE, SELECT/CASE, WHILE, etc.

120 Not allowed while program running. The program must be stopped before you can execute this command.

121 Line not in main program. The run line specified in a LOAD or GET is not in the main context.

122 Program is not continuable. The program is in the stopped state, not the paused state. CONT is allowed only in the paused state.

126 Quote mark in unquoted string. Quote marks must be used in pairs.

127 Statements which affect the knob mode are out of order.

128 Line too long during GET.

131 Unrecognized non-ASCII keycode. An output to the keyboard contained a CHR$(255) followed by an illegal byte.

132 Keycode buffer overflow. Trying to send too many characters to the keyboard buffer with an OUTPUT 2 statement.

133 DELSUB of non-existent or busy subprogram.

134 Improper SCRATCH statement.

135 READIO/WRITEIO to nonexistent memory location.

136 REAL underflow. The input or result is closer to zero than 10^{-308} (approximately).

140 Too many symbols in the program. Symbols are variable names, I/O path names, COM block names, subprogram names, and line identifiers.

141 Variable cannot be allocated. It is already allocated.

142 Variable not allocated. Attempt to DEALLOCATE a variable that was not allocated.

143 Reference to missing OPTIONAL parameter. The subprogram is trying to use an optional parameter that didn't have any value passed to it. Use NPAR to check the number of passed parameters.

145 May not build COM at this time. Attempt to add or change COM when a program is running. For example, a program does a LOADSUB and the COM in the new subprogram does not match existing COM.

146 Duplicate line label in context. There cannot be two lines with the same line label in one context.

150 Illegal interface select code or device selector. Value out of range.

152 Parity error.

153 Insufficient data for ENTER. A statement terminator was received before the variable list was satisfied.

154 String greater than 32 767 bytes in ENTER.

155 Improper interface register number. Value out of range or negative.

156 Illegal expression type in list. For example, trying to ENTER into a constant.

157 No ENTER terminator found. The variable list has been satisfied, but no statement terminator was received in the next 256 characters. The # specifier allows the statement to terminate when the last item is satisfied.

158 Improper image specifier or nesting images more than 8 deep. The characters used for an image specifier are improper or in an improper order.

159 Numeric data not received. When entering characters for a numeric field, an item terminator was encountered before any "numeric" characters were received.

160 Attempt to enter more than 32 767 digits into one number.

163 Interface not present. The intended interface is not present, set to a different select code, or is malfunctioning.

164 Illegal BYTE/WORD operation. Attempt to ASSIGN with the WORD attribute to a non-word device.

165 Image specifier greater than dimensioned string length.

167 Interface status error. Exact meaning depends upon the interface type. With HP-IB, this can happen when a non-controller operation by the computer is aborted by the bus.

168 Device timeout occurred and the ON TIMEOUT branch could not be taken.

170 I/O operation not allowed. The I/O statement has the proper form, but its operation is not defined for the specified device. For example, using an HP-IB statement on a non-HP-IB interface or directing a LIST to the keyboard.

171 Illegal I/O addressing sequence. The secondary addressing in a device selector is improper or primary address too large for specified device.

172 Peripheral error. PSTS line is false. If used, this means that the peripheral device is down. If PSTS is not being used, this error can be suppressed by using control register 2 of the GPIO.

173 Active or system controller required. The HP-IB is not active controller and needs to be for the specified operation.

174 Nested I/O prohibited. An I/O statement contains a user-defined function. Both the original statement and the function are trying to access the same file or device.

177 Undefined I/O path name. Attempting to use an I/O path name that is not assigned to a device or file.

178 Trailing punctuation in ENTER. The trailing comma or semicolon that is sometimes used at the end of OUTPUT statements is not allowed at the end of ENTER statements.

301 Cannot do while connected.

303 Not allowed when trace active.

304 Too many characters without terminator.

306 Interface card failure. The datacomm card has failed self-test.

308 Illegal character in data. Datacomm error.

310 Not connected. Datacomm error.

313 USART receive buffer overflow. Overrun error detected. Interface card is unable to keep up with incoming data rate. Data has been lost.

314 Receive buffer overflow. Program is not accepting data fast enough to keep up with incoming data rate. Data has been lost.

315 Missing data transmit clock. A transmit timeout has occurred because a missing data clock prevented the card from transmitting. The card has disconnected from the line.

316 CTS false too long. The interface card was unable to transmit for a predetermined period of time because Clear-To-Send was false on a half-duplex line. The card has disconnected from the line.

317 Lost carrier disconnect. Data Set Ready (DSR) or Data Carrier Detect (if full duplex) went inactive for too long.

318 No activity disconnect. The card has disconnected from the line because no data was transmitted or received for a predetermined length of time.

