



# **BASIC 5.0/5.1 Language Reference**

**Vol. 1: A-N**

**HP 9000 Series 200/300 Computers**

HP Part Number 98613-90052



**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.

January 1987...Edition 1

November 1987...Edition 2. This edition reflects the 5.0 corrections and 5.1 additions.

May 1988...Update. This edition contains new information regarding the 5.11 revision. (See SYSTEM\$, and STATUS/CONTROL register for cache memory at select code 32.)



**HP Computer Museum**  
**[www.hpmuseum.net](http://www.hpmuseum.net)**

**For research and education purposes only.**

# Table of Contents

---

## Volume 1

### Using the Keyword Dictionary

Legal Usage Table .....	1
Syntax Drawings Explained .....	2
Keywords and Spaces .....	3
Space Between Keywords and Names .....	3
No Spaces in Keywords or Reserved Groupings .....	3
Using Keyword Letters for a Name .....	3
Keyboards .....	4

### Keyword Dictionary A-N

## Volume 2

### Keyword Dictionary O-Z

### Appendix A: Language History

### Appendix B: Glossary

### Appendix C: Interface Registers

I/O Path Registers .....	C-1
Registers for All I/O Paths .....	C-1
I/O Path Names Assigned to a Device .....	C-1
I/O Path Names Assigned to an ASCII File .....	C-2
I/O Path Names Assigned to a BDAT File .....	C-2
I/O Path Names Assigned to an HP-UX File .....	C-3
I/O Path Names Assigned to a Buffer .....	C-3
Summary of CRT STATUS and CONTROL Registers .....	C-5
Summary of Keyboard Status and Control Registers .....	C-10
Summary of HP-IB Status and Control Registers .....	C-16
Summary of RS-232C Serial STATUS and CONTROL Registers .....	C-22
Overview of Datacomm Status and Control Registers .....	C-31
Summary of Datacomm Interface Status and Control Registers .....	C-33
Summary of Powerfail Status and Control Registers .....	C-47
Summary of GPIO STATUS and CONTROL Registers .....	C-50
Summary of BCD Status and Control Registers .....	C-53
Summary of EPROM Programmer STATUS and CONTROL Registers .....	C-57
Parity, Cache, Float, and Clock STATUS and CONTROL Registers (Pseudo Select Code 32) .....	C-59
SRM Interface STATUS Registers .....	C-61

## Appendix D: Useful Tables

Option Numbers .....	D-1
Interface Select Codes .....	D-2
Display-Enhancement Characters .....	D-3
Monochrome Enhancements .....	D-3
Color Enhancements .....	D-3
U.S. ASCII Character Codes .....	D-4
U.S./European Display Characters .....	D-6
Katakana Display Characters .....	D-12
Master Reset Table .....	D-16
Graphic Reset Table .....	D-19
Interface Reset Table .....	D-20
Second Byte of Non-ASCII Key Sequences (String) .....	D-22
Selected High-Precision Metric Conversion Factors .....	D-27

## Appendix E: Error Messages

## Appendix F: Keyword Summary

Booting the System .....	F-1
Program Entry/Editing .....	F-1
Program Debugging and Error Handling .....	F-2
Memory Allocation and Management .....	F-3
Comparison Operators .....	F-3
General Math .....	F-4
Complex Math .....	F-5
Binary Functions .....	F-5
Trigonometric Operations .....	F-5
Hyperbolic Operations .....	F-6
String Operations .....	F-6
Logical Operators .....	F-7
Mass Storage .....	F-7
Program Control .....	F-8
Event-Initiated Branching .....	F-10
HP-HIL Device Support .....	F-13
Graphics Control .....	F-13
Graphics Plotting .....	F-14
Graphic Axes and Labeling .....	F-15
HP-IB Control .....	F-15
Clock and Calendar .....	F-16
General Device Input/Output .....	F-16
Display and Keyboard Control .....	F-18
Array Operations .....	F-19
Vocabulary .....	F-20



# Table of Contents

---

## Using the Keyword Dictionary

Legal Usage Table .....	1
Syntax Drawings Explained .....	2
Keywords and Spaces .....	3
Space Between Keywords and Names .....	3
No Spaces in Keywords or Reserved Groupings .....	3
Using Keyword Letters for a Name .....	3
Keyboards .....	4





# 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	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN	No

Option Required	indicates what must be resident in the computer (other than BASIC Main) 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 <b>EXECUTE</b> , <b>ENTER</b> , or <b>Return</b> 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 for execution with an OUTPUT KBD statement. This is <b>not</b> 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.

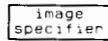
---

## 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.

```
100 PRINT "Hello" ! This is a comment.  
110 ! This is also a comment.
```

The text following the exclamation point may contain any characters in any order.

## 2 Using the Keyword Dictionary

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

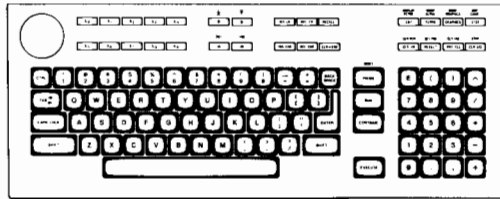
Attempting to store the line `IF X=1 THEN END` will generate an error because `END` is a keyword not allowed in an `IF...THEN`. To create a line label called “End”, type `IF X=1 THEN ENd`. This or any other mixture of uppercase and lowercase will prevent the name from being recognized as a keyword.

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`).

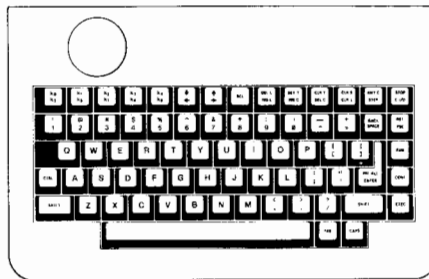
---

## Keyboards

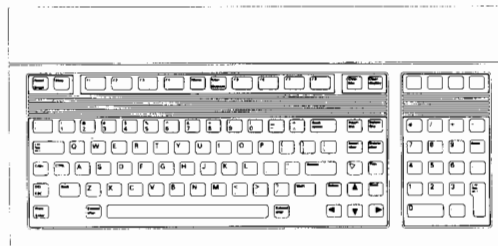
The Series 200/300 Computers support three keyboard styles:



**HP 98203B and 98203C**



**HP 98203A**



**ITF (such as the HP 46021A)**

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, **ENTER** and **Return** normally have the same meaning, but only one of them appears on any one keyboard. The manual entitled *Using the BASIC System* discusses the keyboards.

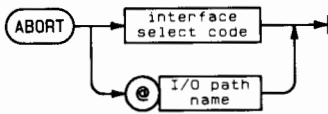
Within this manual, the keys for each keyboard are listed the first time they are used in a statement description. Thereafter, only one keyboard's keys are used.

### 4 Using the Keyword Dictionary

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



This statement ceases activity on the specified interface.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	5, 7 thru 31
I/O path name	name assigned to an HP-IB interface	—

## Example Statements

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

## Semantics

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 $\geq 100\mu\text{sec}$ ) REN ATN	Error	ATN MTA UNL ATN	Error
Not Active Controller	IFC (duration $\geq 100\mu\text{sec}$ )* REN 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	Range
I/O path name	name assigned to a device, a group of devices, or 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.

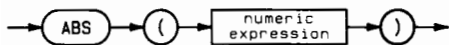
ABORTIO has no effect if no TRANSFER is taking place through the I/O path name.

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.



Item	Description/Default	Range Restrictions
argument	numeric expression	within valid ranges of INTEGER and REAL data types for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

## Examples Statements

```
Magnitude=ABS(Vector)  
PRINT "Value = ";ABS(CMPLX(2.45,-4))
```

## Semantics

To compute the absolute value of a COMPLEX value, the COMPLEX binary must be loaded.



### Range Restriction Specifics

The formula for computing ABS for COMPLEX arguments is:

```
SQRT(Real_part*Real_part + Imag_part*Imag_part)
```

where **Real\_part** is the real part of the COMPLEX argument and **Imag\_part** is the imaginary part of the COMPLEX argument in the ABS function. Some values of a COMPLEX argument may cause errors in this computation. For example:

```
ABS(CMPLX(MAXREAL,MAXREAL))
```

will cause error 22 due to the computation  $\text{Real\_part} * \text{Real\_part}$ .

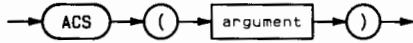
The result returned when executing the ABS function for COMPLEX numbers is always a positive REAL value.

Taking the ABS of the INTEGER  $-32768$  will cause an error.

# ACS

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

This function returns the principle 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 for INTEGER and REAL arguments; see "Range Restriction Specifics" below for COMPLEX arguments

## Examples Statements

```
Angle=ACS(Cosine)
PRINT "Angle = ";ACS(CMPLX(2.67,-6))
```

## Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

The angle mode (**RAD** or **DEG**) for REAL and INTEGER arguments indicates whether you should interpret the value returned in degrees or radians. If the current angle mode is DEG, the range of the result is 0° to 180°. If the current angle mode is RAD, the range of the result is 0 to  $\pi$  radians. The angle mode is radians unless you specify degrees with the DEG statement.

To compute the ACS of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula used for computing the ACS of a COMPLEX value is:

$$-i * \text{LOG}(\text{Argument} + \text{SQRT}(\text{Argument} * \text{Argument} - 1))$$

where  $i$  is the COMPLEX value `CMLX(0,1)` and `Argument` is a COMPLEX argument to the ACS function. Some values of a COMPLEX argument may cause errors in this computation. For example,

```
ACS(CMLX(MAXREAL,0))
```

will cause error 22 due to the `Argument*Argument` computation.

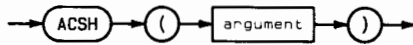
The principle value, which has a real part between 0 and  $\pi$ , is returned for COMPLEX arguments.



# ACSH

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

This function returns the hyperbolic arccosine of a numeric expression.



Item	Description/Default	Range Restrictions
argument	numeric expression	INTEGER and REAL arguments must be $\geq 1$ and $< 1.340\ 780\ 792\ 99\ E\ 154$ ; see "Range Restriction Specifics" for COMPLEX arguments

## Example Statements

```

Result=ACSH(5.7089)
PRINT "Hyperbolic Arccosine = ";ACSH(Expression)
  
```

## Semantics

If an INTEGER or REAL argument is given, this function returns a REAL value. If a COMPLEX argument is given, this function returns a COMPLEX value.

## Range Restriction Specifics

The formula for computing ACSH is as follows:

$$\text{LOG}(\text{Argument} + \text{SQRT}(\text{Argument} * \text{Argument} - 1))$$

where **Argument** is the argument to the ACSH function. Some values of an argument may cause errors in this computation. For example,

```
ACSH(MAXREAL)
```

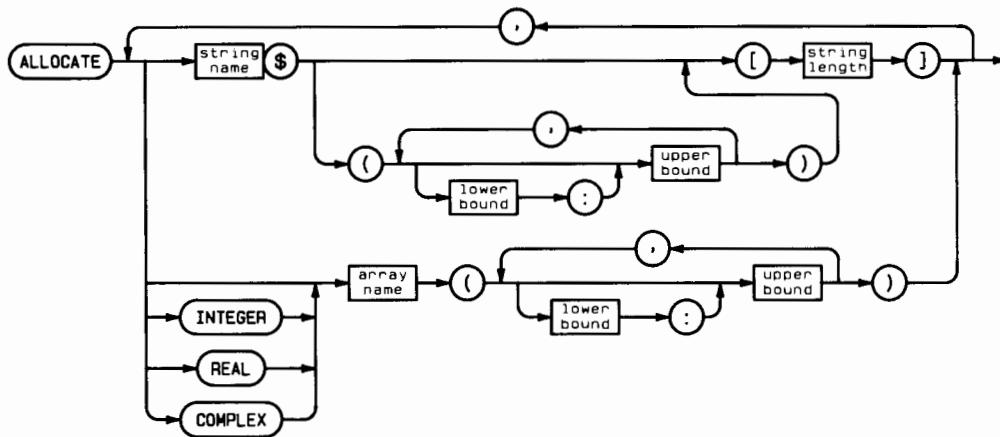
will cause error 22 due to the computation **Argument\*Argument**.

Note that the hyperbolic arccosine of a COMPLEX number returns a principle value which has an imaginary part which falls in the range of 0 to  $+\pi$ .

# 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	Range
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 available 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 availability of deallocated memory.

The total number of elements that can be allocated for variables within any one context or COM area (i.e., any “value area”) is limited to  $2^{24}-1$ , or 16 777 215, bytes.

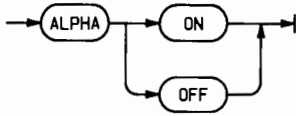
The variables in an ALLOCATE statement cannot have appeared in COM, COMPLEX, DIM, INTEGER, or REAL declaration statements. If variable(s) are to be allocated in a subprogram, the variable(s) cannot have been included in the subprogram’s formal parameter list. Implicitly declared variables cannot be allocated. Numeric variables which are not specified as INTEGER or COMPLEX 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.

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

This statement turns the alphanumeric raster on or off.



## Example Statements

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

## Semantics

Items sent to the printout area while the alphanumeric raster is disabled are placed in the display memory even though they are not visible. Items sent to the keyboard input line, the DISP line, or the system message line will turn on the alphanumeric raster. The alphanumeric and graphic rasters 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 raster.

## Bit-Mapped Alpha Displays

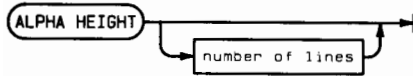
This statement has no effect on a bit-mapped alpha display when the alpha write-enable mask specifies all planes. This is the default state on those displays.

If ALPHA MASK  $\langle \rangle 2^n - 1$ , then planes enabled for alpha can be turned on and off. See SET ALPHA MASK in this reference for more information.

# ALPHA HEIGHT

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

This statement is used to vary the number of lines in the alpha area of the CRT.



Item	Description/Default	Range Restrictions
number of lines	numeric expression	(see Semantics)

## Example Statements

```
ALPHA HEIGHT Num_of_lines
ALPHA HEIGHT 18
IF Total_lines = 10 THEN ALPHA HEIGHT 18
ALPHA HEIGHT
```

## Semantics

ALPHA HEIGHT is used to restrict the alpha screen to the bottom  $n$  lines of the display, leaving the upper part of the display for graphics. This can be used to prevent alpha from interfering with graphics.

The number of lines available for alpha on the CRT depends on which display is being used. The following are the upper limits for the ALPHA HEIGHT statement: 25, 26, and 48. The lower limit is 9 in all cases.

ALPHA HEIGHT without any parameters restores the default height (one of the upper limits mentioned above). The minimum argument to this statement is always 9; however, when you are in Edit mode the minimum alpha height is 14. Note that upon entering the Edit mode if the ALPHA HEIGHT is a value in the range of 9 to 13 it will be changed to 14.

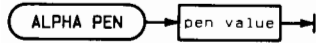
Note that the functionality of this statement can be achieved through CRT CONTROL register 13; however, you **cannot** execute the CONTROL statement without a parameter in order to get the default alpha height.



# ALPHA PEN

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

This statement selects the alpha display color for the CRT.



Item	Description/Default	Range Restrictions
pen value	numeric expression	(see Semantics)

## Example Statements

ALPHA PEN Pen\_value  
ALPHA PEN 142  
IF Cyan THEN ALPHA PEN 140



## Semantics

This statement has no effect on monochrome displays.

The set of alpha colors for the Model 236C is given in the table below:

Value	Result
< 16	The number is evaluated MOD 8 and resulting values produce the following: 0 — black 1 — white 2 — red 3 — yellow 4 — green 5 — cyan 6 — blue 7 — magenta
16 to 135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144 to 255	Ignored

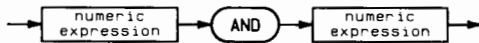
For bit-mapped alpha displays, ALPHA PEN specifies the graphics pen to be used for subsequent alpha output. The range of values allowed with this statement are 0 through 255; these values are treated as MOD  $2^n$ , where  $n$  is the number of display planes.

ALPHA PEN  $n$  or CONTROL CRT,5;  $n$  set the values of the CRT registers 15, 16, and 17 (or PRINT PEN, KEY LABELS PEN and KBD LINE PEN, respectively), but the converse is not true. That is, STATUS CRT,5 may not accurately reflect the CRT state if control registers 15, 16, and/or 17 have been set.

Note that the functionality of this statement can be achieved through CRT CONTROL register 5.

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.

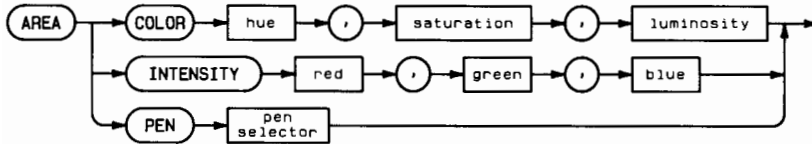
The logical AND is shown in this table:

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	Range
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, GINIT, or LOAD BIN "GRAPH" (when this binary is not already resident in memory).

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**

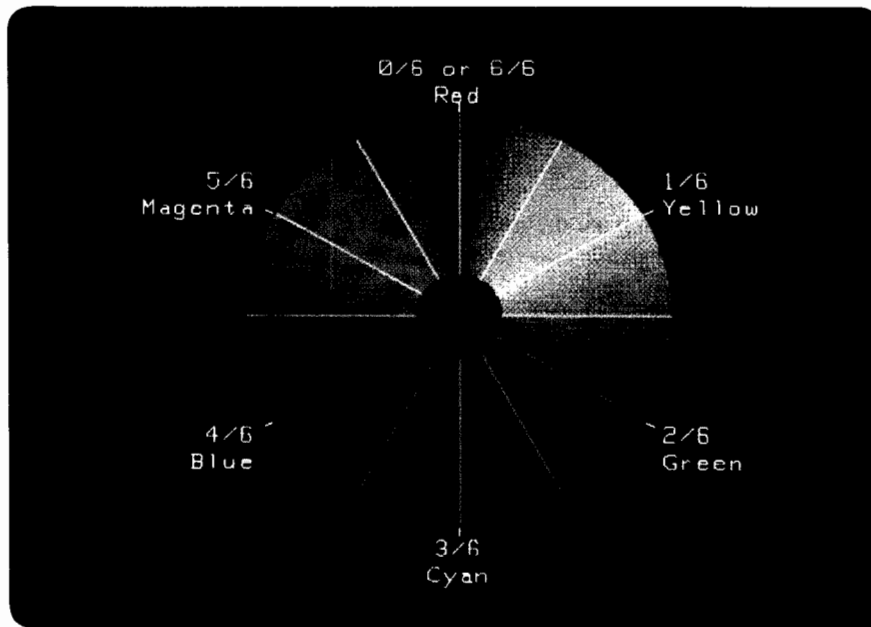
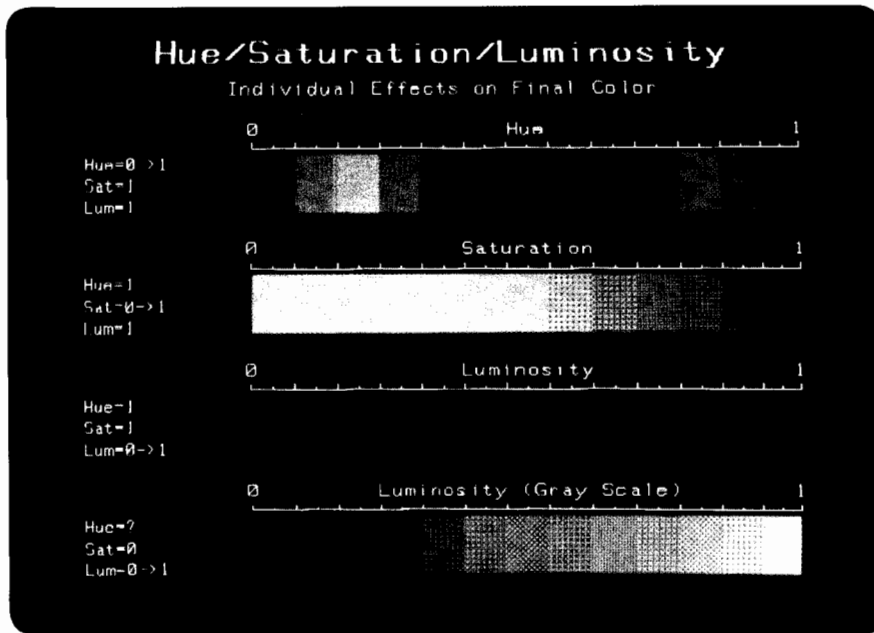
When AREA COLOR is executed on a machine with a color display, the HSL parameters are converted to RGB values. Then, if the color requested is not available in the color map, the computer creates the closest possible color in RGB color space to the one requested by filling the 4×4 dither cell with the 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  $2^n$  total colors (where  $n$  is the number of planes in the graphics display), and they can be redefined with SET PEN.