319 Connection not established. Data Set Ready or Data Carrier Detect (if full duplex) did not become active within a predetermined length of time.

324 Card trace buffer overflow.

325 Illegal databits/parity combination. Attempting to program 8 bits-per-character and a parity of "1" or "0".

326 Register address out of range. A control or status register access was attempted to a non-existent register.

327 Register value out of range. Attempting to place an illegal value in a control register.

328 USART Transmit underrun.

330 User-defined LEXICAL ORDER IS table size exceeds array size.

331 Repeated value in pointer. A MAT REORDER vector has repeated subscripts. This error is not always caught.

332 Non-existent dimension given. Attempt to specify a non-existent dimension in a MAT REORDER operation.

333 Improper subscript in pointer. A MAT REORDER vector specifies a non-existent subscript.

334 Pointer size is not equal to the number of records. A MAT REORDER vector has a different number of elements than the specified dimension of the array.

335 Pointer is not a vector. Only single-dimension arrays (vectors) can be used as the pointer in a MAT REORDER or a MAT SORT statement.

337 Substring key is out-of-range. The specified substring range of the sort key exceeds the dimensioned length of the elements in the array.

338 Key subscript out-of-range. Attempt to specify a subscript in a sort key outside the current bounds of the array.

340 Mode table too long. User-defined LEXICAL ORDER IS mode table contains more than 63 entries.

341 Improper mode indicator. User-defined LEXICAL ORDER IS table contains an illegal combination of mode type and mode pointer.

342 Not a single-dimension integer array. User-defined LEXICAL ORDER IS mode table must be a single-dimension array of type INTEGER.

343 Mode pointer is out of range. User-defined LEXICAL ORDER IS table has a mode pointer greater than the existing mode table size.

344 1 for 2 list empty or too long. A user-defined LEXICAL ORDER IS table contains an entry indicating an improper number of 1 for 2 secondaries.

345 CASE expression type mismatch. The SELECT statement and its CASE statements must refer to the same general type, numeric or string.

346 INDENT parameter out-of-range. The parameters must be in the range: 0 thru eight characters less than the screen width.

347 Structures improperly matched. There is not a corresponding number of structure beginnings and endings. Usually means that you forgot a statement such as END IF, NEXT, END SELECT, etc.

349 CSUB has been modified. A contiguous block of compiled subroutines has been modified since it was loaded. A single module that shows as multiple CSUB statements has been altered because program lines were inserted or deleted.

353 Data link failure.

369-399 Errors in this range are reported if a run-time Pascal error occurs in a CSUB. To determine the Pascal error number, subtract 400 from the BASIC error number. Information on the Pascal error can be found in the *Pascal Workstation System* manual.

401 Bad system function argument. An invalid argument was given to a time, date, base conversion, or SYSTEM$ function.

403 Copy failed; program modification incomplete. An error occurred during a COPYLINES or MOVELINES resulting in an incomplete operation. Some lines may not have been copied or moved.

427 Priority may not be lowered.

450 Volume not found—SRM error.

451 Volume labels do not match—SRM error.

453 File in use—SRM error.

454 Directory formats do not match—SRM error.

455 Possibly corrupt file—SRM error.

456 Unsupported directory operation—SRM error.

457 Passwords not supported —SRM error.

458 Unsupported directory format—SRM error.

459 Specified file is not a directory—SRM error.

460 Directory not empty—SRM error.

462 Invalid password—SRM error.

465 Invalid rename across volumes—SRM error.

471 TRANSFER not supported by the interface.

481 File locked or open exclusively—SRM error.

482 Cannot move a directory with a RENAME operation—SRM error.

483 System down—SRM error.

484 Password not found—SRM error.

485 Invalid volume copy—SRM error.

488 DMA hardware required. HP 9885 disc drive requires a DMA card or is malfunctioning.

511 The result array in a MAT INV must be of type REAL.

600 Attribute cannot be modified. The WORD/BYTE mode cannot be changed after assigning the I/O path name.

601 Improper CONVERT lifetime. When the CONVERT attribute is included in the assignment of an I/O path name, the name of a string variable containing the conversion is also specified. The conversion string must exist as long as the I/O path name is valid.

602 Improper BUFFER lifetime. The variable designated as a buffer during an I/O path name assignment must exist as long as the I/O path name is valid.

603 Variable was not declared as a BUFFER. Attempt to assign a variable as a buffer without first declaring the variable as a BUFFER.

604 Bad source or destination for a TRANSFER statement. Transfers are not allowed to the CRT, keyboard, or tape backup on CS80 drives. Buffer to buffer or device to device transfers are not allowed.

605 BDAT file type required. Only BDAT files can be used in a TRANSFER operation.

606 Improper TRANSFER parameters. Conflicting or invalid TRANSFER parameters were specified, such as RECORDS without and EOR clause, or DELIM with an outbound TRANSFER.

607 Inconsistent attributes. Such as CONVERT or PARITY with FORMAT OFF.

609 IVAL or DVAL result too large. Attempt to convert a binary, octal, decimal, or hexadecimal string into a value outside the range of the function.

612 BUFFER pointers in use. Attempt to change one or more buffer pointers while a TRANSFER is in progress.

700 Improper plotter specifier. The characters used as a plotter specifier are not recognized. May be misspelled or contain illegal characters.

702 CRT graphics hardware missing. Hardware problem.

704 Upper bound not greater than lower bound. Applies to P2<=P1 or VIEWPORT upper bound and CLIP limits.

705 VIEWPORT or CLIP beyond hard clip limits.

708 Device not initialized.

713 Request not supported by specified device. Trying to equate color CRT characteristics with an external device; such as COLOR MAP on a plotter.

733 GESCAPE opcode not recognized. Only values 1 thru 5 can be used.

900 Undefined typing aid key.

901 Typing aid memory overflow.

902 Must delete entire context. Attempt to delete a SUB or DEF FN statement without deleting its entire context. Easiest way to delete is with DELSUB.

903 No room to renumber. While EDIT mode was renumbering during an insert, all available line numbers were used between insert location and end of program.

904 Null FIND or CHANGE string.

905 CHANGE would produce a line too long for the system. Maximum line length is two lines on the CRT.

906 SUB or DEF FN not allowed here. Attempt to insert a SUB or DEF FN statement into the middle of a context. Subprograms must be appended at the end.

909 May not replace SUB or DEF FN. Similar to deleting a SUB or DEF FN.

910 Identifier not found in this context. The keyboard-specified variable does not already exist in the program. Variables cannot be created from the keyboard; they must be created by running a program.

911 Improper I/O list.

920 Numeric constant not allowed.

921 Numeric identifier not allowed.

922 Numeric array element not allowed.

923 Numeric expression not allowed.

924 Quoted string not allowed.

925 String identifier not allowed.

926 String array element not allowed.

927 Substring not allowed.

928 String expression not allowed.

929 I/O path name not allowed.

930 Numeric array not allowed.

931 String array not allowed.

932 Excess keys specified. A sort key was specified following a key which specified the entire record.

935 Identifier is too long: 15 characters maximum.

936 Unrecognized character. Attempt to store a program line containing an improper name or illegal character.

937 Invalid OPTION BASE. Only 0 and 1 are allowed.

939 OPTIONAL appears twice. A parameter list may have only one OPTIONAL keyword. All parameters listed before it are required, all listed after it are optional.

940 Duplicate formal parameter name.

942 Invalid I/O path name. The characters after the @ are not a valid name. Names must start with a letter.

943 Invalid function name. The characters after the FN are not a valid name. Names must start with a letter.

946 Dimensions are inconsistent with previous declaration. The references to an array contain a different number of subscripts at different places in the program.

947 Invalid array bounds. Value out of range, or more than 32 767 elements specified.

948 Multiple assignment prohibited. You cannot assign the same value to multiple variables by stating X=Y=Z=0. A separate assignment must be made for each variable.

949 This symbol not allowed here. This is the general "syntax error" message. The statement you typed contains elements that don't belong together, are in the wrong order, or are misspelled.

950 Must be a positive integer.

951 Incomplete statement. This keyword must be followed by other items to make a valid statement.

961 CASE expression type mismatch. The CASE line contains items that are not the same general type, numeric or string.

962 Programmable only: cannot be executed from the keyboard.

963 Command only: cannot be stored as a program line.

977 Statement is too complex. Contains too many operators and functions. Break the expression down so that it is performed by two or more program lines.

980 Too many symbols in this context. Symbols include variable names, I/O path names, COM block names, subprogram names, and line identifiers.