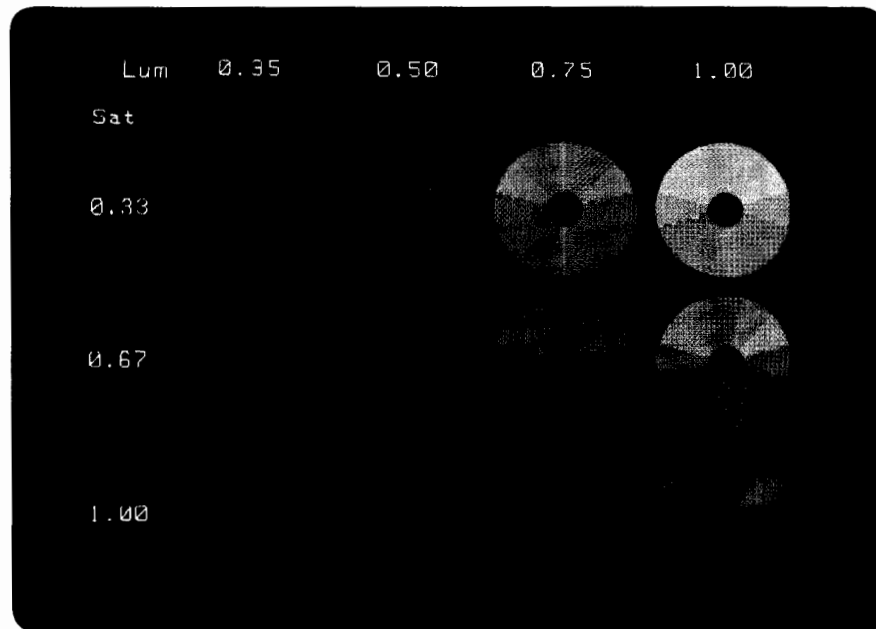
The first photo of the screen on the next page shows the changes brought about by varying one of HSL parameters at a time. The bottom bar shows that when saturation (the amount of color) is zero, hue makes no difference, and varying luminosity results in a gray scale.

The second photo on the next page is a color wheel which shows 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.



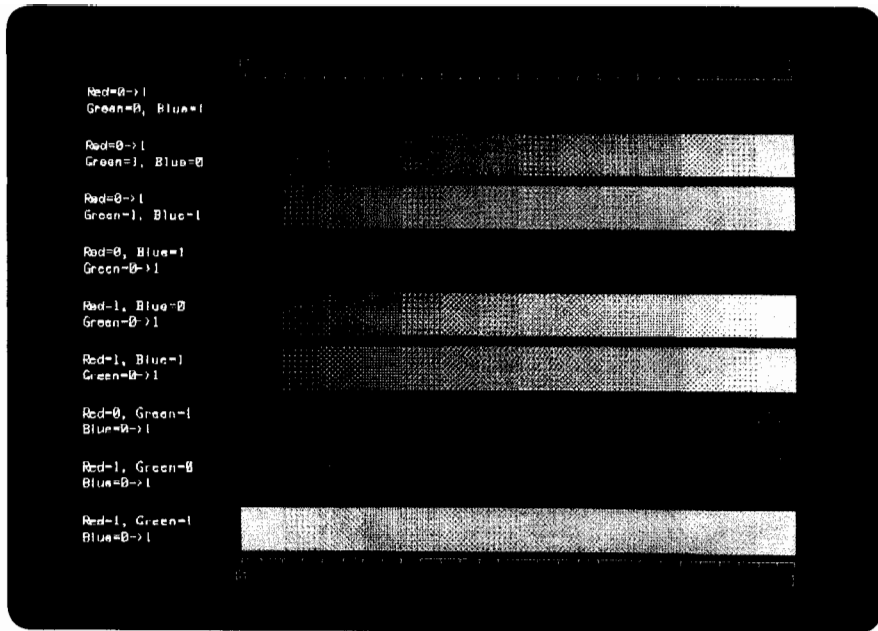
**Effects of Saturation and Luminosity on Color**

**AREA INTENSITY**

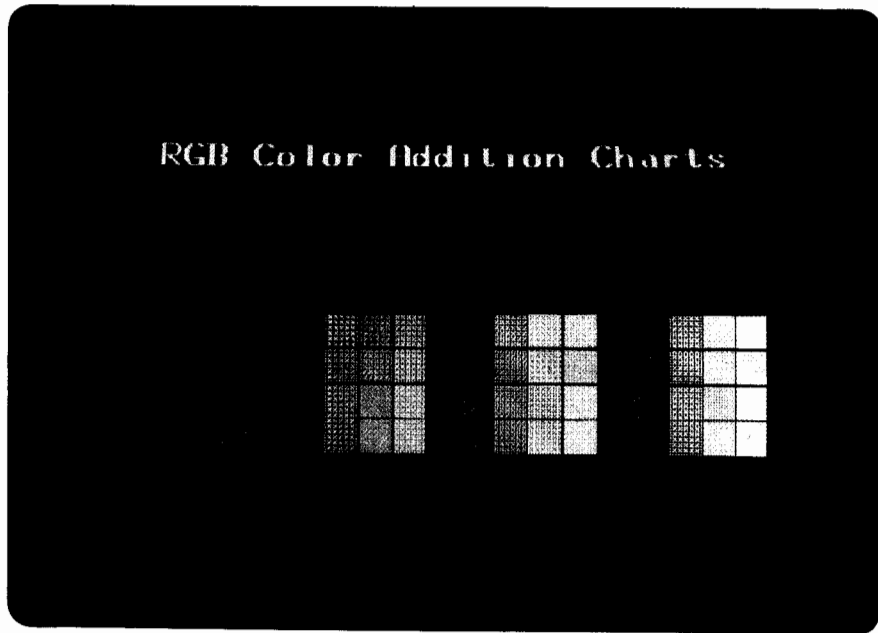
The first photo on the next page demonstrates the effect of varying the intensity of one color component when the other two remain constant.

The second photo (on next page) 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.





RGB Addition: One Color at a Time



**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 Model 236 with color monitor when the color map is not enabled (see PLOTTER IS), using only eight colors when calculating the closest combination.

**Monochromatic Displays**

When doing shading on a monochromatic display, 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**

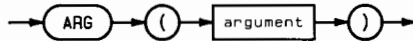
If the alternate drawing mode is in effect when the fill is performed, the area will be filled with non-dominant color. See GESCAPE operation selectors 4 and 5.

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

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



This function returns the argument (or the angle in polar coordinates) of a COMPLEX number.



Item	Description/Default	Range Restrictions
argument	numeric expression	any valid INTEGER, REAL, or COMPLEX value

## Example Statements

```
X=ARG(Complex_expr)
Y=ARG(Real_expr)
Z=ARG(Integer_expr)
Result=ARG(CMPLX(2.1,-8))
```

## Semantics

This is equivalent to `ATN2(CMPLX(Imag_part,Real_part))` in FORTRAN. The value returned is REAL. If the current angle mode is DEG, the range of the result is  $-180^\circ$  thru  $+180^\circ$ . If the current angle mode is RAD, the range of the result is  $-\pi$  thru  $+\pi$  radians. The default mode is radians.

This function returns 0 when given an INTEGER or REAL argument.

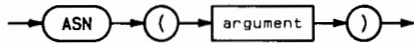
## ASCII

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 principle 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 for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

## Examples Statements

```
Angle=ASN(Sine)
PRINT "Angle = ";ASN(CMPLX(-2.98,3))
```

## Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

The angle mode (RAD or DEG) for REAL and INTEGER arguments indicates whether you should interpret the value returned in degrees or radians. If the current angle mode is DEG, the range of the result is  $-90^\circ$  to  $90^\circ$ . If the current angle mode is RAD, the range of the result is  $-\pi/2$  to  $+\pi/2$  radians. The angle mode is radians unless you specify degrees with the DEG statement.

To compute the ASN of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula for computing the ASN of a COMPLEX value is:

```
-i*LOG(i*Argument+SQRT(1-Argument*Argument))
```

where *i* is the COMPLEX value `CMPLX(0,1)` and *Argument* is a COMPLEX argument to the ASN function. Some values of a COMPLEX argument may cause errors in this computation. For example:

```
ASN(CMPLX(MAXREAL,0))
```

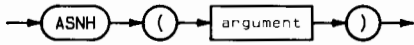
will cause error 22 due to the *Argument\*Argument* computation.

The principle value, which has a real part between  $-\pi/2$  and  $+\pi/2$ , is returned for COMPLEX arguments.

# ASNH

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

This function returns the hyperbolic arcsine of a numeric expression.



Item	Description/Default	Range Restrictions
argument	numeric expression	absolute value < 1.340 780 792 99 E+154 and > 1.491 668 146 24 E -154 for INTEGER and REAL arguments; see "Range Restriction Specifics" for additional restrictions.

## Example Statements

```
Result=ASNH(-.2475)  
PRINT "Hyperbolic Arcsine = ";ASNH(Expression)
```

## Semantics

If an INTEGER or REAL argument is given, this function returns a REAL value. If a COMPLEX argument is given, this function returns a COMPLEX value.

### Range Restriction Specifics

The formula use for computing the ASNH is as follows:

```
LOG(Argument+SQRT(Argument*Argument+1))
```

where **Argument** is the argument to the ASNH function. Some values of a COMPLEX argument may cause errors in this computation. For example:

```
ASNH(CMPLX(MAXREAL,0))
```

will cause error 22 (REAL overflow) due to the **Argument\*Argument** computation.

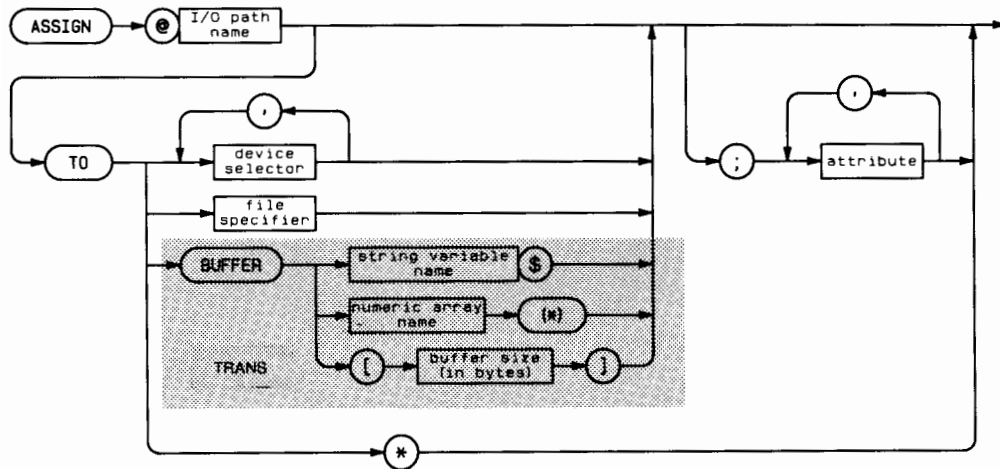
Note that the ASNH of a COMPLEX number returns a principle value which has an imaginary part that falls in the range of  $-\pi/2$  to  $+\pi/2$ .

# ASSIGN

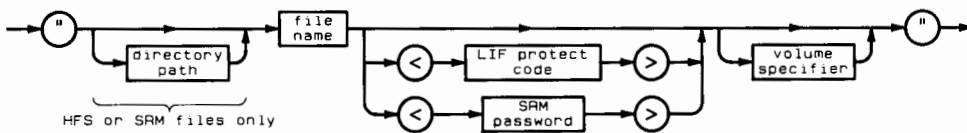
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used to perform one of the following actions:

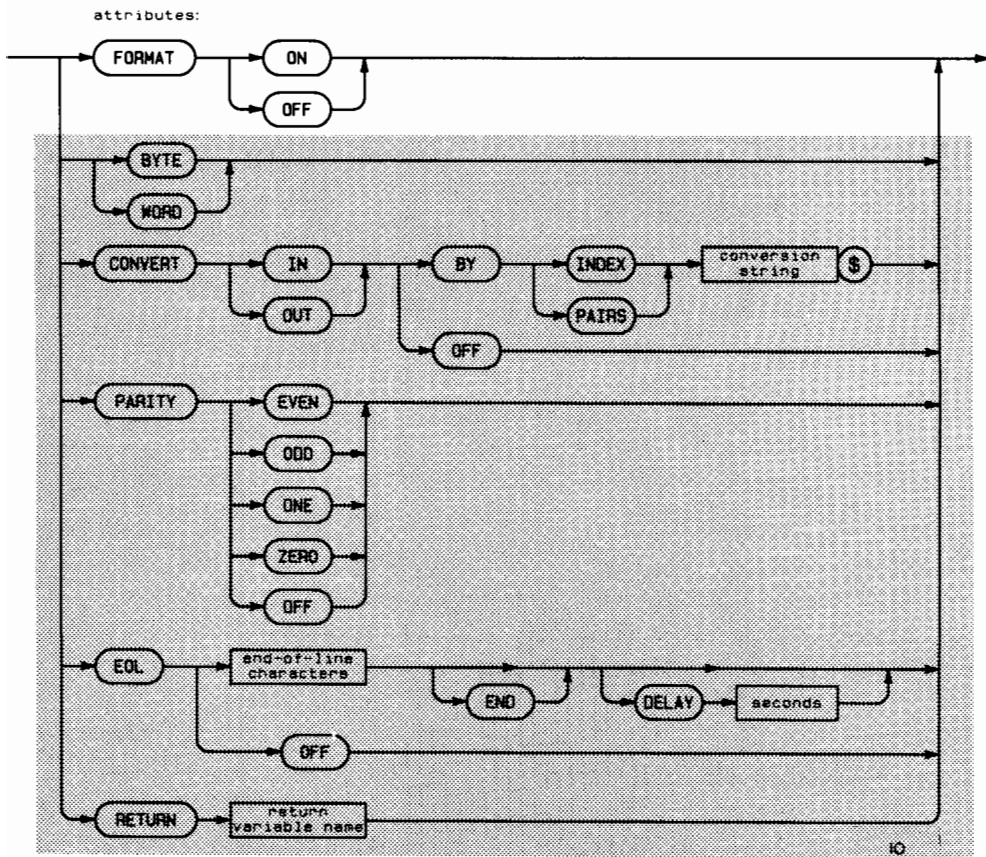
- assign an I/O path name and attributes to:
  - a device,
  - a group of devices,
  - a mass storage file,
  - or a buffer;
- change attributes;
- or close an I/O path name.



literal form of file specifier:







Item	Description	Range
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 (see Glossary)
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 690
attribute	attribute to be assigned to the I/O path name	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
LIF protect code	literal; first two non-blank characters are significant	> not allowed
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	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 @Source TO Isc;FORMAT OFF
ASSIGN @Source;FORMAT ON
ASSIGN @Device TO 724
ASSIGN @Listeners TO 711,712,715
ASSIGN @Dest TO *

ASSIGN @File TO File_name$
ASSIGN @File TO Dir_path$&File_name$&Vol_spec$
ASSIGN @File TO "WorkDir/File"
ASSIGN @File TO "/RootDir/MyDir/MyFile: ,700"
ASSIGN @File TO "DIR_JOHN/dir_proj/file1"
ASSIGN @Srm_file TO "P1/FredsData<pass>:REMOTE"

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 148 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:** 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 this 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 Attribute
interface/device	FORMAT ON
ASCII file	(always ASCII format)
BDAT file	FORMAT OFF
HP-UX 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 display. 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 connected to 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 (that is, it “opens” the file”). The mass storage file must be a data file (a file of type ASCII, BDAT, or HP-UX). The file must already exist on the media, as ASSIGN does not do an implied CREATE.

Data 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. The position pointer is reset to the beginning of the file when the file is opened, and updated with each ENTER or OUTPUT that uses that I/O path name. (It is best if a file is open with only one I/O path name at a time.)

BDAT and HP-UX files have an additional *physical* end-of-file pointer. This end-of-file pointer (which resides on the media) is read when the file is opened. This 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.

### **HFS Permissions**

ASSIGN opens any existing ASCII, BDAT, or HP-UX file if you currently have R (read) access permission on the file as well as X (search) permission on the parent and all superior directories. Otherwise, error 183 will be reported.

### **SRM Access Capabilities**

ASSIGN opens any existing ASCII, BDAT, or HP-UX 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) will result in error 62.

The file's specific access capabilities are not checked at ASSIGN time. A subsequent operation on the file associated with the I/O path name is not performed, however, unless you have the proper 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 generates error 62, since the WRITE access capability is not public (this operation would be successful if you specify the WRITE password in the ASSIGN statement).

### **Locked SRM Files**

With SRM volumes, existing ASCII, BDAT, and HP-UX 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 @Io\_path TO \*) unlocks as well as closes the file (regardless of the number of LOCKs in effect for the file at the time).

### **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, COMPLEX, 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" section at the back of this manual for I/O path register definitions; the *BASIC Interfacing Techniques* manual provides tutorial information about these interface registers.

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.

#### **Additional 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 by BASIC 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 (such as the EOI signal on HP-IB interfaces); 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:

<b>Option</b>	<b>Effect on Incoming Data</b>	<b>Effect on Outbound Data</b>
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, ERROR 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** (**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 which has a tangent equal to the argument. This is the arctangent function.



Item	Description/Default	Range Restrictions
argument	numeric expression	within valid ranges of INTEGER or REAL data types for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

## Examples Statements

```
Angle=ATN(Tangent)
PRINT "Angle = ";ATN(CMPLX(-1.5,3.5))
```

## Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

The angle mode (RAD or DEG) for REAL and INTEGER arguments indicates whether you should interpret the value returned in degrees or radians. If the current angle mode is DEG, the range of the result is  $-90^\circ$  to  $90^\circ$ . If the current angle mode is RAD, the range of the result is  $-\pi/2$  to  $+\pi/2$  radians. The angle mode is radians unless you specify degrees with the DEG statement.

To compute the ATN of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula for computing the ATN of a COMPLEX value is as follows:

$$(i/2)*\text{LOG}((i+\text{Argument})/(i-\text{Argument}))$$

where  $i$  is the COMPLEX value `CMLX(0,1)` and `Argument` is a COMPLEX argument to the ATN function. Some values of a COMPLEX argument may cause errors in this computation. For example,

$$\text{ATN}(\text{CMLX}(\text{MAXREAL}, \text{MAXREAL}))$$

will cause error 22 due to the computation of

$$(i+\text{Argument})/(i-\text{Argument})$$

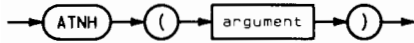
ATN is not defined at  $i$  and  $-i$  and will generate error 623 given those arguments.

The principle value, which has a real part between  $-\pi/2$  and  $+\pi/2$ , is returned for COMPLEX arguments.

# ATNH

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

This function returns the hyperbolic arctangent of a numeric expression.



Item	Description/Default	Range Restrictions
argument	numeric expression	-1 through +1 for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments.

## Example Statements

```
Result=ATNH(-.4571)
PRINT "Hyperbolic Arctangent = ";ATNH(X1)
```

## Semantics

If an INTEGER or REAL argument is given, this function returns a REAL value. If a COMPLEX argument is given, this function returns a COMPLEX value.