982 Too many subscripts: maximum of six dimensions allowed.

983 Wrong type or number of parameters. An improper parameter list for a machine-resident function.

985 Invalid quoted string.

987 Invalid line number: must be a whole number 1 thru 32 766.

Keyword Summary

Program Entry/Editing

CHANGE	Performs search and replace operations while editing a program.
COPYLINES	Copies program lines from one place to another.
EDIT	Accesses a program edit mode to enter new program lines or modify existing ones. Also used with typing aids.
FIND	Searches for a character sequence in a program.
DEL	Deletes specified program lines from memory.
DELSUB	Deletes specified subprograms from memory.
INDENT	Indents a program to reflect its structure.
LIST	Lists program lines or typing aids.
LIST BIN	Lists options in the system.
MOVELINES	Moves program lines from one place to another.
REM and !	Allows comments on program lines.
REN	Renumbers programs.
SECURE	Makes program lines unlistable.
XREF	Provides a cross-reference to all identifiers used in a program.

Program Debugging

ERRDS	Returns the device selector involved in the last I/O error.
ERRL	Indicates if an error occurred during execution of a specified line.
ERRM$	Returns the text of the last error message.
ERRN	Returns the most recent program execution error.
TRACE ALL	Allows tracing of program flow and variable assignments during program execution.
TRACE PAUSE	Causes program execution to pause at a specified line.
TRACE OFF	Disables TRACE ALL and TRACE PAUSE.

Memory Allocation

ALLOCATE	Allocates memory for arrays or string variables during program execution.
COM	Reserves memory for variables in a common area for access by more than one context.
DEALLOCATE	Reclaims memory previously allocated.
DIM	Dimensions and reserves memory for REAL numeric arrays and strings.
INTEGER	Dimensions and reserves memory for INTEGER variables and arrays.
OPTION BASE	Specifies the default lower bound for arrays.
REAL	Dimensions and reserves memory for full precision variables and arrays.
SCRATCH	Erases selected portions of memory.
SYSBOOT	Returns system control to the boot ROM.

General Math

+	Addition operator.
–	Subtraction operator.
*	Multiplication operator.
/	Division operator.
↑	Exponentiation operator.
ABS	Returns an argument's absolute value.
DIV	Divides one argument by another and returns the integer portion of the quotient.
DROUND	Returns the value of an expression, rounded to a specified number of digits.
EXP	Raises the base e to a specified power.
FRACT	Returns the fractional portion of an expression.
INT	Returns the integer portion of an expression.
LET	Assigns values to variables.
LGT	Returns the log (base 10) of an argument.
LOG	Returns the natural logarithm (base e) of an argument.

MAX	Returns the largest value in a list of arguments.
MAXREAL	Returns the largest number available.
MIN	Returns the smallest value in a list of arguments.
MINREAL	Returns the smallest number available.
MOD	Returns the remainder of integer division.
MODULO	Return the molulo of division.
PI	Returns an approximation of π.
PROUND	Returns the value of an expression, rounded to the specified power of ten.
RANDOMIZE	Modifies the seed used by the RND function.
RES	Returns last live keyboard numeric result.
RND	Returns a pseudo-random number.
SGN	Returns the sign of an argument.
SQR	Returns the square root of an argument.

Binary Functions

BINAND	Returns the bit-by-bit logical-and of two arguments.
BINCMP	Returns the bit-by-bit complement of an argument.
BINEOR	Returns the bit-by-bit exclusive-or of two arguments.
BINIOR	Returns the bit-by-bit inclusive-or of two arguments.
BIT	Returns the state of a specified bit of an argument.
ROTATE	Returns a value obtained by shifting an argument's binary representation a number of bit positions, with wrap-around.
SHIFT	Returns a value obtained by shifting an argument's binary representation a number of bit positions, without wrap-around.

Trigonometric Operations

ACS	Returns the arccosine of an argument.
ASN	Returns the arcsine of an argument.
ATN	Returns the arctangent of an argument.
COS	Returns the cosine of an angle.
DEG	Sets the degrees mode.
RAD	Sets the radians mode.
SIN	Returns the sine of an angle.
TAN	Returns the tangent of an angle.

String Operations

&	Concatenates two string expressions.
CHR$	Converts a numeric value into an ASCII character.
DVAL	Converts an alternate-base representation into a numeric value.
DVAL$	Converts a numeric value into an alternate-base representation.
IVAL	Converts an alternate-base representation into an INTEGER number.
IVAL$	Converts an INTEGER into an alternate-base representation.
LEN	Returns the number of characters in a string expression.
LEXICAL ORDER IS	Determines the collating sequence used in string comparisons.
LWC$	Returns the lowercase value of a string expression.
NUM	Returns the decimal value of the first character in a string.
POS	Returns the position of a string within a string expression.
REV$	Reverses the order of the characters in a string expression.
RPT$	Repeats the characters in a string expression a specified number of times.
TRIM$	Removes the leading and trailing blanks from a string expression.
UPC$	Returns the uppercase value of a string expression.
VAL	Converts a string of numerals into a numeric value.
VAL$	Returns a string expression representing a specified numeric value.

Logical Operators

AND	Returns 1 or 0 based on the logical AND of two arguments.
EXOR	Returns 1 or 0 based on the logical exclusive-or of two arguments.
NOT	Returns 1 or 0 based on the logical complement of an argument.
OR	Returns 1 or 0 based on the logical inclusive-or of two arguments.

Comparison Operators

=	Equality.
< >	Inequality.
<	Less than.
<=	Less than or equal to.
>	Greater than.
>=	Greater than or equal to.

Mass Storage

ASSIGN	Assigns an I/O path name and attributes to a file.
CAT	Lists the contents of the mass storage media's directory.
CHECKREAD	Enables or disables read-after-write verification of a mass storage operation.
COPY	Provides a method of copying mass storage files and volumes.
CREATE ASCII	Creates an ASCII-type file on the mass storage media.
CREATE BDAT	Creates a BDAT-type file on the mass storage media.
CREATE DIR	Creates an SRM directory file.
GET	Reads an ASCII file into memory as a program.
INITIALIZE	Prepares the mass storage media for use.
LOAD	Loads a PROG-type file into memory.
LOAD BIN	Loads a BIN-type file into memory.
LOAD KEY	Loads typing-aid definitions for the softkeys.
LOADSUB	Loads BASIC subprograms from a PROG-type file into memory.
LOCK	Prevents other SRM workstation computers from accessing the remote file to which the I/O path is currently assigned.
MASS STORAGE IS MSI	Specifies the default mass storage device.
PRINT LABEL	Writes a string expression to the label of a media.
PROTECT	Specifies a protect code for PROG, BDAT, and BIN files.
PURGE	Deletes a file entry from the directory.
READ LABEL	Reads the label of a media to a string variable.
RENAME	Changes a file's name.
SAVE RE-SAVE	Creates an ASCII file and copies BASIC program lines from memory into the file.
STORE RE-STORE	Creates a PROG file and copies BASIC program lines from memory into the file in an internal format.
STORE KEY RE-STORE KEY	Creates a BDAT file and stores the typing-aid definitions into the file.
STORE SYSTEM	Stores BASIC and options currently in memory into a SYSTM file.
UNLOCK	Removes exclusive access to a remote (SRM) file set by the LOCK statement.

Program Control

CALL	Transfers program execution to a specified subprogram and passes parameters.
CONT	Resumes execution of a paused program.
DEF FN FNEND	Defines the bounds of a user-defined function subprogram.
END	Terminates program execution and marks the end of the main program segment.
FN	Invokes a user-defined function.
FOR...NEXT	Defines a loop which is repeated a specified number of times.
GOTO	Transfers program execution to a specified line.
GOSUB	Transfers program execution to a specified subroutine.
IF...THEN ELSE	Provides a conditional execution of a program segment.
LOOP EXIT IF	Provides looping with conditional exit.
NPAR	Returns the number of parameters passed to the current subprogram.
KBD$	Returns the contents of the ON KBD buffer.
KNOBX	Returns the number of horizontal knob pulses.
KNOBY	Returns the number of vertical knob pulses.
ON expression	Transfers program execution to one of several locations based on the value of an expression.
PAUSE	Suspends program execution.
REPEAT...UNTIL	Allows execution of a program segment until the specified condition is true.
RETURN	Transfers program execution from a subroutine to the line following the invoking GOSUB.
RETURN expression	Transfers program execution from a user-defined function by returning a value to the calling context.

RUN	Starts program execution.
SELECT...CASE	Allows execution of one program segment of several.
STOP	Terminates execution of the program.
SUB SUBEND	Defines the bounds of a subprogram.
SUBEXIT	Transfers control from within a subprogram to the calling context.
SUSPEND/ RESUME INTERACTIVE	Allows suspending and resuming interactive keyboard operation while a program is running.
SYSTEM$	Returns selected system status and configuration information.
WAIT	Causes program execution to wait a specified number of seconds.
WAIT FOR EOR	Causes program execution to wait for an end-of-record during a TRANSFER.
WAIT FOR EOT	Causes program execution to wait for an end-of-transfer.
WHILE	Allows execution of a program segment while the specified condition is true.