## Range Restriction Specifics

The formula for computing ATNH is as follows:

$$\text{LOG}((1+\text{Argument})/(1-\text{Argument}))/2$$

where **Argument** is the argument to the ATNH function. Some values of the argument may cause errors in this computation. For example:

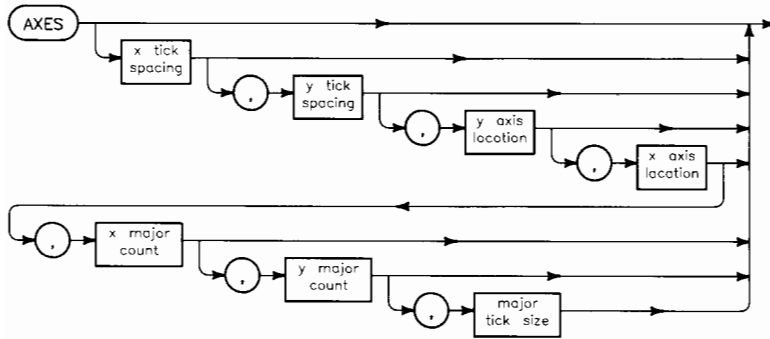
$$\text{ATNH}(\text{CMPLX}(\text{MAXREAL}, \text{MINREAL}))$$

will cause error 22 (REAL overflow) due to the divide operation in the formula.

Note that the hyperbolic arctangent of a COMPLEX number returns a principle value, that has an imaginary part which falls in the range of  $-\pi/2$  to  $+\pi/2$ .

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	Range
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 tickmarks; 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			<i>Note 4</i>
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES and GRID)	X				
Location of labels	<i>Note 1</i>	<i>Note 3</i>		<i>Note 2</i>	

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.

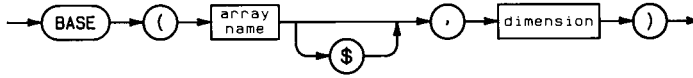


## Notes

## 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	Range
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
```

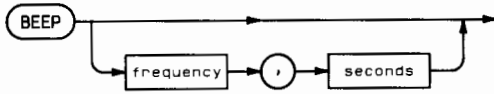
## BDAT

See the CREATE BDAT statements.

# BEEP

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

This statement produces one of 64 audible tones.



Item	Description	Range
frequency	numeric expression, rounded to the nearest tone; Default = 1220.7	81 thru 5208 (see table)
seconds	numeric expression, rounded to the nearest hundredth; Default = 0.2	

## 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 in the table on the following page. 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.

Rounding is performed by the system to produce the number in the first column of the following table. With an HP-HIL interface, the frequency produced is the corresponding number in the second column. (Note that the frequencies generated by a computer with a 98203A/B style keyboard are slightly different than those generated by an HP-HIL keyboard/interface combination.)

### Rounding of BEEP Frequency Parameters

Series 200	HP-HIL
81.38	81.45
162.76	162.12
244.14	244.37
325.52	324.25
406.90	408.49
488.28	496.03
569.66	578.70
651.04	651.03
732.42	744.04
813.80	833.33
895.18	905.79
976.56	992.06
1057.94	1096.49
1139.32	1157.40
1220.70	1225.49
1302.08	1302.08
1383.46	1388.88
1464.84	1461.98
1546.22	1543.20
1627.60	1633.98
1708.98	1700.67
1790.36	1773.04
1871.74	1851.84
1953.12	1937.98
2034.50	2032.51
2115.88	2136.74
2197.26	2192.97
2278.64	2252.24
2360.02	2380.94
2441.40	2450.97
2522.78	2525.24
2604.16	2604.16

Series 200	HP-HIL
2685.54	2688.16
2766.92	2777.77
2848.30	2873.55
2929.68	2976.18
3011.06	2976.18
3092.44	3086.41
3173.82	3205.12
3255.20	3205.12
3336.58	3333.32
3417.96	3472.21
3499.34	3472.21
3580.72	3623.17
3662.10	3623.17
3743.48	3787.86
3824.86	3787.86
3906.24	3968.24
3987.62	3968.24
4069.00	4166.65
4150.38	4166.65
4231.76	4166.65
4313.14	4385.95
4394.52	4385.95
4475.90	4385.95
4557.28	4629.61
4638.66	4629.61
4720.04	4629.61
4801.42	4901.94
4882.80	4901.94
4964.18	4901.94
5045.56	4901.94
5126.94	5208.31
5208.32	5208.31

The resolution of the seconds parameter is 0.01 seconds. Any duration shorter than 0.005 seconds is treated as 0. Any duration longer than 2.55 seconds is treated as 2.55 seconds.

In a few cases with an HIL interface, the frequency produced will not be the closest table entry. For example, `BEEP 203.4, .5` will BEEP at 162.12 on an HIL interface even though the specified frequency is closer to 244.37 (in the HP-HIL column) because the closest Series 200 entry is 162.76.

Series 200	HP-HIL
162.76	162.12
244.14	244.37

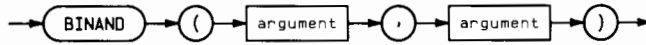
## **BIN**

---

See the `LOAD`, `LIST`, and `SCRATCH` statements.

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	Range
argument	numeric expression, rounded to an integer	-32 768 thru +32 767

## Example Statements

```
Low_4_bits=BINAND(Byte,15)
IF BINAND(Stat,8) THEN Bit_3_set
```

## Semantics

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

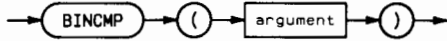
For example, the statement `Ctrl_word=BINAND(Ctrl_word,-9)` clears bit 3 of `Ctrl_word` without changing any other bits.

	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	Range
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 equals +8:

	bit 15		bit 0
	↓		↓
-9 =	11111111	11110111	
+8 =	00000000	00001000	



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	Range
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 OR'ed with the corresponding bit in the other argument. The results of all the exclusive OR's are used to construct the integer which is returned.

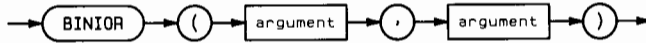
For example, the statement `Ctrl_word=BINEOR(Ctrl_word,4)` inverts bit 2 of `Ctrl_word` without changing any other bits.

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	Range
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 OR'ed with the corresponding bit in the other argument. The results of all the inclusive OR's are used to construct the integer which is returned.

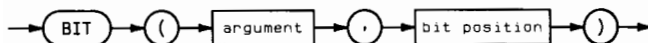
For example, the statement `Ctrl_word=BINIOR(Ctrl_word,6)` sets bits 1 & 2 of `Ctrl_word` without changing any other bits.

	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	

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	Range
argument	numeric expression, rounded to an integer	-32 768 thru +32 767
bit position	numeric expression, rounded to an integer	-32 768 thru +32 767

## 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.

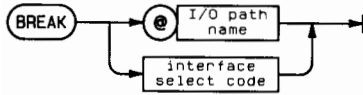
```

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	Range
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 an HP 98626 or 98644 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, COMPLEX, COM, ASSIGN, SUB, and DEF FN statements.

## **BYTE**

---

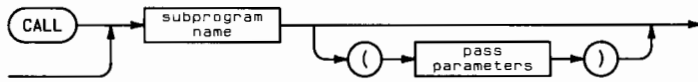
See the ASSIGN statement.

# Notes

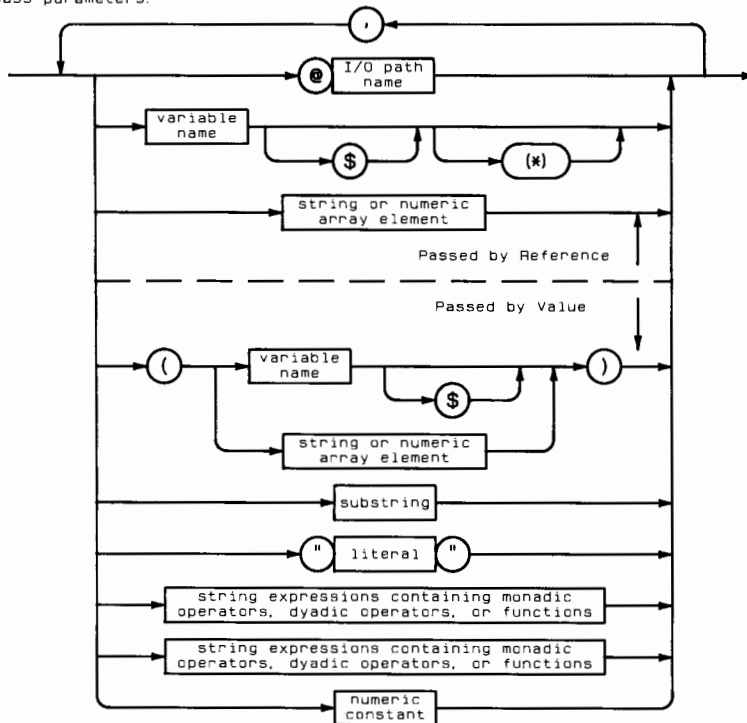
# CALL

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 parameters to the subprogram. SUB programs are created with the SUB statement. (Also see the ON... statements.)



pass parameters:



Item	Description	Range
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 <code>ANY CHAR</code> key	—
numeric constant	numeric quantity expressed using numerals, and optionally a sign, a decimal point, and/or exponent notation	---

### Example Statements

```
CALL Process(Ref, (Value), @Path)
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`, `INTEGER`, or `COMPLEX`) 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 (such as ON END, ON ERROR, ON KEY, etc.), program execution resumes at the point at which the event-initiated branch was permitted.

When CALL is executed from the keyboard, the subprogram is executed in its own separate context. Furthermore, the current state of the system determines the system'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

---

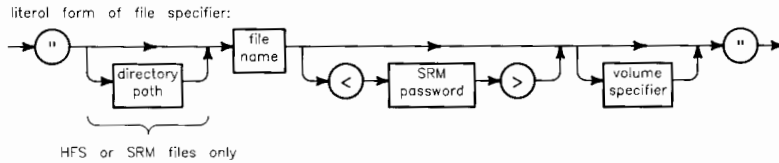
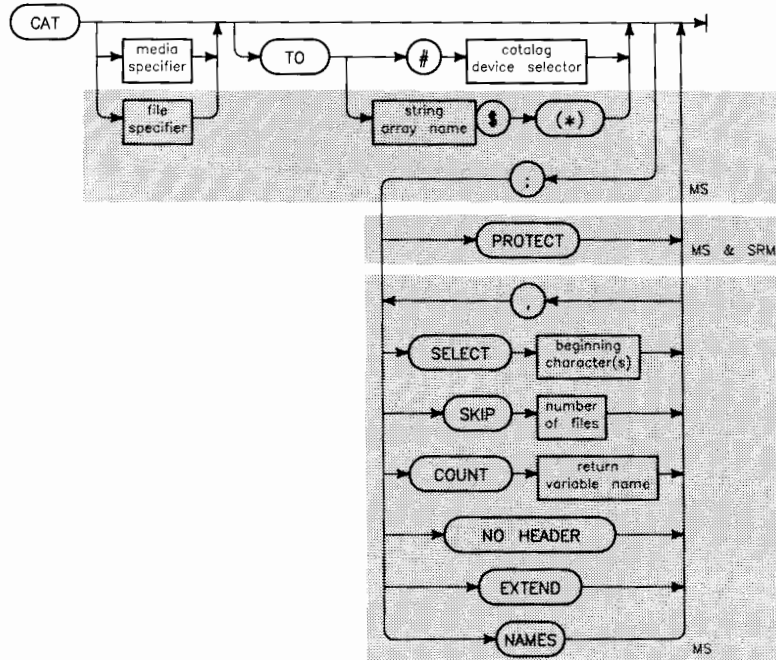
See the SELECT...CASE construct.



# 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 lists information regarding a specified PROG-type file.



Item	Description	Range
media specifier	string expression; Default=MASS STORAGE IS directory	(see MASS STORAGE IS)
file specifier	string expression specifying a PROG-type file	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
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 (LIF); 1 to 14 characters (HFS); 1 to 16 characters (SRM)
number of files	numeric expression, rounded to an integer	1 thru 32 767
return variable name	name of a numeric variable	any valid name

### Example Statements

```

CAT
CAT TO #701
CAT ":,700,1"
CAT ":REMOTE"
CAT ":REMOTE; LABEL Mastervol"

CAT "A/B/C:REMOTE"
CAT ".../..."
CAT "Dir1/Dir2"

CAT "HFS_Dir";NAMES

CAT "PROG_File"

CAT;SELECT "D",SKIP Ten_files,NO HEADER

CAT TO String_array$(*)
CAT TO Hfs_dir$(*);EXTEND

CAT "My_File";PROTECT

```

## Semantics

A directory entry is listed for each file in the specified directory. The catalog shows information such as 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 file types recognized in BASIC are: ASCII, BDAT (BASIC data), BIN (binary program), HP-UX, PROG (BASIC program), and SYSTM (operating system). An ID number is listed for any unrecognized file types.

## LIF Catalogs

The LIF catalog format is shown below.

```
:CS80,700
VOLUME LABEL: B9836
FILE NAME PRO TYPE REC/FILE BYTE/REC ADDRESS DATE TIME
MyProg      PROG      14    256    16 23-May-87 7:58
VisiComp    ASCII     29    256    30  8-Apr-87 6:00
GRAPH       BIN      171   256    59  1-May-87 1:00
GRAPHX      BIN      108   256   230 10-Aug-87 9:00
```

The first line of the catalog shows the volume specifier (:CS80,700 in this example).

The second line shows the volume label—a name, containing up to 6 characters, stored on the media (B9836 in this example).

The third line labels the columns of the remainder of the catalog. Here is what each column means:

<b>FILE NAME</b>	lists the names of the files in the directory (up to 10 characters).
<b>PRO</b>	indicates whether the file has a protect code (* is listed in this column if the file has a protect code).
<b>FILE TYPE</b>	lists the type of each file.
<b>REC/FILE</b>	indicates the number of records in the file.
<b>BYTE/REC</b>	indicates the record size.
<b>ADDRESS</b>	indicates the number of the beginning sector in the file.
<b>DATE</b>	indicates when the date the file was last modified.
<b>TIME</b>	indicates the time the file was last modified.

## HFS Catalogs

In order to perform a CAT of an HFS directory, you need to have R (read) and X (search) permissions on the directory to be cataloged, as well as X (search) permissions on all superior directories.

In order to perform a CAT of an HFS file, you need to have R (read) permission on the file to be cataloged, as well as X (search) permissions on all superior directories.

Here is a typical catalog listing of an HFS directory. Note that a 50 column display truncates this catalog listing after the column with TIME in it. Therefore, the PERMISSION, OWNER, and GROUP columns will not be listed.

```
:CS80, 700
LABEL: MyVol
FORMAT: HFS
AVAILABLE SPACE:      60168
FILE      NUM  REC   MODIFIED
FILE NAME  TYPE RECS  LEN DATE      TIME PERMISSION OWNER GROUP
=====
lost+found DIR    0   32 19-Nov-86 10:47  RWXRWRWX   18   9
FILEIOD    PROG  191  256 21-Nov-86  9:03  RW-RW-RW-  18   9
RBDAT      BDAT   2   256 21-Nov-86  9:10  RW-RW-RW-  18   9
CATTOSTR   PROG   2   256  1-Dec-86  8:02  RW-RW-RW-  18   9
```

The first line of the catalog shows the volume specifier (:CS80,700 in this example).

The second line shows the volume label—a name, containing up to 6 characters, stored on the media (MyVol in this example).

The third line shows the format of the disc (HFS in this example).

The fourth line lists the number of available 256-byte sectors on the disc (60168 in this example). If the sector size is 1024 bytes, then each 1024-byte sector would count as 4 256-byte sectors.

The fifth line labels the columns of the remainder of the catalog. Here is what each column means:

FILE NAME	Lists the name of the file.
FILE TYPE	Lists the file's type (for instance, DIR specifies that the file is a directory; PROG specifies a BASIC program file; BDAT specifies a BASIC DATA file; etc.).

NUM RECS	number of <i>logical</i> records (the number of records allocated to the file when it was created). For a DIR file, this indicates the number of directory entries.
REC LEN	the <i>logical</i> record size (default is 256 bytes; BDAT files can have user-selected record lengths). For a DIR file, this indicates the size of the directory entry. You <b>cannot</b> specify record length for ASCII or HP-UX files. The record length for HP-UX files is 1.
MODIFIED DATE      TIME	the day and time when the file was last modified.
PERMISSION	specifies who has access rights to the file: <ul style="list-style-type: none"> <li>R indicates that the file can be read;</li> <li>W indicates that the file can be written;</li> <li>X indicates that the file can be searched (meaningful for directories only).</li> </ul> <p>There are 3 classes of user permissions for each file:</p> <ul style="list-style-type: none"> <li>OWNER (left-most 3 characters);</li> <li>GROUP (center 3 characters);</li> <li>OTHER (right-most 3 characters).</li> </ul> <p>See PERMIT for further information.</p>
OWNER	specifies the owner identifier for the file (for BASIC files, the default owner identifier is always 18).
GROUP	specifies the group identifier of the file or directory (for BASIC, the default group identifier is always 9, which is used for “workstations” such as Series 200/300 BASIC and Pascal).

### CAT of an SRM Directory

In order to perform a CAT of an SRM directory or file, you need to have R (read) access capability on the directory to be cataloged, as well as R capability on all superior directories.

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:

```
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0
LABEL:   Disc1
FORMAT:  SDF
AVAILABLE SPACE:      54096
FILE NAME      PUB FILE  NUMBER RECORD  OPEN
                ACC  TYPE  RECORDS LENGTH  STAT
=====  ===  =====  =====  =====
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 an SRM directory on an 80-column display, the system uses the following catalog format:

```
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0
LABEL:   Disc1
FORMAT:  SDF
AVAILABLE SPACE:  54096 SYS  FILE  NUMBER  RECORD  MODIFIED  PUB  OPEN
FILE NAME        LEV TYPE  TYPE  RECORDS  LENGTH  DATE      TIME  ACC  STAT
=====
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 15:06 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 specifier, including volume specifier. 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 left-most 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 being cataloged.

*line 3* Directory format, such as SDF (Structured Directory Format).

*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.





Each **column** of the remaining catalog gives you the following information:

**FILE NAME** lists the names of the files and directories in the directory being cataloged.

**LEV** (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.)

**PUB ACC** 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.

**SYS TYPE** (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. 98X6 denotes a Series 200/300 computer. (If the system does not recognize the system type, a coded identifier, obtained from the system being identified, appears in this column.)

**FILE TYPE** 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. For example, in the display illustrated earlier, DATA and PROG files created on an HP 9845 are listed as such, but shown with the question mark.

File types recognized by the BASIC system on SRM are: ASCII, BDAT, BIN, DIR, HP-UX, 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.

NUMBER RECORDS	indicates the number of records in the file.
RECORD LENGTH	indicates the number of bytes constituting each of the file's records.
MODIFIED	(80-column format only) show the date and time the file's contents were last changed.
OPEN STAT	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 System Administrator's Guide* (or *SRM Operating System Manual*) for details.

---

#### **CAT to a Device**

When the symbol # is included in a CAT statement, the numeric expression following this symbol must be a device selector. The catalog listing is sent to the device specified by this expression.

#### **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.

```

:CS80,702,0
VOLUME LABEL: B9836
FORMAT: LIF
AVAILABLE SPACE: 11

```

```

      SYS FILE  NUMBER  RECORD  MODIFIED  PUB OPEN
FILE NAME  TYPE  TYPE  RECORDS  LENGTH DATE      TIME ACC STAT
=====
SYSTEM_BA5  1 98X6 SYSTM   1024   256 29 Nov 86 15:24:55 MRW
AUTOST      1 98X6  PROG    38    256 29 Nov 86 09:25:07 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	37 thru 45
Record Length	46 thru 54
Time Stamp	56 thru 71
Public Access Capabilities	73 thru 75
Open Status	77 thru 80

#### The EXTEND Option (Requires MS)

If EXTEND is specified in a CAT TO String\_array\$(\*) operation, the directory catalog will be in HFS format for an HFS disc or in LIF format for a LIF disc. Otherwise, this option has no effect on the CAT operation.

With an HFS disc, each element of the array must contain at least 49 characters. If each element has less than 72 characters, the PERMISSION, OWNER, and GROUP are omitted. With a LIF disc, each element of the array must contain at least 47 characters. If each element has less than 65 characters, the DATE and TIME will be omitted.

NAMES takes precedence over EXTEND if both are given.

### 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.

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

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.

### 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.

```
CAT; SELECT "B21"
```

If the string expression contains more characters than are allowed in a file name (10 for LIF, 14 for HFS, and 16 for SRM), then only the first 10 characters if LIF, 14 characters if HFS, or 16 characters if SRM 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.

```
CAT; SKIP 12
```

If SKIP is not included, no files are skipped.

If an option is given more than once, only the last instance is used.

### How Many Entries? (Requires MS)

Including COUNT provides a means of determining the number of lines sent to the destination.

```
CAT; COUNT N_files
```

The variable that follows COUNT receives the sum of the number of selected files *plus* the number of lines in the catalog header (and trailer for PROG 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 in the LIF format have a five-line header; in SRM and HFS formats they have seven-line headers. Catalogs to string arrays are SRM format unless EXTEND is added. 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 no entries are sent to the destination (because the directory is empty, or because no entries were selected, or because all selected entries were skipped), the value returned depends on whether there is a header. If there is no header, then zero (0) is returned. If there is a header, then the value returned is the size of the header plus the number following the skip option (the number requested to be skipped).

If an option is given more than once, only the last instance is used.

### Using the NAMES Option (Requires MS)

Using the NAMES option, as shown in the following statement, will produce a multi-column listing of *only* the names of the files.

```
CAT ; NAMES 
lost+found      WORKSTATIONS      SYSTEM_BA5      MY_PROG      DATA_13
PROJECTS
```

Executing the following statement:

```
CAT TO A$(*);NAMES
```

will put one file name in each array element.

### 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.

```
CAT ; NO HEADER
```

When NO HEADER is specified, the lines of the header (and trailer) are then omitted from the COUNT variable.

When NAMES is specified, there is NO HEADER whether or not NO HEADER is specified.

### The PROTECT Option (SRM Only)

PROTECT is a CAT option available *only* on SRM volumes. This option requires the SRM, DCOMM, and MS binaries. 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.

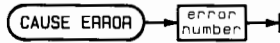
PROTECT must be specified separately from other CAT options, and is allowed only with SRM files and directories. Using PROTECT with media other than SRM results in **ERROR 1 Configuration error.**

# CAUSE ERROR

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



This statement simulates the occurrence of an error of the specified number.



Item	Description	Range
error number	numeric expression, rounded to an integer	1 thru 999; 1001 thru 1080

## Example Statements

```
CAUSE ERROR Err_num  
IF Testing THEN CAUSE ERROR 80
```

## Semantics

When this statement is executed, it initiates the normal error-reporting action taken by the system when an error is encountered in a program line.

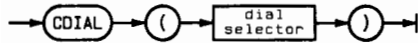
If ON ERROR is in effect and CAUSE ERROR is executed in a program line, the appropriate branch is initiated—just as if an actual error occurred on that line. When executed from a running program, CAUSE ERROR affects the error indications ERRN, ERRM\$, ERRL, and ERRLN; each is set to the value appropriate for the specified error number and line number. However, ERRDS is not affected.

If CAUSE ERROR is executed at the keyboard, or if executed in a running program (while ON ERROR is not in effect), BASIC shows the error number (and error message, if the ERR binary is present) in the system message line of the display. (Note that errors caused by executing statements at the keyboard do not affect the error indications listed in the preceding paragraph.)

# CDIAL

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

This function returns information about “control dial” devices.



Item	Description	Range
dial selector	numeric expression, rounded to an integer	0 thru 15

## Example Statements

```
CDIAL(1)
IF BIT(CDIAL(0),3) THEN GOSUB Dial3_touched
Pulses_dial_9=CDIAL(9)
```

## Semantics

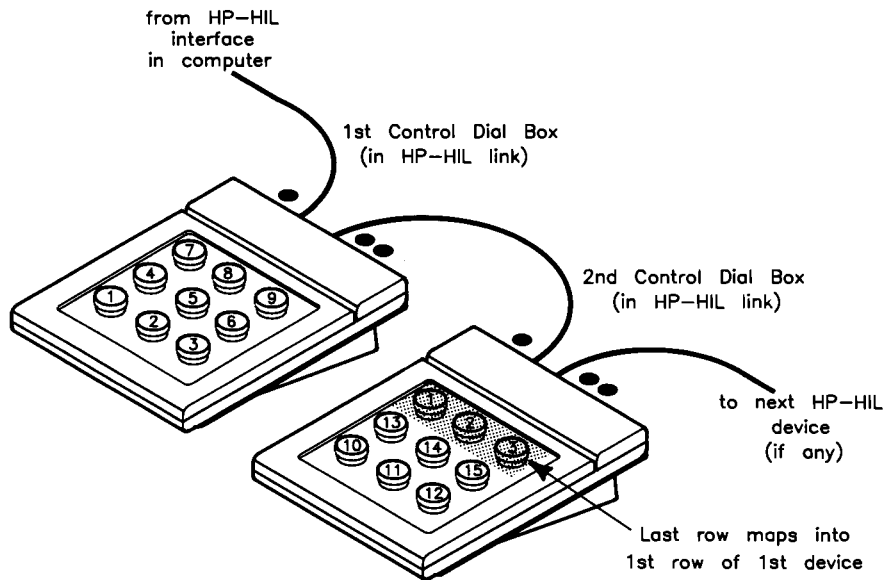
Two different types of results can be returned by this function:

- If a value of **0** is passed to the CDIAL function, it returns a 16-bit status word specifying which knobs have been rotated. Bit 1 set indicates that dial 1 has been rotated; bit 2 set indicates that dial 2 has been rotated; and so forth through bit 15. (Bit 0 is not used.)
- If a value of **1 through 15** is passed, the function returns the number of pulses accumulated for that particular dial (and clears the corresponding pulse counter).

## Mapping of Dials

Here is the mapping of control dial numbers to physical dial locations (this example shows two HP 46085 Control Dial Boxes in use on one HP-HIL interface).





Once CDIAL is used to read information about a particular dial (registers 1 thru 15), that register is automatically cleared as is the corresponding bit in register 0. Reading register 0 does not change register 0.

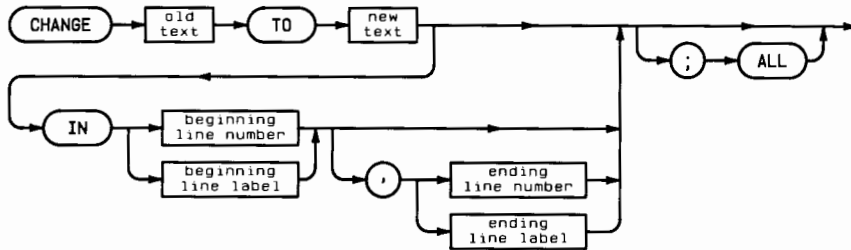
All registers are automatically cleared when ON CDIAL is executed.

If ON CDIAL is used to set up control dial interrupts and then disabled (with OFF CDIAL), the CDIAL function can still be used to determine valid information about control dials: however, note that subsequent pulses will not be accumulated into the CDIAL registers, and when a register is read with CDIAL, that register is automatically cleared by the system.

# CHANGE

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

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



Item	Description	Range
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"
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 text to new text, and asks you to confirm the change.

- You confirm a particular change by pressing **Return** or **ENTER**.
- You may bypass a particular change by pressing **CONTINUE** (**F2** on an ITF keyboard), or **Shift-Clear line** followed by **Return** (**CLR LN** followed by **ENTER** on a 98203 keyboard).

**↑** and **↓** exit CHANGE mode. **EXECUTE** confirms a change, and exits CHANGE mode.

When the specified range is exhausted or the end of the program is reached, the CHANGE command is terminated and the message "new text" not found is displayed.

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 erroneous 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.

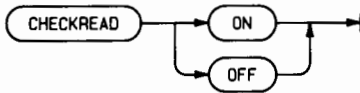
CHANGE is not allowed while a program is running; however, it may be executed while a program is paused. The program is continuable if it has not been altered by pressing **Return** or **Delete line** (**ENTER** or **DEL LN**).

While in the CHANGE mode, keyboard execution of commands is only possible with the **EXECUTE** key on a 98203 keyboard. Using **ENTER** 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.



## 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-STORE
CREATE	PROTECT	SAVE
CREATE ASCII	PURGE	STORE
CREATE BDAT	RENAME	TRANSFER
OUTPUT	RE-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.

### CHECKREAD of SRM and HFS Volumes

For SRM, CHECKREAD is implemented as a no-op, because the CHECKREAD function is already performed (by the SRM system) for every BASIC operation that reads or writes an SRM file. For HFS, CHECKREAD is also implemented as a no-op.

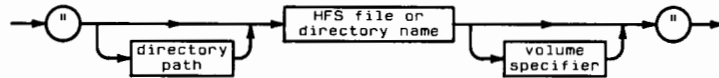
Option Required HFS  
 Keyboard Executable Yes  
 Programmable Yes  
 In an IF... THEN Yes



This statement modifies the group identifier of an HFS file or directory.



literal form of HFS file or directory specifier:



Item	Description	Range
group identifier	numeric expression, rounded to an integer	0 thru 32 767
HFS file or directory specifier	string expression specifying a file on an HFS volume	(see drawing)
directory path	literal	(see MASS STORAGE IS)
HFS directory or file name	literal	1 to 14 characters (see Glossary)
volume specifier	literal	(see MASS STORAGE IS)

**Example Statements**

```
CHGRP New_group_id,"/DirPath/HFSfile"
CHGRP 10,"TheirFile"
```

## Semantics

In order to execute CHGRP, you must currently own the file or directory; that is, the owner identifier must be 18, which is *always* the owner identifier for BASIC. If you change the ownership with CHOWN, then you *cannot* subsequently use CHGRP to change the group identifier of the file.

If no directory path is specified, the current working directory is assumed. If no volume is specified, the current default volume is assumed.

### Major Objective: HP-UX Compatibility

This keyword is implemented primarily for compatibility with the HP-UX operating system. Group identifiers allow files and directories to be accessed by all users in the same group, while restricting access to users in all other groups. Therefore, you can use CHGRP to give group permissions to a specific HP-UX group. BASIC will no longer have group permissions on the file, but it will retain owner permissions (unless ownership is changed—such as with CHOWN).

For a list of the group identifiers used on an HP-UX system, see your HP-UX system administrator or look at the identifiers listed in the `/etc/group` file on the HP-UX system. This file could contain the following entry, which defines the relationship between the group named `workstation` and the group identifier 9:

```
workstation::9:basic,pws
```

If this group identifier is currently being used on an HP-UX system that is to share a disc with BASIC, then the HP-UX system administrator may need to change the `/etc/group` file so that BASIC is assigned this group identifier. Otherwise, all other HP-UX users with this group identifier will have the current group access permissions to *all* BASIC files and directories.

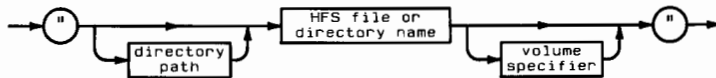
Note that the Series 200/300 BASIC and Pascal operating systems have the same group identifier of 9; however, Pascal has an owner identifier of 17.

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

This statement modifies the owner identifier of an HFS file or directory.



literal form of HFS file or directory specifier:



Item	Description	Range
owner identifier	numeric expression, rounded to an integer	0 thru 32 767
HFS file or directory specifier	string expression specifying a file on an HFS volume	(see drawing)
directory path	literal	(see MASS STORAGE IS)
HFS directory or file name	literal	1 to 14 characters (see Glossary)
volume specifier	literal	(see MASS STORAGE IS)

## Example Statements

```

CHOWN Other_owner_id, "/DirPath/HFSfile"
CHOWN 23, "HerFile"
  
```

## Semantics

In order to execute CHOWN, you must currently own the file or directory; that is, the owner identifier must be 18, which is *always* the owner identifier for BASIC. If you change the ownership with CHOWN, then you *cannot* subsequently use CHOWN to change the owner identifier of the file or directory.

If no directory path is specified, the current working directory is assumed. If no volume is specified, the current default volume is assumed.

### Major Objective: HP-UX Compatibility

This keyword is implemented primarily for compatibility with the HP-UX operating system. Owner identifiers allow files and directories to have certain access permissions only available to the owner, while restricting access to all other users. Therefore, CHOWN can be used to give an HP-UX user the owner permissions of files and directories. BASIC, however, will still have group permissions of the file (unless the group identifier is changed—such as with CHGRP).

For a list of the owner identifiers used on an HP-UX system, see your HP-UX system administrator or look at the identifiers listed in the `/etc/passwd` file on the HP-UX system. It could contain the following entry, which defines the relationship between the owner named `basic` and the owner identifier 18:

```
basic:*:18:9:#BASIC workstation user:/WORKSTATIONS:/bin/false
```

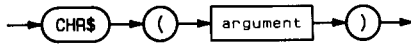
If this owner identifier is currently being used on an HP-UX system that is to share a disc with BASIC, then the HP-UX system administrator will need to change the `/etc/passwd` file so that BASIC is assigned this owner identifier. Otherwise, any HP-UX user with this owner identifier will have the current owner access permissions to *all* BASIC files and directories.

Note that the Series 200/300 Pascal system has an owner identifier of 17; however, BASIC and Pascal operating systems have the same group identifier of 9.



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	Range
argument	numeric expression, rounded to an integer	0 thru 255

## Example Statements

```

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

# CHRX

---

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

This function returns the number of columns (width) of a character cell (on bit-mapped alpha/graphics displays) or 0 (on non-bit-mapped alpha displays).



## Example Statements

```
CHRX  
ALLOCATE INTEGER Char_cell(1:CHRY,1:CHRX)
```

## Semantics

Character cells are 16 (rows) by 8 (columns) for high-resolution bit-mapped alpha displays, and 15 (rows) by 12 (columns) for medium-resolution bit-mapped alpha displays.

If the alpha display is not bit-mapped (that is, if the alpha is separate from the graphics raster, and is generated by character-generator-ROM hardware), then this function returns 0.

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

This function returns the number of rows (height) of a character cell (on bit-mapped alpha/graphics displays) or 0 (on non-bit-mapped alpha displays).



## Example Statements

```
CHRY  
ALLOCATE INTEGER Char_cell(1:CHRY,1:CHRX)
```

## Semantics

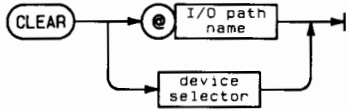
Character cells are 16 (rows) by 8 (columns) for high-resolution bit-mapped alpha displays, and 15 (rows) by 12 (columns) for medium-resolution bit-mapped alpha displays.

If the alpha display is not bit-mapped (that is, if the alpha is separate from the graphics raster, and is generated by a character-generator-ROM hardware), then this function returns 0.

# 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	Range
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 pre-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.

# CLEAR ERROR

---

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

This statement resets most error indicators (ERRN, ERRM\$, etc.) to their power-up values.



## Example Statements

```
CLEAR ERROR  
IF Done THEN CLEAR ERROR
```

## Semantics

CLEAR ERROR affects error indications as shown below:

ERRN—subsequently returns 0.

ERRM\$—subsequently returns the null string (a string with length of 0).

ERRL—subsequently returns 0.

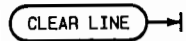
ERRLN—subsequently returns 0.

ERRDS—not affected.

# CLEAR LINE

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

This statement clears the keyboard input line. Executing this statement is equivalent to pressing `Shift-Clear line` (`CLR LN` on a 98203 keyboard).



## Example Statements

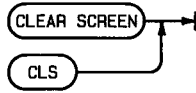
```
CLEAR LINE  
IF Flag THEN CLEAR LINE
```

# CLEAR SCREEN

---

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

This statement clears the contents of the alpha display.



## Example Statements

```
CLS  
CLEAR SCREEN  
IF Loop_count=1 THEN CLEAR SCREEN
```

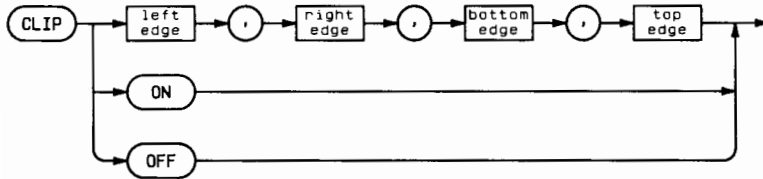
## Semantics

When this statement is executed it clears alpha memory. It has the same effect as executing `OUTPUT KBD;CHR$(255)&"K"`; or pressing the `Clear display` (`(CLR SCR)`) key.



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	Range
left edge	numeric expression in current units	—
right edge	numeric expression in current 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.

## CMPLEX

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

This function creates a complex number with the first argument representing the real part and the second argument representing the imaginary part. The arguments may be REAL, INTEGER, or COMPLEX expressions.



Item	Description/Default	Range Restrictions
argument	numeric expression	any valid INTEGER, REAL, or COMPLEX expression

### Example Statements

```
C=CMPLEX(-2,1)
Result=CMPLEX(-2.356,.0012)
Complex_value=CMPLEX(Real_part,Imaginary_part)
```

### Semantics

Arguments used by this function are converted to two 8-byte (64-bit) floating-point values and handled accordingly. If arguments are COMPLEX, then only the real part of that COMPLEX argument is used.

## COLOR

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



Item	Description	Range
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 COMPLEX Result
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, SCRATCH BIN, 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 begin (such as LOAD "File",1).

When COM allocation is performed at prerun, the new program's COM area is compared to the COM area currently in memory. When comparing the old and new areas, BASIC 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, BASIC 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 BASIC. 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 stated below), 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.

The total number of COM elements is limited to a maximum memory usage of  $2^{24}-1$ , or 16 777 215, bytes (or limited by the amount of available memory, whichever is less).

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, INTEGER, or COMPLEX. 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 REAL unless their type has been changed to INTEGER or COMPLEX.

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 statement 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 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 (limited only by available memory). 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.

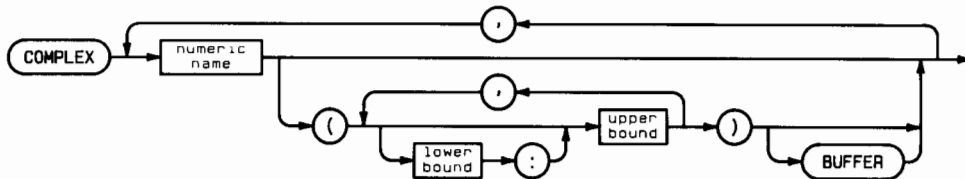
#### **Declaring Buffers**

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. String arrays cannot be declared to be buffers.

# COMPLEX

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

This statement declares COMPLEX variables and arrays and reserves storage for them. (For information about COMPLEX 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 Statement

```
COMPLEX X,Y,Z
COMPLEX Array(-23:2,26)
COMPLEX A(512) BUFFER
```



## Semantics

Each COMPLEX variable or array element consists of two floating-point values, one for the real part and one for the imaginary part of the COMPLEX number. Each complex value requires sixteen bytes of 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 COMPLEX elements is limited by the fact that the maximum memory usage for *all* variables—COMPLEX, INTEGER, REAL, and string—within any context is  $2^{24}-1$ , or 16 777 215, bytes (or limited by the amount of available memory, whichever is less).

## Declaring Buffers

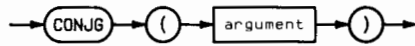
To declare COMPLEX variables to be buffers, each variable's name must be followed by the keyword BUFFER; the designation BUFFER applies only to the variable which it follows.



# CONJG

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

This function returns the complex conjugate of a COMPLEX number.



Item	Description/Default	Range Restrictions
argument	numeric expression	any valid INTEGER, REAL, or COMPLEX value

## Example Statements

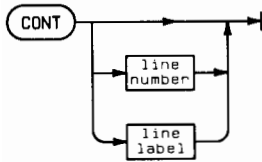
```
X=CONJG(Complex_expr)
Y=CONJG(Real_expr)
Z=CONJG(Integer_expr)
Result=CONJG(CMPLX(2.1, -8))
```

## Semantics

The complex conjugate of a COMPLEX number  $\text{CMPLX}(X,Y)$  is  $\text{CMPLX}(X,-Y)$ . That is, the imaginary part of the argument is negated. An INTEGER or REAL argument is returned unchanged.

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	Range
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

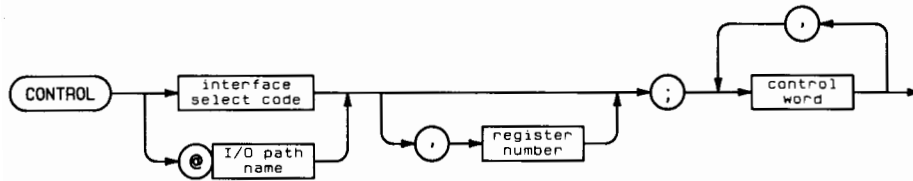
Continue can be executed by pressing the **CONTINUE** key (**F2** in the System menu of an ITF keyboard), or by executing a CONT command. Variables retain their current values whenever CONT is executed. CONT causes the program to resume execution at the next statement which would have occurred unless 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.)



Item	Description	Range
interface select code	numeric expression, rounded to an integer	1 thru 32 (interface-dependent)
I/O path name	name assigned to a device, devices, mass storage file, or buffer	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 $2^{31}-1$ (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 or HPUX file or a buffer. I/O path names have an association table that can be accessed 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 this 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.

---

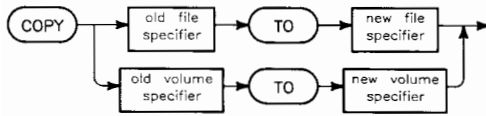
## **CONVERT**

See the ASSIGN statement

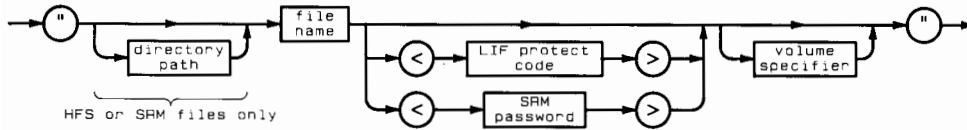
# COPY

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

This statement allows copying of an individual files or an entire disc.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
volume specifier	string expression	(see MASS STORAGE IS)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
LIF protect code	literal; first two non-blank characters are significant	> not allowed
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	string expression	(see MASS STORAGE IS)

## Example Statements