Graphics Control

ALPHA ON/OFF	Turns the alpha display on or off.
AREA	Selects an area fill color.
CLIP	Redefines a soft-clip area.
DIGITIZE	Inputs the coordinates of a digitized point.
DUMP GRAPHICS	Copies the contents of the graphics display to a printing device.
DUMP DEVICE IS	Specifies the device for DUMP operations.
GCLEAR	Clears the graphics area.
GESCAPE	Sends device-dependent information to the display device.
GINIT	Resets graphics parameters to power-on values.
GLOAD	Loads the graphics display from an INTEGER array.
GRAPHICS ON/OFF	Turns the graphics display on or off.
GRAPHICS INPUT IS	Specifies the device for digitizing operations.
GSTORE	Copies the contents of the graphics display to an INTEGER array.
PLOTTER IS	Specifies the default plotting device.
RATIO	Returns the physical aspect ratio of the plotter's hard-clip limits.
READ LOCATOR	Samples the locator device, without waiting for a digitize signal.
SET ECHO	Specifies the coordinates of an echo on the current plotting device.
SET LOCATOR	Sets the locator position on the input device.
SET PEN	Defines the color of entries in the color map.
SHOW	Defines plotting units that will appear in the VIEWPORT area.
TRACK...ON/OFF	Enables and disables locator tracking on the current display device.
VIEWPORT	Specifies an area in which WINDOW and SHOW statements are mapped.
WHERE	Returns the current logical position of the pen.
WINDOW	Specifies the min and max values for the plotting area specified by VIEWPORT.

Graphics Plotting

DRAW	Draws a line to a specified point.
LINE TYPE	Selects a plotting line type.
IDRAW	Draws a line incrementally to a specified point.
IMOVE	Moves the pen incrementally to a specified point.
IPLOT	Draws a line incrementally to the specified point with optional pen control.
MOVE	Moves the pen to a specified point.
PDIR	Specifies rotation for IPLOT, RPLOT, RECTANGLE, POLYGON and POLYLINE.
PEN	Selects a plotter pen.
PENUP	Lifts the pen from the plotting surface.
PIVOT	Specifies rotation for lines made with moves, draws, plots, polygons, or rectangles.
PLOT	Draws a line to the specified point with optional pen control.
POLYGON	Draws all or part of a closed polygon.
POLYLINE	Draws all or part of an open polygon.
RECTANGLE	Draws a rectangle that can be filled and edged.
RPLOT	Draws a line relative to a movable origin with optional pen control.

Graphic Axes and Labeling

AXES	Draws axes with optional tick marks.
CSIZE	Sets the size and aspect ratio for labeled characters.
FRAME	Draws a frame around the current clipping area.
GRID	Draws a full grid pattern for axes.
LABEL	Draws alphanumeric labels.
LDIR	Defines the angle for drawing labels.
LORG	Specifies a labeling location relative to the pen location.
SYMBOL	Allows labeling with user-defined symbols.

Input/Output

ABORTIO	Terminates an active TRANSFER.
ASSIGN	Associates an I/O path name and attributes with a device, group of devices, mass storage file, or buffer.
BEEP	Produces one of 63 audible tones.
BREAK	Sends a Break signal on a serial interface.
CONTROL	Sends control information to an interface or a table associated with an I/O path name.
CRT	Returns the device selector of the CRT.
DATA	Specifies data accessible via READ statements.
DISP	Outputs items to the CRT display line.
DUMP ALPHA	Transfers contents of the CRT output area to a specified device.
DUMP DEVICE IS	Specifies a device for DUMP ALPHA and DUMP GRAPHICS operations.
ENTER	Inputs data from a device, file, string, or buffer to a list of variables.
IMAGE	Provides formats for use with ENTER, OUTPUT, DISP,LABEL and PRINT operations.
INPUT	Inputs data from the keyboard to a list of variables.
KBD	Returns the device selector of the keyboard.
LINPUT	Inputs literal data from the keyboard to a string variable.
OUTPUT	Outputs items to a specified device, file, string, or buffer.
PRINT	Outputs items to the current PRINTER IS device.
PRINTALL IS	Specifies a device for logging messages sent to the display.
PRINTER IS	Specifies a device for PRINT, CAT and LIST statements.
PRT	Returns 701, usually the device selector of an external printer.
READ	Inputs data from DATA lists to variables.
READIO	Reads the contents of the specified hardware registers on the specified interface.
RESET	Resets an interface or pointers of an I/O path.
RESTORE	Causes a READ statement to access the specified DATA statement.
SC	Returns the interface select code associated with an I/O path.
STATUS	Returns the value from a specified interface status register.
TAB	Moves the print position ahead to a specified point; used within PRINT and DISP statements.
TABXY	Specifies the print position on the internal CRT; used with PRINT statements.
TRANSFER	Initiates unformatted I/O transfers.
WRITEIO	Writes an integer representation of the register data to the specified hardware register on the specified interface.