```
COPY "OLD_FILE" TO "New_file"  
COPY File$ TO File$$Other_volume$
```

```
COPY "/Dir_1/File_1" TO "Dir_3/File_1"  
COPY "File:INTERNAL" TO "File:REMOTE 21,0"  
COPY Dir_path$$File$$Vol$$ TO "File:,700"
```

```
COPY Left_disc$ TO Right_disc$  
COPY ":",700" TO ":",700,1"  
COPY ":",4,1" TO ":",4,0"
```

## 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 (LIF directories) may be specified for the new file, to prevent accidental erasure, etc. The old file and the new file can exist in the same directory, but the new file name must be unique.

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 volume specifier (msvs) is omitted from a file specifier, the MASS STORAGE IS device is assumed.

If the directory path is also omitted, the MASS STORAGE IS directory is assumed.

### Copying an Entire LIF or HFS Volume

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

When copying an entire volume, the volume specifiers must be unique. File names are not allowed. Disc-to-disc copy time is dependent on media type and interleave factors.

### **HFS Permissions**

With HFS, COPY allows copying of individual files and volumes. HFS directories cannot be copied.

In order to COPY a file on an HFS volume, you need to have R (read) permission on the source file, as well as X (search) permission on the parent directory and all other superior directories. In addition, you will need W (write) and X (search) permission on the destination file's parent directory, as well as X (search) permission on all other superior directories.

### **HFS File Headers**

When copying a file from LIF or SRM to HFS, a special header is added to the beginning of that file. This action is taken because that is the only way to "type" files (which would otherwise be "typeless"). When copying a file from HFS to LIF or SRM volumes, this file header is removed (since these volumes have typed files). Note that BASIC handles the file headers automatically and requires no special treatment in programs that use these files.

When copying a SYSTM file from LIF or SRM volumes to HFS volumes, it will be given a header and will remain a SYSTM file. However, it will not be bootable. Conversely, copying a bootable HP-UX file from HFS to LIF or SRM will result in an HP-UX file that is not bootable. (STORE SYSTEM will copy the current BASIC system from memory onto an HFS volume only in the root directory. STORE SYSTEM will error if the HFS directory is not the root directory (/). If the destination file is in the HFS volume's root directory, it will be a bootable system.)

### **SRM Passwords**

With SRM, COPY allows copying of individual files. SRM directories and volumes cannot be copied.

In order to COPY an SRM file, you need to have R (read) access capability on the file, on the parent directory, and on all other superior directories. You must also have W (write) access capability on the destination directory, as well as R access capability on all superior directories.

Although you may include a password in the new file specifier, the system ignores the password. If you wish to protect access to the new file, you must assign the password with PROTECT.

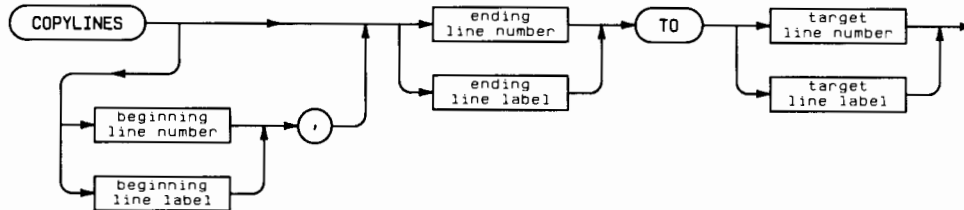


# COPYLINES

Option Required      EDIT and 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	Range
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 exceptions: line number references in lines not being copied remain linked to the source lines rather than being renumbered; 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.

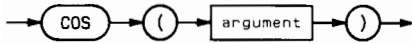
If you try to copy a program segment to a line number *contained* in the segment, an error will be reported and no copying will occur.

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.

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

This function returns the cosine of the angle represented by the argument. The range of the returned real value is -1 through +1.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle when INTEGER or REAL argument  numeric expression in radians when COMPLEX argument	absolute values less than 1.708 312 772 2 E+10 deg. or 2.981 568 244 292 04 E+8 rad. for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

**Examples Statements**

```

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

## Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

To compute the COS of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula used for computing the COS of a COMPLEX argument is:

$$\text{CMPLX}(\text{COS}(\text{Real}) * \text{COSH}(\text{Imag}), -\text{SIN}(\text{Real}) * \text{SINH}(\text{Imag}))$$

where **Real** is the real part the COMPLEX argument and **Imag** is the imaginary part of the COMPLEX argument. Some values of a COMPLEX argument may cause errors in this computation. For example,

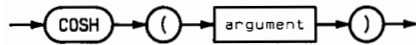
$$\text{COS}(\text{CMPLX}(0, \text{MAXREAL}))$$

will cause error 22 due to the **COSH(Imag)** calculation.

Note that any COMPLEX function whose definition includes a sine or cosine function will be evaluated in the radian mode regardless of the current angle mode (i.e. RAD or DEG).

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

This function returns the hyperbolic cosine of a numeric expression.



Item	Description/Default	Range Restrictions
argument	numeric expression	-710 through 710 for INTEGER or REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

### Example Statements

```

Result=COSH(10.3499)
PRINT "Hyperbolic Cosine = ";COSH(Expression)
  
```

### Semantics

If an INTEGER or REAL argument is given, this function returns a REAL value. If a COMPLEX argument is given, this function returns a COMPLEX value.

### Range Restriction Specifics

The formula used for computing COSH is as follows:

$$(EXP(Argument)+EXP(-Argument))/2$$

where **Argument** is the argument of the COSH function. Some arguments may cause errors in intermediate values computed during this computation. For example,

```
COSH(MAXREAL)
```

will cause error 22 due to the EXP(MAXREAL) computation.

## COUNT

---

See the CAT and TRANSFER statements.



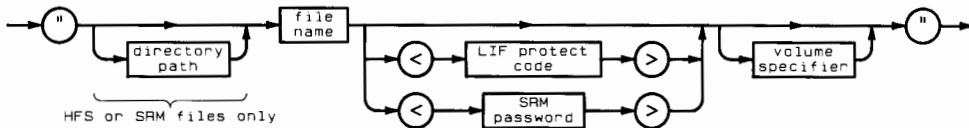
Option Required MS  
 Keyboard Executable Yes  
 Programmable Yes  
 In an IF...THEN Yes



This statement creates an HP-UX file.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
LIF protect code	literal; first two non-blank characters are significant	> not allowed
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	literal	(see MASS STORAGE IS)
number of records	numeric expression, rounded to an integer	1 thru $2^{31} - 1$

### Example Statements

```

CREATE File_spec$,N_records
CREATE "HPUX_file",12
CREATE "OnLIF<pc>",50*N
  
```

## Semantics

CREATE creates a new file of type HP-UX on the default or specified volume or hierarchical directory. A corresponding directory entry is also made. The name of the newly created file must be unique within its directory. CREATE does not open the file; that is performed by ASSIGN. In the event of an error, no directory entry is made and the file is not created.

The number of records parameter specifies how many *logical* records are to be initially allocated to the file. The logical record size is always 1 for HP-UX files. On LIF volumes, the number of records allocated for the file is fixed; however, with HFS and SRM volumes, files are extensible (see the following explanation of extensible files).

The data representation used in the file depends on the FORMAT option used in the ASSIGN statement used to open the file. See ASSIGN for details.

### Extensible Files (HFS and SRM Volumes Only)

If the file is created on an HFS or SRM volume, the file is “extensible”. With HFS volumes, the initial size of the file is 0, but the file will automatically be extended as many bytes as necessary whenever an OUTPUT operation would otherwise overflow the file. On SRM volumes, the “number of records” parameter determines the “extent size” of the file (that is, the amount of space automatically appended to the file whenever it is extended). “Preallocating” the file on an SRM volume (initially creating a file of sufficient size) will improve the data transfer rate with extensible files, because the file system will not have to extend the file during data transfer operations.

### LIF Protect Codes

A protect code is not allowed on an HP-UX file.

### HFS Permissions

In order to create a file on an HFS volume, you need to have W (write) and X (search) permission of the immediately superior directory, as well as X (search) permission on all other superior directories.

When a file is created on an HFS volume, access permission bits are set to **RW-RW-RW-**. (You can modify them with PERMIT, if desired.)



**SRM Access Capabilities**

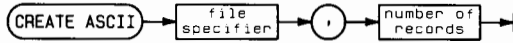
In order to **CREATE** an HP-UX file in an SRM directory, you need to have **READ** and **WRITE** capabilities on the immediately superior directory, as well as **READ** capabilities on all other superior directories.

When a file is created on an SRM volume, all access capabilities are public. Including an SRM password in the file specifier does not protect the file. You must use **PROTECT** to assign a password. You will not receive an error message for including a password, but a password in the **CREATE** statement is ignored.

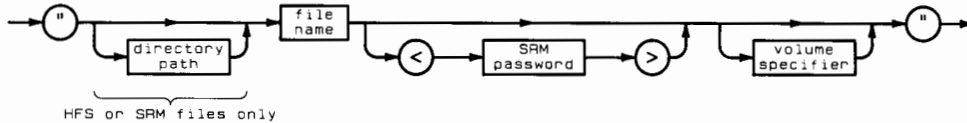
# CREATE ASCII

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

This statement creates an ASCII file.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
number of records	numeric expression, rounded to an integer	1 thru $2^{31} - 1$
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	literal	(see MASS STORAGE IS)

## Example Statements

```
CREATE ASCII "TEXT",100
CREATE ASCII Name$&": ,700,1".Length
CREATE ASCII "/Dir1/Dir2/AsciiFile",25
```

## **Semantics**

CREATE ASCII creates a new ASCII file and directory entry on the mass storage media. The name of the newly created ASCII file must be unique within its containing directory. CREATE ASCII does not open the new file; that is performed by the ASSIGN statement. In the event of an error, no directory entry is made and the file is not created.

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.

### **Extensible Files (HFS and SRM Volumes Only)**

If the file is created on an HFS or SRM volume, the file is “extensible”. With HFS volumes, the initial size of the file is the size specified in the CREATE ASCII statement, but the file will automatically be extended as many bytes as necessary whenever an OUTPUT operation would otherwise overflow the file. On SRM volumes, the “number of records” parameter multiplied by the record size (256 for ASCII files) determines the “extent size” of the file (that is, the amount of space automatically appended to the file whenever it is extended). “Preallocating” the file on an SRM or HFS volume (initially creating a file of sufficient size) will improve the data transfer rate with extensible files, because the file system will not have to extend the file during data transfer operations.

### **LIF Protect Codes**

On a LIF disc, a protect code is not allowed on an ASCII file. Including a protect code in the CREATE ASCII statement will give an error.

### **HFS Permissions and File Headers**

In order to create a file on an HFS volume, you need to have W (write) and X (search) permissions on the immediately superior directory, as well as X (search) permissions on all other superior directories.

On HFS volumes, access permission bits are set to **RW-RW-RW-** when an ASCII file is created. (You can modify them with PERMIT, if desired.)

On an HFS volume, the first 512 bytes of an ASCII file are used by the BASIC file system to describe the file’s type (this is the only way for BASIC to create a “typed” file on an HFS volume, since HFS files are otherwise “typeless”). This file header is handled automatically by BASIC, but it should be skipped when reading and writing the file with other HP-UX languages. See the “Porting and Sharing Files” chapter of *BASIC Programming Techniques* for details.

**SRM Access Capabilities**

In order to create an ASCII file in an SRM directory, you need to have R (read) and W (write) capabilities on the immediately superior directory, as well as R capability on all other superior directories.

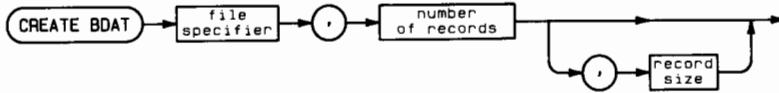
When an ASCII file is created on an SRM volume, all access capabilities are public. Including a password in the file specifier does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but SRM passwords in the CREATE ASCII statement are ignored.

# CREATE BDAT

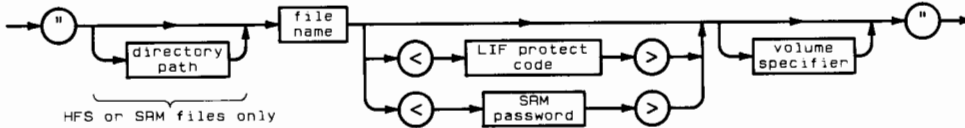
Option Required      None  
 Keyboard Executable    Yes  
 Programmable            Yes  
 In an IF...THEN        Yes



This statement creates a BDAT file.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
number of records	numeric expression, rounded to an integer	1 thru $2^{31} - 1$
record size	numeric expression, rounded to next even integer (except 1), which specifies bytes/record; Default = 256	1 thru 65 534
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
LIF protect code	literal; first two non-blank characters are significant	> not allowed
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	string expression	(see MASS STORAGE IS)

## Example Statements

```
CREATE BDAT "File",Records,Rec_size
CREATE BDAT "George",48
CREATE BDAT "Protected<PC>",Length,128
CREATE BDAT Name$&Volume$,Bytes,1
CREATE BDAT "/Dir1/Dir2/BDATfile",25,128
CREATE BDAT "Dir/File:REMOTE",10
```

## Semantics

CREATE BDAT creates a new BDAT file and directory entry on the mass storage media. The name of the newly created BDAT file must be unique within its containing directory. CREATE BDAT does not open the file; that is performed by the ASSIGN statement. In the event of an error, no directory entry is made and the file is not created.

A sector at the beginning of the file is reserved for system use. This sector cannot be directly accessed by BASIC programs.

### Extensible Files (HFS and SRM Volumes Only)

If the file is created on an HFS or SRM volume, the file is “extensible”. With HFS volumes, the initial size of the file is the size specified in the CREATE BDAT statement, but the file will automatically be extended as many bytes as necessary whenever an OUTPUT operation would otherwise overflow the file. On SRM volumes, the “number of records” parameter multiplied by the record size determines the “extent size” of the file (that is, the amount of space automatically appended to the file whenever it is extended). “Preallocating” the file on an SRM volume (initially creating a file of sufficient size) will improve the data transfer rate with extensible files, because the file system will not have to extend the file during data transfer operations.

### LIF Protect Codes

On LIF volumes, an optional protect code may be specified; the first two characters become the protect code of the file. (You can modify the protect code with PROTECT, if desired.)

**HFS Permissions and File Headers**

In order to create a file on an HFS volume, you need to have W (write) and X (search) permission of the immediately superior directory, as well as X (search) permission on all other superior directories.

When a file is created on an HFS volume, access permission bits are set to **RW-RW-RW-**. (You can modify them with PERMIT, if desired.)

On HFS volumes, the first 512 bytes of a BDAT file are used by the BASIC file system to describe the file's type (this is the only way for BASIC to create a "typed" file on an HFS volume, since HFS files are otherwise "typeless"). This file header is handled automatically by BASIC, but it should be skipped when reading and writing the file with other HP-UX languages. See the "Porting and Sharing Files" chapter of *BASIC Programming Techniques* for details.

**SRM Access Capabilities**

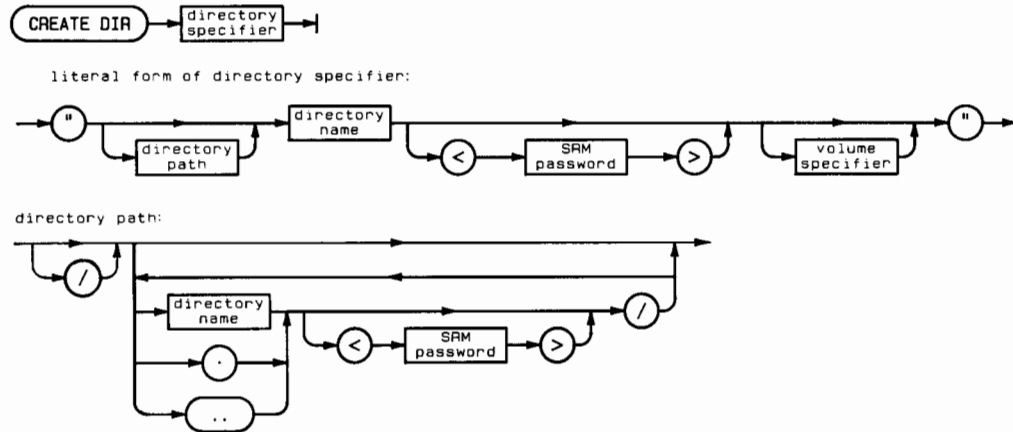
In order to create a file in an SRM directory, you need to have R (read) and W (write) capabilities on the immediately superior directory, as well as R capability on all other superior directories.

When a file is created on an SRM volume, all access capabilities are public. Including an SRM password in the file specifier does not protect the file. You must use PROTECT to assign a password. You will not receive an error message for including a password, but a password in the CREATE BDAT statement is ignored.

# CREATE DIR

Option Required	SRM & DCOMM, or HFS
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 directory of an SRM or HFS volume.



Item	Description	Range
directory specifier	string expression	(see drawing)
directory path	literal	(see drawing)
directory name	literal	depends on volume's format (14 characters for HFS; 16 characters for SRM; see Glossary for details)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	literal	(see MASS STORAGE IS)



## 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<SRM_RW_pass>/New_dir"  
CREATE DIR "Dir3/Dir4:,700"
```

## Semantics

This statement creates a directory and a corresponding directory entry in the current working directory or specified 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.

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.

## HFS Permissions

In order to create a directory on an HFS volume, you need to have W (write) and X (search) permission of the immediately superior directory, as well as X (search) permission on all other superior directories.

When a directory is created on an HFS volume, access permission bits are set to **RWXRXRX**. (You can modify them with PERMIT, if desired.)

As each directory or data file is created within an HFS directory, a 32-byte record identifying the addition is added to the DIR file.

## SRM Access Capabilities

In order to create a directory in an SRM directory, you need to have R (read) and W (write) capabilities on the immediately superior directory, as well as R (read) capabilities on all other superior directories.

When a directory is created on an SRM volume, all access capabilities are public. Including an SRM password in the directory specifier does not protect the file. You must use PROTECT to assign a password. You will not receive an error message for including a password, but a password in the CREATE DIR statement is ignored.

As each directory or data file is created within an SRM directory, a 24-byte record identifying the addition is added to the DIR file.

# 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 CRT display.

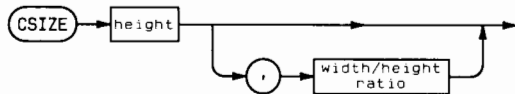


## Example Statements

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

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

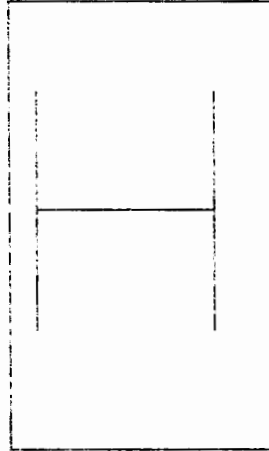
## Example Statements

```
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 compiled Pascal programs, linked to BASIC by 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 CSUB (instead of SUB). 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, or re-storing the line by pressing the **Return** or **ENTER** keys. Operations which do not check syntax are allowed. This includes things like scrolling and renumbering.

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. The only statements which can be entered directly after a CSUB are SUB and DEF FN. As always, these must be entered at the end of the program.

## CSUM

---

See the MAT statement.

## CYCLE

---

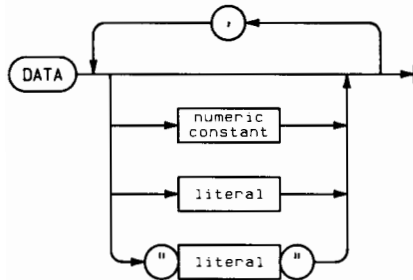
See the OFF CYCLE and the ON CYCLE statements.



# Notes

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	Range
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 <b>ANY CHAR</b> key ( <b>f7</b> ) in the system menu of an ITF keyboard)	—

## 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.

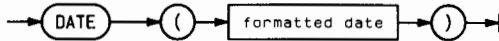
A numeric constant must be read into a variable which can store the value it represents. The computer cannot determine the intent of the programmer; although attempting to read a string value into a numeric variable will generate an error, numeric constants will be read into string variables with no 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.

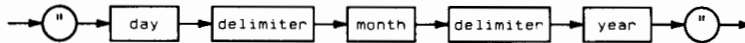


Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

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	Range
formatted date	string expression	(see drawing and text)
day	integer constant	1 thru end-of-month
month	literal (lettercase ignored)	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
year	integer constant	1900 thru 2079

### 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.

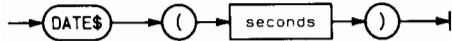
Specifying an invalid date, such as the thirty-first of February, will result in an error.

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.

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	Range
seconds	numeric expression	-4.623 683 256 E+13 thru 4.653 426 335 039 9 E+13

## 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.

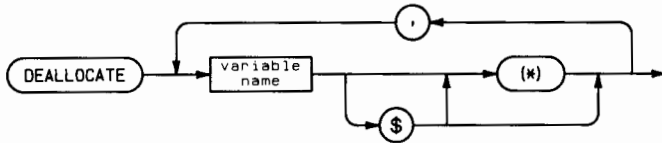
The first letter of the month is capitalized and the rest are lowercase characters.

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	Range
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.

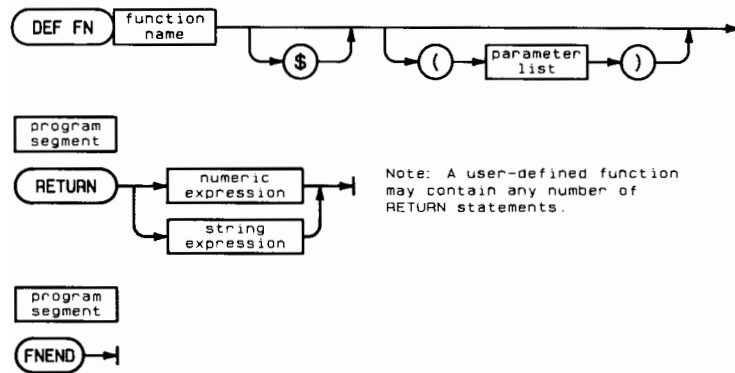
Strings and arrays must be deallocated completely. Deallocation of an array is requested by the (\*) specifier.

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.

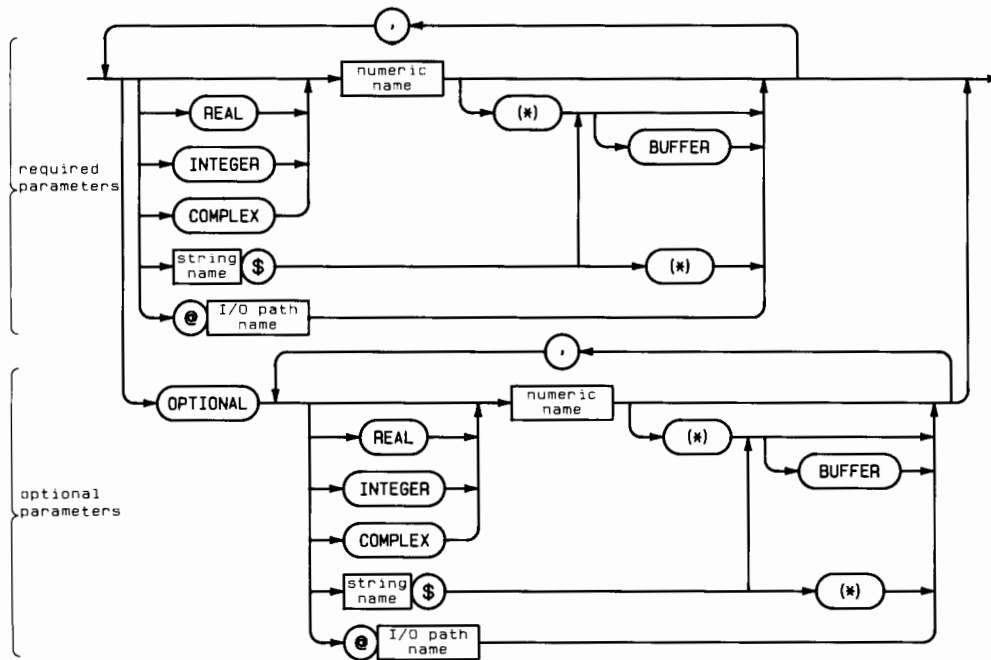
# DEF FN

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.



parameter list:



Item	Description	Range
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 FNComplex(COMPLEX Real_part)
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. RETURN <integer expression> yields a real function result. RETURN <complex expression> yields a complex function result.

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 <b>IF . . . THEN</b>	Yes

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



### Semantics

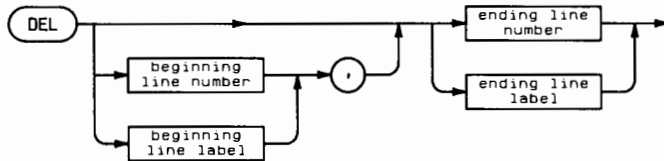
All functions which return an angle will return an angle in degrees. All operations with parameters representing angles will interpret the angle in degrees.

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).



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

This command deletes program line(s).



Item	Description	Range
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 than 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**

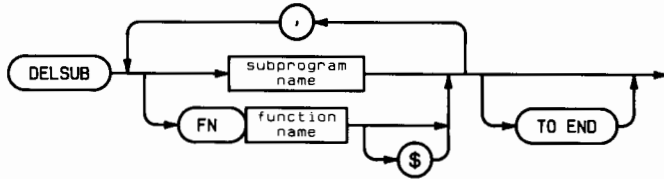
---

See the TRANSFER statement.

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	Range
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.

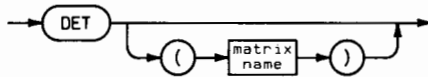
You cannot delete:

- 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.

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

This function returns the determinant of a matrix.



Item	Description	Range
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.

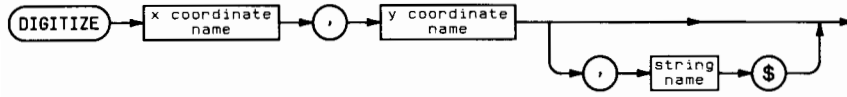
The determinant is significant as an indication of whether an inverse is valid. If the determinant of a matrix equals 0, then the matrix has no inverse. If the determinant is very small compared with the elements of its matrix, then the inverse may be invalid and should be checked.

If the matrix is COMPLEX, the value returned is COMPLEX. Otherwise, the value returned is REAL.

# 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	Range
x coordinate	name name of a numeric variable	any valid name
y coordinate name	name of a numeric variable	any valid name string
string name	name of a string variable	any valid name string

## 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:

`Return`, `Enter`, `Stop`, `ENTER`, `EXECUTE`, `STOP`, `PAUSE`, `STEP`, `CONTINUE`, `ENTER`, `EXEC`, `STOP`, `PSE`, `STEP`, and `CONT`.

Note that if ON KBD is in effect while executing DIGITIZE when the keyboard or mouse or knob is the GRAPHICS INPUT IS device, the keys listed above will not be placed into KBD\$. Once the DIGITIZE is complete, these keys will be placed into KBD\$.

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.

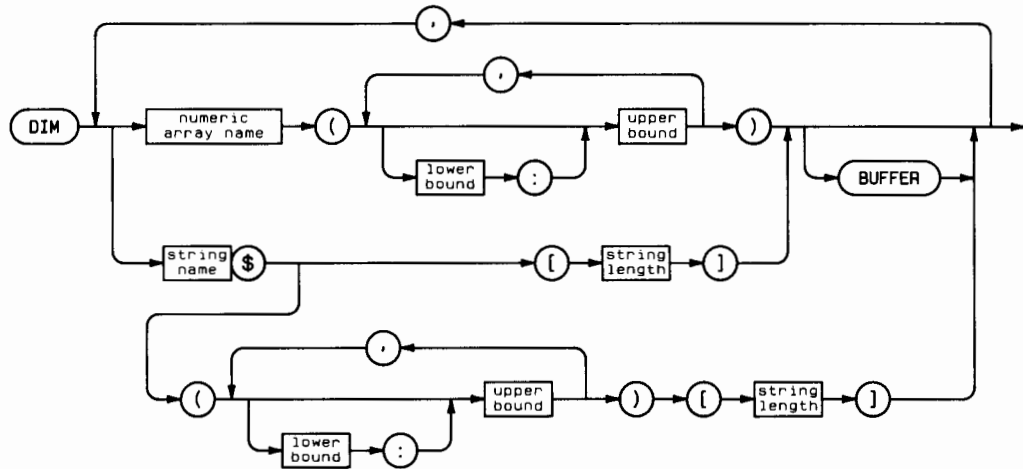
If the locator device (e.g., stylus or puck) goes out of proximity, a “button 7” is indicated in the “button number” bytes. Bytes 7 and 8 will be exactly “64” regardless of whether any actual buttons are being held down at the time. Proximity is reported only from HP-HIL locators; the HP 9111A always 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	Range
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_scalar[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 total number of variables is limited by the fact that the maximum memory usage for *all* variables—COMPLEX, INTEGER, REAL, and string—within any context is  $2^{24}-1$ , or 16 777 215, bytes (or limited by the amount of available memory, whichever is less).

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 a keyword BUFFER; the designation BUFFER applies only to the variable which it follows.

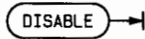
String arrays **cannot** be declared to be buffers.

# 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

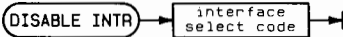
If an event occurs while the event-initiated branches are disabled, only the first occurrence of each event is logged; there is no record of how many of each type of event has occurred.

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, and 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.



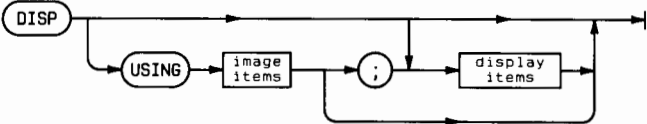
## Example Statements

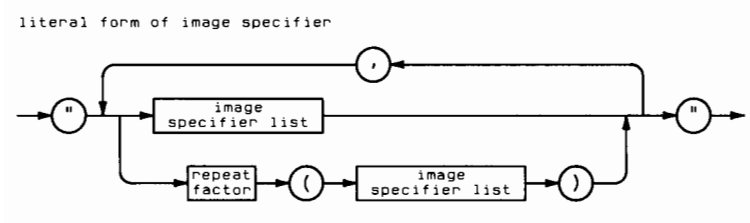
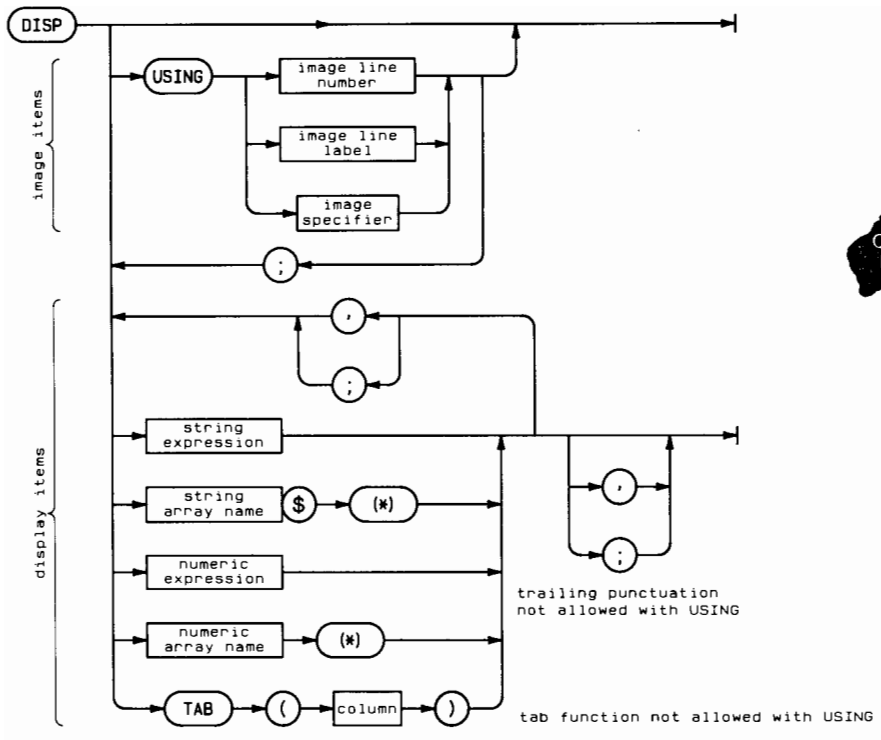
```
DISABLE INTR 7  
DISABLE INTR Isc
```

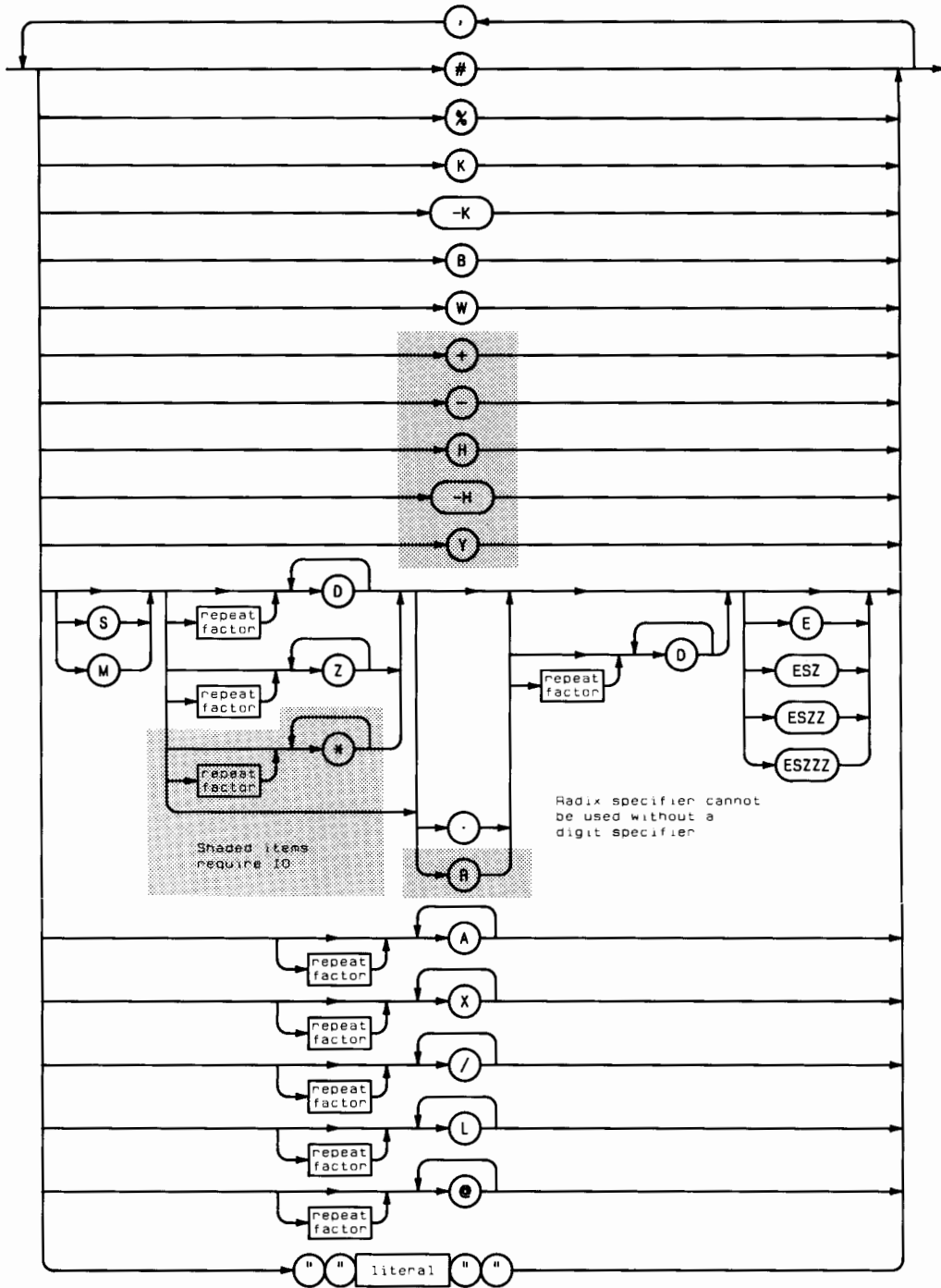
# DISP

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.







Item	Description	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	1 thru screenwidth
image specifier list	literal	(see diagram)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters entered from the keyboard, including those generated using the <code>ANY CHAR</code> key	quote mark not allowed

### Example Statements

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

### 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.

COMPLEX numbers are treated like two REAL numbers separated by a semicolon.

#### 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	form-feed	Clear the display line.
CHR\$(13)	CTRL-M	carriage-return	Move the 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

Entire arrays may be displayed using the asterisk specifier. Each element in an array is treated as a separate item by the DISP 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. COMPLEX array elements are treated as if the real and imaginary parts are separated by a semicolon. The array is output in row major order (rightmost subscript varies fastest).

## 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 punctuation 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 specifier 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 (and the specifier requires a display specifier). If the image specifiers are exhausted before the display items, they are reused, starting at the beginning.

COMPLEX values require 2 REAL image specifiers (i.e. each COMPLEX value is treated like 2 REAL values).

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 DISP statement are shown in the following table:

<b>Image Specifier</b>	<b>Meaning</b>
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)
.	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.

Image Specifier	Meaning
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.

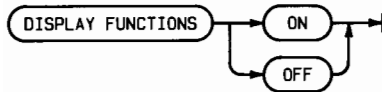


# DISPLAY FUNCTIONS

---

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

This statement enables and disables the display functions mode. This mode causes control characters sent to the CRT to be displayed.



## Example Statements

```
DISPLAY FUNCTIONS ON  
DISPLAY FUNCTIONS OFF  
IF No_ctrl_char THEN DISPLAY FUNCTIONS OFF
```

## Semantics

Except for the carriage-return character, all subsequent control characters sent to the display (while in the display functions mode) **do not** invoke their defined function, but are **only displayed**. The carriage-return is **both displayed and** causes the print position to move to the beginning of the next line (both CR and LF functions invoked).

Also available as CRT CONTROL register 4.

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	Range
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  $SGN(A/B) \times INT(ABS(A/B))$ .

This operator is **not** defined for COMPLEX arguments.

# 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	Range
vector name	name of a one-dimensional numeric array	any valid name

## Example Statements

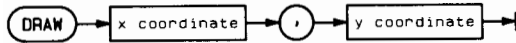
```
PRINT DOT(A,B)
B=DOT(A,A)
```

## 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. If either vector is COMPLEX, the product will be COMPLEX. Otherwise, the product will be of type REAL.

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	Range
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 statement.