HP-IB Control

ABORT	Terminates bus activity and asserts IFC.
CLEAR	Places specified devices in a device-dependent state.
LOCAL	Returns specified devices to their local state.
LOCAL LOCKOUT	Sends the LLO message, disabling all device's front-panel controls.
PASS CONTROL	Passes Active Controller capability to another device.
PPOLL	Returns a parallel poll byte from the bus.
PPOLL CONFIGURE	Programs a parallel poll bit for a specified device.
PPOLL RESPONSE	Defines the computers response to a parallel poll.
PPOLL UNCONFIGURE	Disables parallel poll for specified devices.
REMOTE	Sets specified devices to their remote state.
REQUEST	Sends a service request to the Active Controller.
SEND	Sends explicit command and data messages on the bus.
SPOLL	Returns a serial poll byte from a specified device.
TRIGGER	Sends the trigger message to specified devices.

Array Operations

BASE	Returns the lower bound of a dimension of an array.
DET	Returns the determinant of a matrix.
DOT	Returns the dot product of two vectors.
MAT	Performs various operations on numeric and string arrays.
MAT REORDER	Reorders the elements in an array according to the subscript list in a vector.
MAT SORT	Sorts an array along one dimension according to lexical or numeric order.
RANK	Returns the number of dimensions in an array.
REDIM	Changes the subscript range of an array.
SIZE	Returns the number of elements in a dimension of an array.
SUM	Returns the sum of all the elements in a numeric array.

Clock and Calendar

DATE	Converts a formatted date into a number of seconds.
DATE$	Converts a number of seconds into a formatted date.
SET TIME	Sets the time of day on the real-time clock
SET TIMEDATE	Sets the time and date on the real-time clock.
TIME	Converts a formatted time of day into a number of seconds past midnight.
TIME$	Converts a number of seconds past midnight into a formatted time of day.
TIMEDATE	Returns the value of the real-time clock.

Event-Initiated Branching

ENABLE DISABLE	Enables or disables event-initiated branching (except for ON END, ON ERROR, and ON TIMEOUT).
ENABLE INTR DISABLE INTR	Enables or disables interrupts defined by the ON INTR statement.
ON CYCLE OFF CYCLE	Sets up an event-initiated branch at periodic intervals.
ON DELAY OFF DELAY	Sets up an event-initiated branch after a specified elapsed time.
ON END OFF END	Sets up an event-initiated branch when an end-of-file condition occurs.
ON EOR OFF EOR	Sets up an event-initiated branch when an end-of-record occurs during a TRANSFER.
ON EOT OFF EOT	Sets up an event-initiated branch when an end-of-transfer occurs.
ON ERROR OFF ERROR	Sets up an event-initiated branch when a trappable program error occurs.
ON INTR OFF INTR	Sets up an event-initiated branch when a specified interface card generates an interrupt.
ON KBD OFF KBD	Sets up an event-initiated branch when a key is pressed.
ON KEY...LABEL OFF KEY	Sets up an event-initiated branch when a specified softkey is pressed.
ON KNOB OFF KNOB	Sets up an event-initiated branch when the knob (cursor wheel) is rotated.
ON SIGNAL OFF SIGNAL	Sets up an event-initiated branch when a software interrupt is generated.
ON TIME OFF TIME	Sets up an event-initiated branch when a specified time of day is reached.
ON TIMEOUT OFF TIMEOUT	Sets up an event-initiated branch when an I/O timeout occurs on a specified interface.
SIGNAL	Generates a software interrupt.
SYSTEM PRIORITY	Sets a minimun level of system priority for event-initiated branches.

Vocabulary

The following list contains all the words which are recognized by Series 200/300 computers with BASIC 4.0. Each individual word is some part of one or more valid statements or functions. These words cannot be used as variable names unless you mix their letter case.

ABORT
ABORTIO
ABS
ACS
ALL
ALLOCATE
ALPHA
AND
AREA
ASCII
ASN
ASSIGN
ATN
AXES

BASE
BDAT
BEEP
BIN
BINAND
BINCMP
BINEOR
BINIOR
BIT
BREAK
BUFFER
BY
BYTE

CALL
CASE
CAT
CHANGE
CHECKREAD
CHR$
CLEAR
CLIP
CM
CMD
COLOR
COM
CONFIGURE
CONT
CONTROL
CONVERT
COPY
COPYLINES
COS
COUNT
CREATE
CRT
CSIZE
CSUM
CYCLE

DATA
DATE
DATE$
DEALLOCATE
DEF
DEG
DEL
DELAY
DELETE
DELIM
DELSUB
DES
DET
DEVICE
DIGITIZE
DIM
DIR
DISABLE
DISP
DIV
DOT
DRAW
DROUND
DUMP
DVAL
DVAL$