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			<i>Note 4</i>
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES and GRID)	X				
Location of labels	<i>Note 1</i>	<i>Note 3</i>		<i>Note 2</i>	

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	Range
argument	numeric expression	--
number of digits	numeric expression, rounded to an integer	--

## Example Statements

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



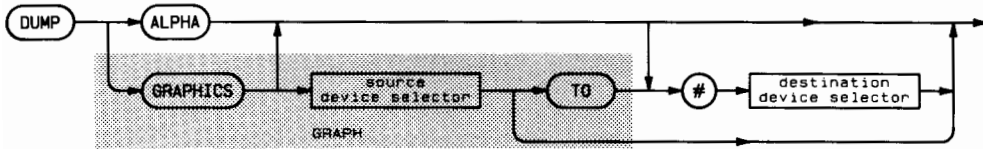
## Semantics

COMPLEX numbers **are not** allowed as arguments to this function.

# DUMP

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

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



Item	Description	Range
source device selector	numeric expression, rounded to an integer; Default = last CRT plotting device	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
```

## Semantics

DUMP ALPHA copies the contents of the alphanumeric display to a printer. With a bit-mapped alpha 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 the current DUMP DEVICE IS device (usually 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. (See the *Configuration Reference* for a complete list of supported HP devices.)

If the destination device is not explicitly specified, it is assumed to be 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 **Break** (**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" (see following description), 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 a non-CRT source which is not the current PLOTTER IS device is explicitly specified, error 708 is reported.

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

## Displays with Nonsquare Pixels

For machines which have a display with nonsquare pixels (such as 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

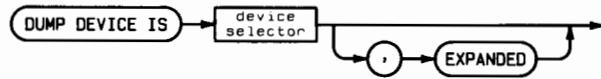
Some printers are not capable of printing 1024 graphics dots per line, so images dumped will be truncated to fit the printer.

---

# 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	Range
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. (See the *Configuration Reference* for a complete list of supported HP devices.)

Specifying EXPANDED results in graphics dumps that are twice as big on each axis (except for displays with non-square 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	Range
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("0101010101010101010101010101",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 uppercase or lowercase letters. 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 3777777777	1 to 11 characters
10	decimal	-2 147 483 648 thru 2 147 483 647	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	Range
"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, respectively.

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	0000000000 thru 3777777777	11 characters
10	decimal	-2 147 483 648 thru 2 147 483 647	11 characters
16	hexadecimal	00000000 thru FFFFFFFF	8 characters



## **ECHO**

---

See the SET ECHO statement.

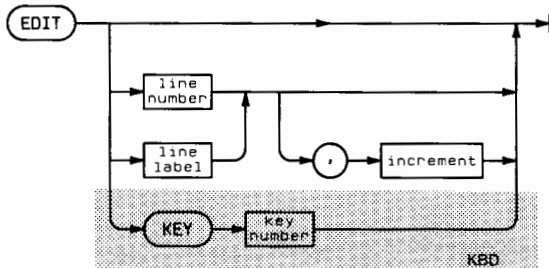
## **EDGE**

---

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

Option Required	EDIT
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	Range
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**, **RESET**, **RUN**, or **STEP** or by executing CAT, LIST (if PRINTER IS CRT), GET, or LOAD. In general any PRINT to the CRT (e.g., executing DISP) will exit you from the EDIT mode. 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 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, but does not require EDIT)**

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 **␣** 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**

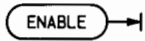
See the **IF...THEN** statement.

## ENABLE

---

Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an <b>IF . . . THEN</b>	Yes

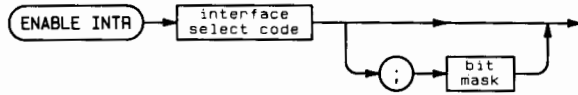
This statement re-enables all event-initiated branches which were suspended by **DISABLE**. **ON END**, **ON ERROR**, and **ON TIMEOUT** are not affected by **ENABLE** and **DISABLE**.



# 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	Range
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	No
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 unserved 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**

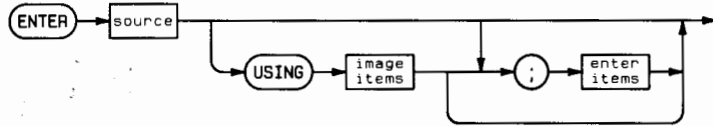
---

See the WHILE statement.

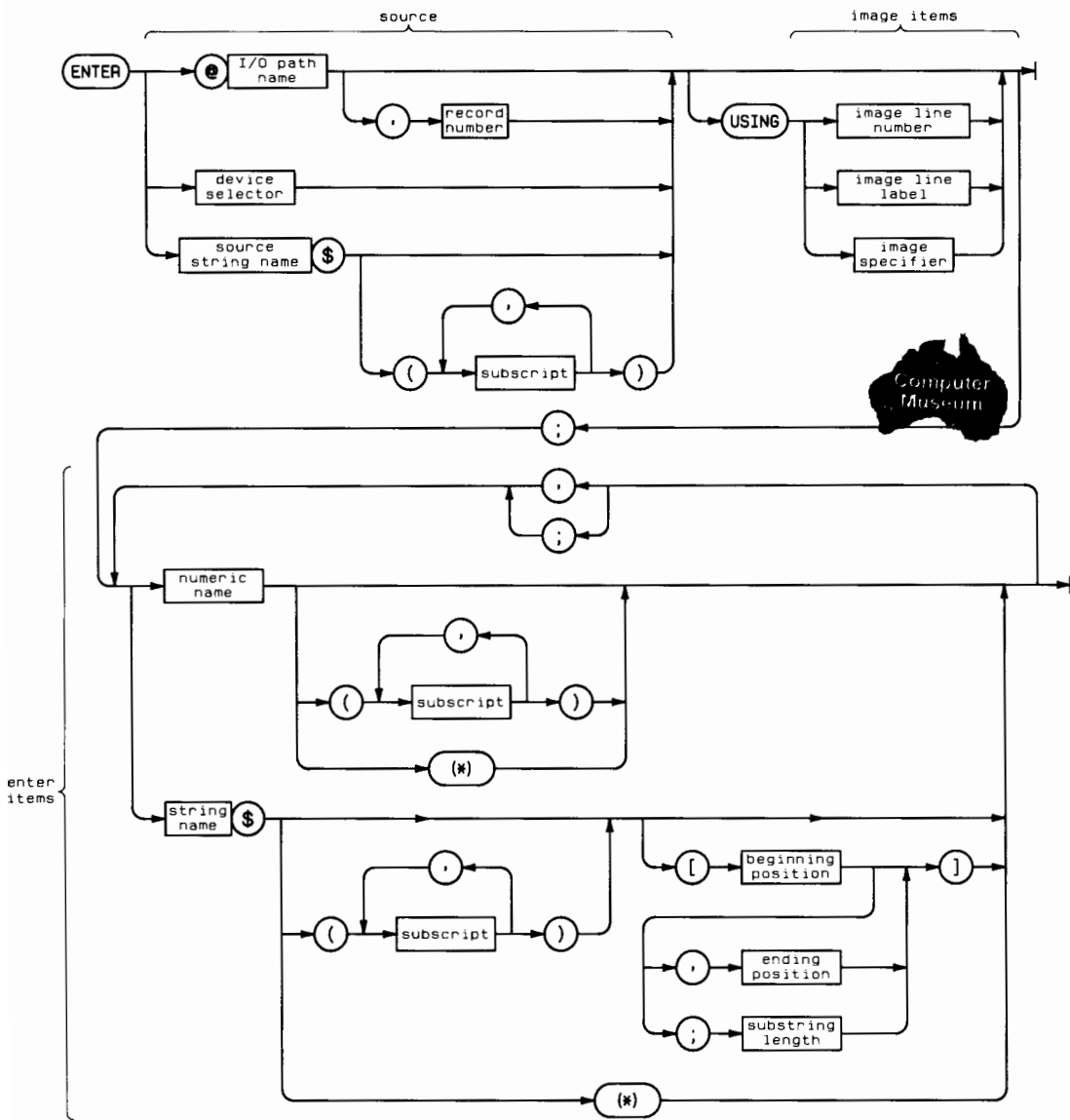
# ENTER

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

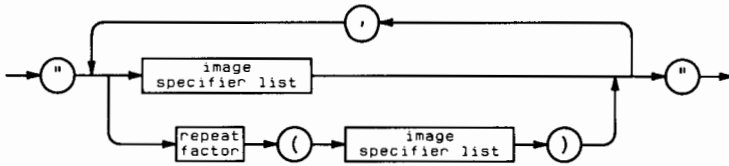
This statement is used to input data from a device, file, string, or buffer and assign the values entered to variables.



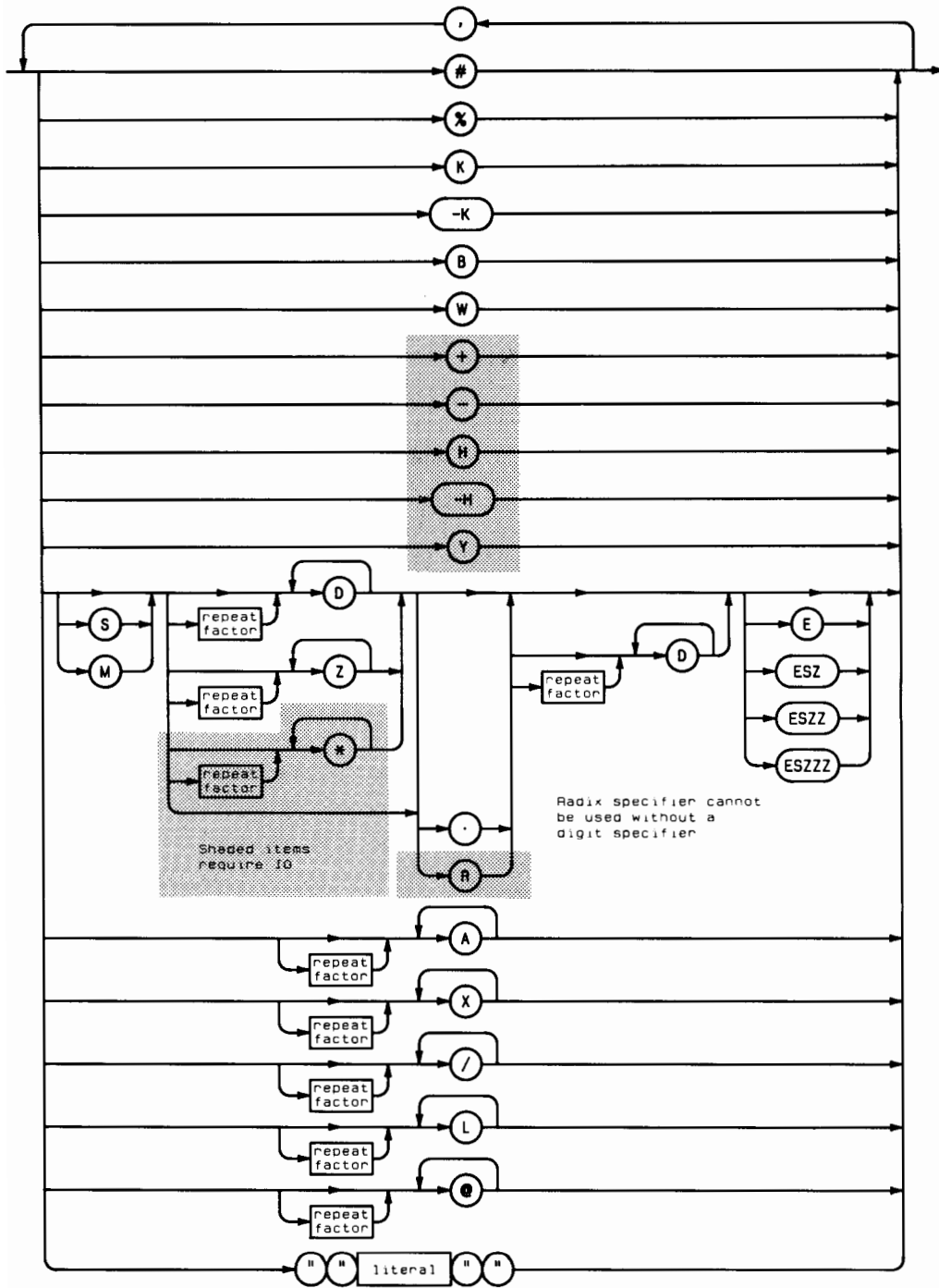
Expanded diagram:



literal form of image specifier



Item	Description	Range
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



## 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 or COMPLEX, the rightmost digits are lost, but any exponent will be built correctly. Overflow occurs only if the exponent overflows. COMPLEX numbers are treated like two real numbers, the first representing the real part and the second representing the imaginary part. When an ENTER statement contains a COMPLEX variable, that variable is satisfied with two REAL values.

### 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). COMPLEX arrays are treated as if they were REAL arrays with twice as many elements (i.e. instead of an  $n$  element array you have a  $2 \times n$  element array).

### 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, BDAT, or HP-UX file. The attributes specified in the ASSIGN statement are used only if the file is a BDAT or HP-UX file. Data read from an ASCII file is always in ASCII format (i.e., you **cannot** use ENTER..USING); however, you can enter the data into a string variable, and then use ENTER..USING from the string variable. Data read from a BDAT or HP-UX file is considered to be in internal representation with FORMAT OFF, and is read as ASCII characters with FORMAT ON.

Serial access is available for ASCII, BDAT, and HP-UX files. Random access is available for BDAT and HP-UX 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 pointer. 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.).

In order to ENTER from a file on an HFS volume, you need to have R (read) permission on the file, as well as X (search) permission on the immediately superior directory and all other superior directories.

In order to read a file in an SRM directory, you need to have READ capability on the immediately superior directory, as well as READ capabilities on all other superior directories. If this capability is not public or if a password protecting this capability was not used at the time the file was assigned an I/O path name (with ASSIGN), an error is reported.

### **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, and sixteen bytes for each COMPLEX value. 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, beginning 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 terminate 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 specifier 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.

ENTER with USING cannot be used to enter data from an ASCII file. Instead, enter the item(s) into a string variable, and then use ENTER with USING to enter the item(s) from the string variable. For instance, use `ENTER @File;String$` then `ENTER String$ USING "5A,X,5DD"; Str2$,Number.`

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

Image Specifier	Meaning
K	Freefield Entry. <b>Numeric:</b> 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.
	<b>String:</b> 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-numeric characters are accepted to fill the character count. Blanks are ignored, other non-numeric characters 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.

When entering FORMAT ON text into string variables, care should be taken to avoid unexpected interactions between terminating on dimensioned string length and termination on line feeds in the text. It is recommended that the string variable be dimensioned at least two characters longer than the text if it will be terminated by a carriage return/line feed. See *Interfacing Techniques*, "Entering Data" for more information.

## **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
```

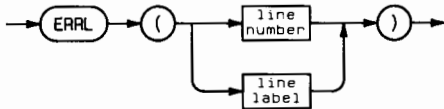
## Semantics

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 selector of the drive containing the file is returned. If the resource is not a device, 0 is returned. If no I/O error has occurred 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.

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	Range
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.

CLEAR ERROR resets ERRL to 0.

## Data Communications

This function returns 0 for all Data Communications errors.

# ERRLN

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

This function returns the number of the program line on which the most recent error occurred.



## Example Statements

```
ERRLN  
IF ERRLN=240 THEN GOSUB Fix_240
```

## Semantics

If no error has occurred since power-on, pre-run, SCRATCH, SCRATCH A, CLEAR ERROR, LOAD, or GET, this function will return a value of 0.

If an error occurs in a TRANSFER 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 ERRLN will not be updated until this time. Therefore, ERRLN 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 (which occur while using the HP 98628 Datacomm Interface).



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, CLEAR ERROR, 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.

CLEAR ERROR resets ERRN to 0.

## ERROR

---

See the CLEAR ERROR, CAUSE ERROR, OFF ERROR and ON ERROR statements.

## ERROR RETURN

---

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

This statement returns program control to the line following the line which caused the corresponding GOSUB.



### Example Statements

```
ERROR RETURN  
IF Dont_retry THEN ERROR RETURN
```

### Semantics

When this statement is executed, it causes program execution to resume at the line *following* the line that caused the most recent GOSUB (usually the line whose error initiated the most recent ON ERROR GOSUB branch). If you want to return to the line that *caused* the error, use RETURN.

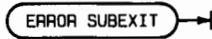
If the error occurred in an END, SUBEND, or FN END statement, then execution returns to that statement since there is no following executable line. If the statement is used to return from a GOSUB then it will behave as a RETURN.

## ERROR SUBEXIT

---

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

This statement returns program control to the line following the line which invoked the subprogram.



### Example Statements

```
ERROR SUBEXIT  
IF Dont_retry THEN ERROR SUBEXIT
```

### Semantics

When this statement is executed, it causes program execution to resume at the line *following* the line that caused the subprogram to be called (usually the line whose error initiated the most recent ON ERROR CALL branch). If you want to return to the line that *caused* the error, use SUBEXIT.

If the error occurred in an END, SUBEND, or FN END statement, then execution returns to that statement since there is no following executable line. If this statement is used from a CALL then it will behave as a SUBEXIT.

## EXIT IF

---

See the LOOP statement.

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.

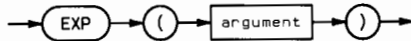
The EXOR function is summarized in this table.

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. With this system, Napierian  $e \approx 2.718\ 281\ 828\ 459\ 05$ .



Item	Description/Default	Range Restrictions
argument	numeric expression	-708.396 418 532 264 thru +709.782 712 893 383 8 for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

## Examples Statements

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

## Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

To compute the EXP of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula used to compute the EXP of a COMPLEX argument is:

```
CMPLX(EXP(Real_part)*COS(Imag_part),EXP(Real_part)*SIN(Imag_part))
```

where `Real_part` is the real part of the COMPLEX argument and `Imag_part` is the imaginary part of the COMPLEX argument. Some values of a COMPLEX argument may cause errors in this computation. For example,

```
EXP(CMPLX(710,0))
```

will cause error 22 due to the `EXP(Real_part)` computation.

Note that any COMPLEX function whose definition includes a sine or cosine function will be evaluated in the radian mode regardless of the current angle mode (i.e. `RAD` or `DEG`).

---

## EXPANDED

See the DUMP DEVICE IS statement.

---

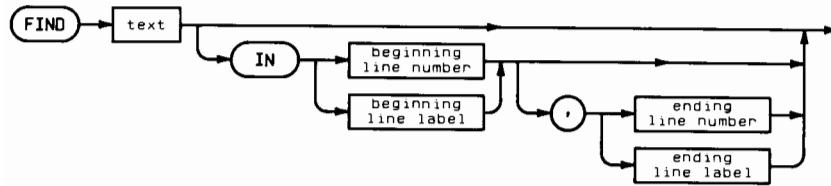
## FILL

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

# FIND

Option Required	Edit and 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	Range
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**.

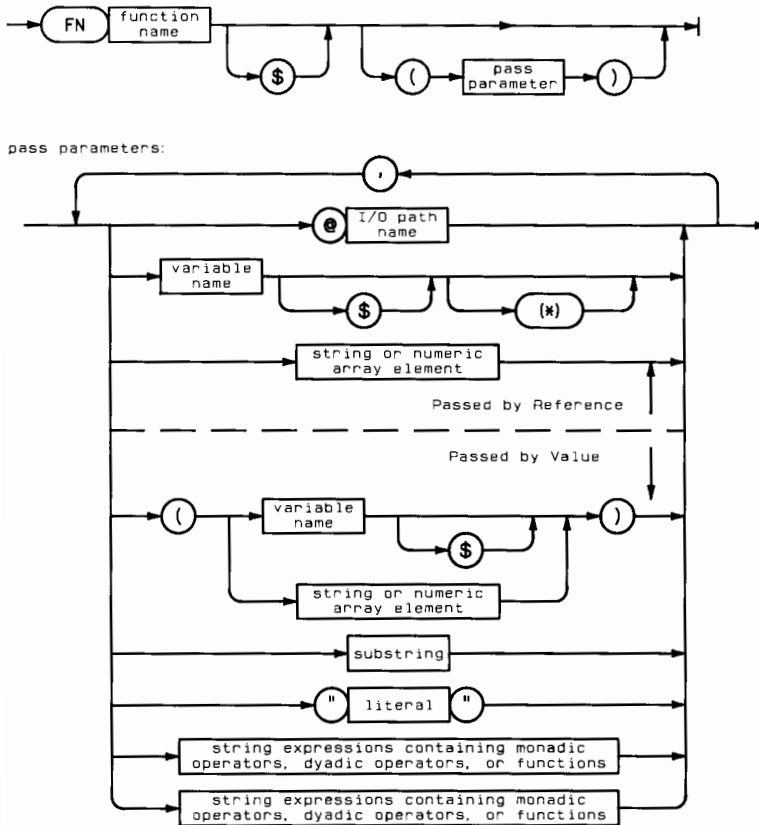
While in the FIND mode, keyboard execution is only possible with the **EXECUTE** key. Using **ENTER** or **Return** causes an error.