ECHO
EDGE
EDIT
ELSE
ENABLE
END
ENTER
EOL
EOR
EOT
ERRDS
ERRL
ERRM$
ERRN
ERROR
EVEN
EXIT
EXOR
EXP
EXPANDED

FILL
FIND
FN
FNEND
FOR
FORMAT
FRACT
FRAME
FRENCH
FROM

GCLEAR
GERMAN
GESCAPE
GET
GINIT
GLOAD
GO
GOSUB
GOTO
GRAPHICS
GRID
GSTORE

HEADER

IDN
IDRAW
IF
IMAGE
IMOVE
IN
INDENT
INDEX
INITIALIZE
INPUT
INT
INTEGER
INTENSITY
INTERACTIVE
INTR
INV
IO
IPLOT
IS
IVAL
IVAL$

KBD
KBD$
KEY
KNOB
KNOBX
KNOBY

LABEL
LDIR
LEN
LET
LEXICAL
LGT
LINE
LINPUT
LIST
LISTEN
LL
LN
LOAD
LOADSUB
LOCAL
LOCATE
LOCATOR
LOCK
LOCKOUT
LOG
LOOP
LORG
LWC$

MANAGER
MAP
MASS
MAT
MAX
MAXREAL
MIN
MINREAL
MLA
MOD
MODE
MODULO
MOVE
MOVELINES
MSI
MTA

NEXT
NF
NO
NOT
NPAR
NUM
NV

ODD
OFF
ON
ONE
OPTION
OPTIONAL
OR
ORDER
OUT
OUTPUT

PAIRS
PARITY
PASS
PAUSE
PDIR
PEN
PENUP
PI
PIVOT
PLOT
PLOTTER
POLYGON
POLYLINE
POS
PPOLL
PRINT
PRINTALL
PRINTER
PRIORITY
PROTECT
PROUND
PRT
PURGE

RAD
RANDOMIZE
RANK
RATIO
RE
READ
READIO
REAL
RECORDS
RECOVER
RECTANGLE
REDIM
REM
REMOTE
REN
RENAME
REORDER
REPEAT
REQUEST
RES
RE-SAVE
RESET
RESPONSE
RESTORE
RE-STORE
RESUME
RETURN
REV$
RND
ROTATE
RPLOT
RPT$
RSUM
RUN

SAVE
SB
SC
SCALE
SCRATCH
SEC
SECURE
SELECT
SEND
SET
SF
SGN
SHIFT
SHOW
SIGNAL
SIN
SIZE
SKIP
SORT
SPANISH
SPOLL
SQR
STANDARD
STATUS
STEP
STOP
STORAGE
STORE
SUB
SUBEND
SUBEXIT
SUM
SUSPEND
SV
SWEDISH
SYMBOL
SYSBOOT
SYSTEM
SYSTEM$

TAB
TABXY
TALK
TAN
THEN
TIME
TIME$
TIMEDATE
TIMEOUT
TO
TRACE
TRACK
TRANSFER
TRIGGER
TRIM$
TRN
TYPE

UN
UNCONFIGURE
UNL
UNLOCK
UNT
UNTIL
UPC$
USING

VAL
VAL$
VIEWPORT

WAIT
WHERE
WHILE
WIDTH
WINDOW
WORD
WRITE
WRITEIO

XREF

ZERO

NOTE 1: Although LOCATE and SCALE are recognized as reserved words when entered, they are stored and listed back as VIEWPORT and WINDOW, respectively.
NOTE 2: Although CSUB can appear as a reserved word in a program listing, it is not recognized as a reserved word when entered from the keyboard.

Manual Comment Sheet Instruction

If you have any comments or questions regarding this manual, write them on the enclosed comment sheets and place them in the mail. Include page numbers with your comments wherever possible.

If there is a revision number, (found on the Printing History page), include it on the comment sheet. Also include a return address so that we can respond as soon as possible.

The sheets are designed to be folded into thirds along the dotted lines and taped closed. Do not use staples.

Thank you for your time and interest.

MANUAL COMMENT SHEET

BASIC 4.0 Language Reference
for the HP 9000 Series 200/300

98613-90051

July 1985

Update No. _____

(See the Printing History in the front of the manual)

Name: _____

Company: _____

Address: _____

_____

Phone No: _____

fold fold

fold fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Fort Collins Systems Division
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525



Reorder Number
98613-90051
Printed in U.S.A. 7/85



98613-90654
Mfg. No. Only