# 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.



Item	Description	Range
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=FNPrond(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, INTEGER, or COMPLEX) 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**

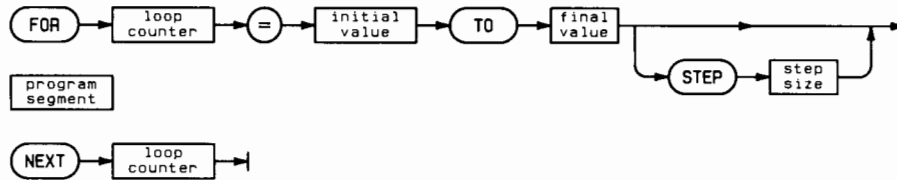
---

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	Range
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.

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,  $X = \text{INT}(X) + \text{FRACT}(X)$ .



## Example Statements

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



## Semantics

This function **does not** allow COMPLEX arguments.

## 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

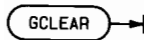
---

See the LEXICAL ORDER IS statement.



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 over-written by graphics to reappear.

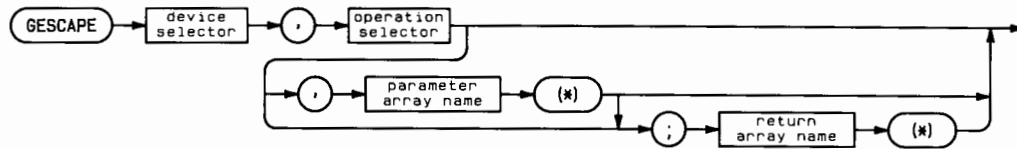
---

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	Range
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. Options 6 and 20 use the parameter array.

### **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. If this operation selector is used on a system with a monochrome display (or an HP 98627A), error 733 will be reported.

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 of 0 to 1 and are multiples of 1/15 (one fifteenth) for the Model 236C and multiples of 1/255 (one two-hundred fifty fifth) for other color displays. 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.

## 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 display-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 63 for 6-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 link.

For the three GESCAPE selectors above—20, 21, and 22—the parameter or return 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.)

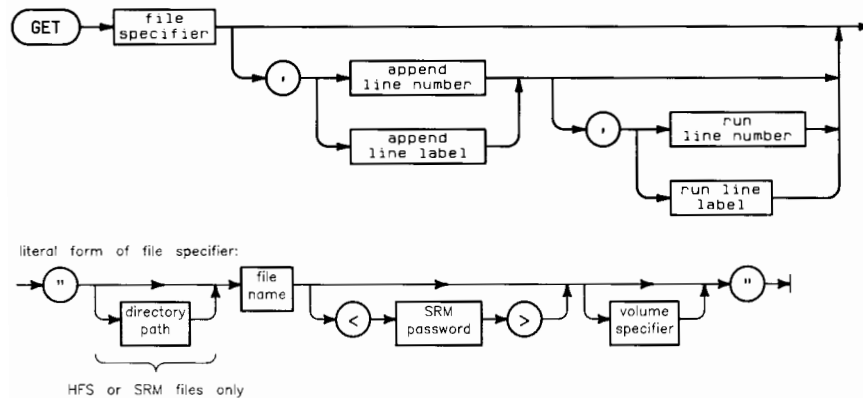
Unlike other GESCAPES, selectors 20 through 22 do not require the device at the specified select code to be currently active. Indeed, to be effective, GESCAPE 2,20, which sets hard clip limits, must be done before doing the GRAPHICS INPUT IS KBD, "TABLET" statement. Operations 20 and 21 will give “DEVICE 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 or Parameter Array
1	A(0): Number of entries in the color map
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 Graphics Device
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
5	Set alternate drawing mode
	} All Color CRTs
6	A(0): Current graphics write-enable mask value A(1): Current graphics display-enable mask value
	} All CRTs
7	A(0): Graphics write-enable mask value to be set A(1): Graphics display-enable mask value to be set
	} Series 300 Multi-Plane Devices
20	A(0): X maximum hard clip value to be set A(1): Y maximum hard clip value to be set
21	A(0): Current X maximum hard clip value A(1): Current Y maximum hard clip value
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

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

This statement reads the specified ASCII or HP-UX file and attempts to store the strings into memory as program lines.



Item	Description	Range
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
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	literal	(see MASS STORAGE IS)

## Example Statements

```
GET "George"  
GET Next_prog$,180,10  
GET "FileName:REMOTE"  
GET "/Dir1/Dir2/Dir3/File<SRM_READ_pass>"
```

## Semantics

The file must be either an ASCII or an HP-UX file (HP-UX files must contain text written in FORMAT ON representation).

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 reported. 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.

### **HFS Permissions**

In order to GET a file on an HFS volume, you need to have R (read) permission on the file, as well as X (search) permission on the immediately superior directory and all other superior directories.

### **GET with SRM Files**

In order to GET a file on an SRM volume, you need to have READ capability on the file and its immediately superior directory, as well as READ capabilities on all other superior directories. If this capability is not public or if a password protecting this capability is not given, an error is reported.

You may use GET with any ASCII or HP-UX 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.

# 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.

**GINIT** →

## 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.

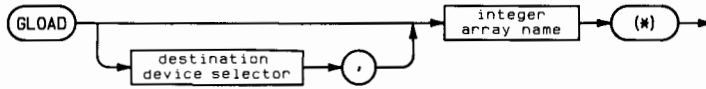
After a GINIT and before a PLOTTER IS statement is executed, the following statements select a default plotter.

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

# 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	Range
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 one bit deep. The Model 236 color display 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 HP 98543A and HP 98545A display boards have 4 planes, allowing 16 colors, and the HP 98700 has 4 or 8 planes, allowing 16 or 256 colors, respectively. The HP 98547A and HP 98549A display boards have 6 planes, allowing 64 colors. The HP 98550A display board has 8 planes, allowing 256 colors.

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
<b>1-plane systems:</b>			
216 (HP 9816) (monochromatic)	400	300	7500
220 (HP 9920) (monochromatic)			
(HP 98204A)	400	300	7500
(HP 98204B)	512	390	12 480
226 (HP 9826) (monochromatic)	400	300	7500



Model	Horizontal Size	Vertical Size	INTEGER Elements
236 (HP 9836) (monochromatic)	512	390	12 480
98546 (monochromatic)	512	390	12 480
237 (HP 9837) (bit-mapped, monochromatic)	1024	768	49 152
98542 (medium-resolution bit-mapped, monochromatic)	1024	400	25 600
98544 (high-resolution, bit-mapped, monochromatic)	1024	768	49 152
98548 (high-resolution, bit-mapped, monochromatic)	1280	1024	163 840
<b>3-plane systems:</b>			
98627A (external color)	512	512	49 152
<b>4-plane systems:</b>			
236 (HP 9836C) (color)	512	390	49 920
98543 (medium-resolution, bit-mapped, color)	1024	400	102 400
98545 (high-resolution, bit-mapped, color)	1024	768	196 608
<b>6-plane systems:</b>			
98547 (high-resolution, bit-mapped, color)	1024	768	393 216
98549 (high-resolution, bit-mapped, color)	1024	768	393 216

Model	Horizontal Size	Vertical Size	INTEGER Elements
<b>8-plane systems:</b>			
98700 (high-resolution, bit-mapped, color)	1024	768	393 216
98550A (high-resolution, bit-mapped, color)	1280	1024	655 360

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

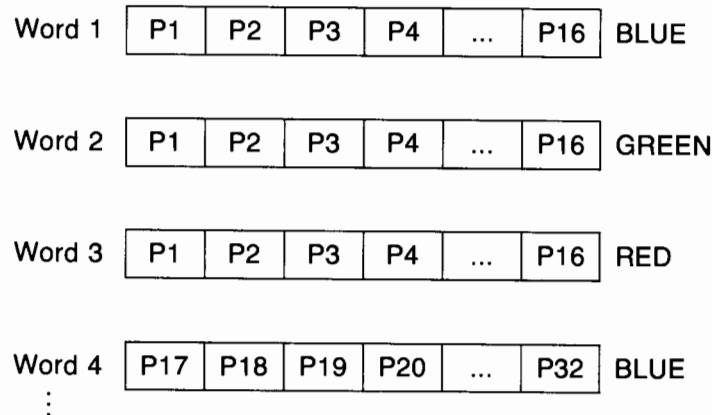
The pixel representation on a monochromatic display is stored sequentially in the array using GSTORE.

The pixel representation for color displays is 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.

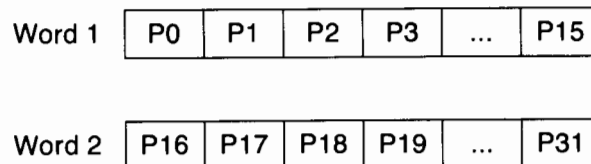


**Storage Format on Multi-Plane Bit-Mapped Displays**

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.

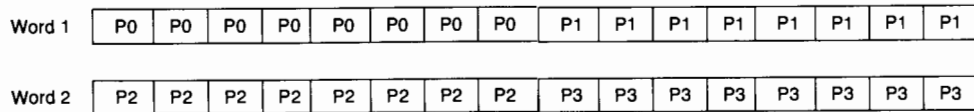
If the highest graphics plane currently enabled is 1 (or none), act like there is 1. The storage format is:



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 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 GLOAD the image. You can change the graphics write mask with GESCAPE.

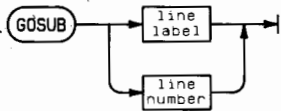
### Non-Square Pixel Displays

With non-square pixel displays, GLOAD 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 GLOADed will appear exactly the same when GSTOREd back into the frame buffer. Since alpha uses the non-square 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.

# 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.)



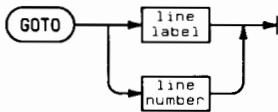
Item	Description	Range
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766

## Example Statements

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

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	Range
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

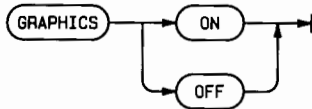
GOTO Loop\_start

IF Full THEN Exit (implied GOTO)

# 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 paragraphs.

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

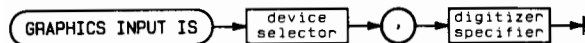
Mixing ALPHA/GRAPHICS ON/OFF with explicit definition of the display-enable mask may cause the `ALPHA` and/or `GRAPHICS` keys to have unexpected results. The reason for this is that explicit setting of the display mask is, in a manner of speaking, working “behind the back” of the operating system. Thus, you could turn off graphics by modifying the display-enable mask, and the internal variables which keep track of `ALPHA` and `GRAPHICS` key presses 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	Range
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.

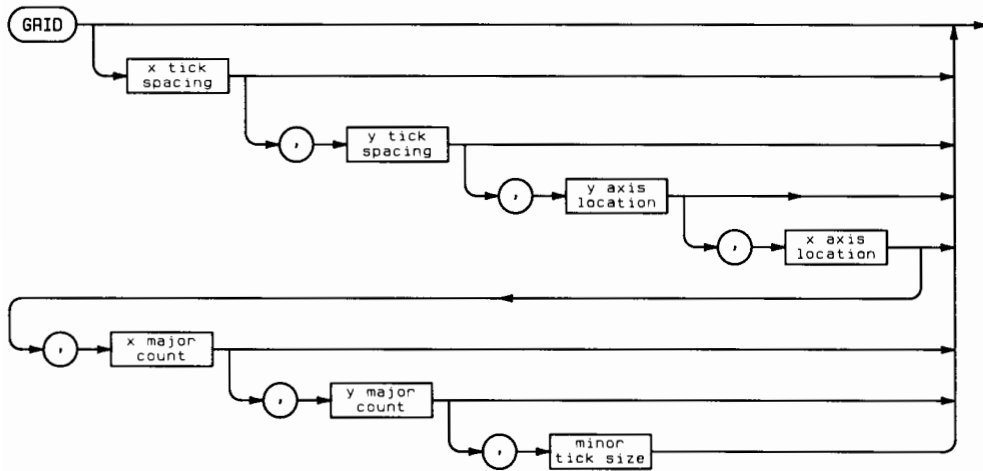
### **Absolute Locator Hard Clip Limits**

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 GESCAPE for 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	Range
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 be- tween 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 be- tween 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.

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

This statement is used to send HPGL commands to the current PLOTTER IS device.



Item	Description	Range
HPGL command string	string expression	device-dependent

**Example Statements**

```

IF Hpgl_device THEN GSEND "IP;"

GSEND String$
  
```

**Semantics**

This statement sends a string of characters to the current PLOTTER IS device, which may be a file or a plotter. The string is to contain Hewlett-Packard Graphics Language (HPGL) command(s). Note that BASIC does not check the syntax of these HPGL commands.

GSEND is most useful when the PLOTTER IS device is a file (it is not possible to OUTPUT an HPGL command to the file while it is the PLOTTER IS device).

An error is reported if the current PLOTTER IS device is not an HPGL device or a file.

After GSEND sends the specified string, it will send a carriage return/line feed (as an EOL sequence). If your device does not recognize a carriage return/line feed as a terminator for an HPGL command, you must include the correct terminating sequence (normally a semicolon) as part of the HPGL command string you are sending.

Note that you **cannot** split HPGL commands over more than one GSEND statement, because of the carriage return/line feed sequence sent after each GSEND statement. The following example **will not work** because it splits the HPGL command over more than one GSEND statement. (Also, the command is not properly terminated.)

```

GSEND "P"
GSEND "F"
  
```

The proper way to send this HPGL command is:

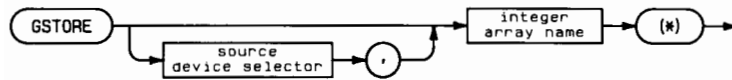
```

GSEND "PF;"
  
```

# GSTORE

Option Required	GRAPH
Keyboard Executable	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	Range
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 one bit deep. The Model 236 color display 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 HP 98543A and HP 98545A display boards have 4 planes, allowing 16 colors, and the HP 98700 has 4 or 8 planes, allowing 16 or 256 colors, respectively. The HP 98547A and HP 98549A display boards have 6 planes, allowing 64 colors. The HP 98550A display board has 8 planes, allowing 256 colors.

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.

<b>Model</b>	<b>Horizontal Size</b>	<b>Vertical Size</b>	<b>INTEGER Elements</b>
<b>1-plane systems:</b>			
216 (HP 9816) (monochromatic)	400	300	7500
220 (HP 9920) (monochromatic)			
(HP 98204A)	400	300	7500
(HP 98204B)	512	390	12 480
226 (HP 9826) (monochromatic)	400	300	7500
236 (HP 9836) (monochromatic)	512	390	12 480
98546 (monochromatic)	512	390	12 480
237 (HP 9837) (bit-mapped, monochromatic)	1024	768	49 152
98542 (medium-resolution bit-mapped, monochromatic)	1024	400	25 600
98544 (high-resolution, bit-mapped, monochromatic)	1024	768	49 152
98548 (high-resolution, bit-mapped, monochromatic)	1280	1024	163 840
<b>3-plane systems:</b>			
98627A (external color)	512	512	49 152

Model	Horizontal Size	Vertical Size	INTEGER Elements
<b>4-plane systems:</b>			
236 (HP 9836C) (color)	512	390	49 920
98543 (medium-resolution, bit-mapped, color)	1024	400	102 400
98545 (high-resolution, bit-mapped, color)	1024	768	196 608
<b>6-plane systems:</b>			
98547 (high-resolution, bit-mapped, color)	1024	768	393 216
98549 (high-resolution, bit-mapped, color)	1024	768	393 216
<b>8-plane systems:</b>			
98700 (high-resolution, bit-mapped, color)	1024	768	393 216
98550A (high-resolution, bit-mapped, color)	1280	1024	655 360



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

### Storage Format

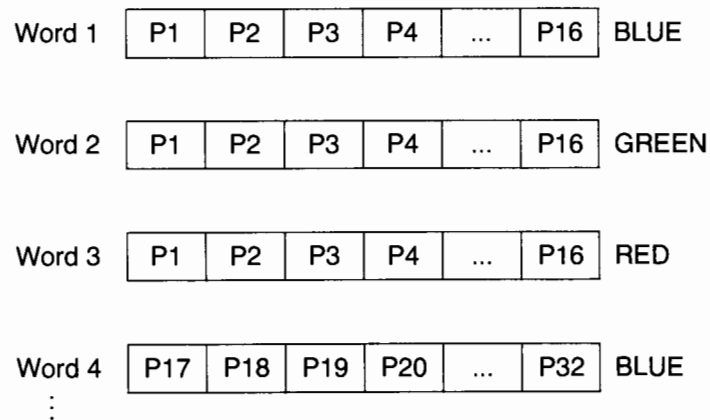
The pixel representation on a monochromatic display is stored sequentially in the array using GSTORE.

The pixel representation for color displays is 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.



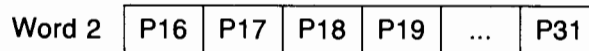
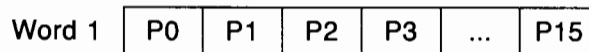


## Multi-Plane Bit-Mapped Displays

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.

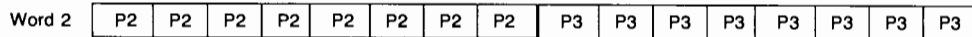
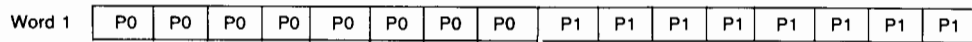
In the following paragraphs, reference is made to the “highest graphics plane.” The “highest graphics plane” is that plane in the frame buffer whose corresponding bit in the graphics write-enable mask has the highest number. For example, the highest graphics plane with a write mask of binary 1000 is 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 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.

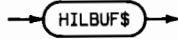
You can conserve space if you are using fewer than the maximum number of planes. For example, on a 98700 with eight planes, if pens 0 through 15 *only* are being used, the graphics write mask could be set to 15 (binary 00001111) rather than the default of 255 (binary 11111111). In this way, only half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

### **Non-Square Pixel Displays**

With non-square 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 non-square 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.

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

This function receives data sent by an HP-HIL device (e.g Describe Records, Poll Records, etc.).



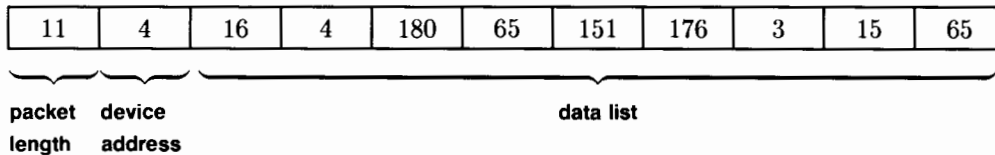
## Example Statement

```
HILBUF$
Temp_buf$=HILBUF$
IF Read_buf THEN Buffer$=HILBUF$
```

## Semantics

This function receives data from HP-HIL devices which have had polling enabled by the ON HIL EXT statement or which have been sent a command by the HIL SEND statement. This data takes the form of 8-bit numbers (bytes) packed into a string. When HILBUF\$ is read, the internal buffer where it accumulates this data is cleared, ready to receive more data.

The format of the string returned by the HILBUF\$ function is as follows: lost packet count, followed by zero or more data packets. The lost packet count will normally be zero (the null character). If the internal buffer overflows (because it is not read), the lost packet count is the total number of packets lost (to a maximum count of 255 packets). Only whole packets are put into the buffer. The format of a packet is: packet length, device address, data list. For example, sending an HIL SEND 4;RSC statement to an ID Module would create a packet of data similar to the following:



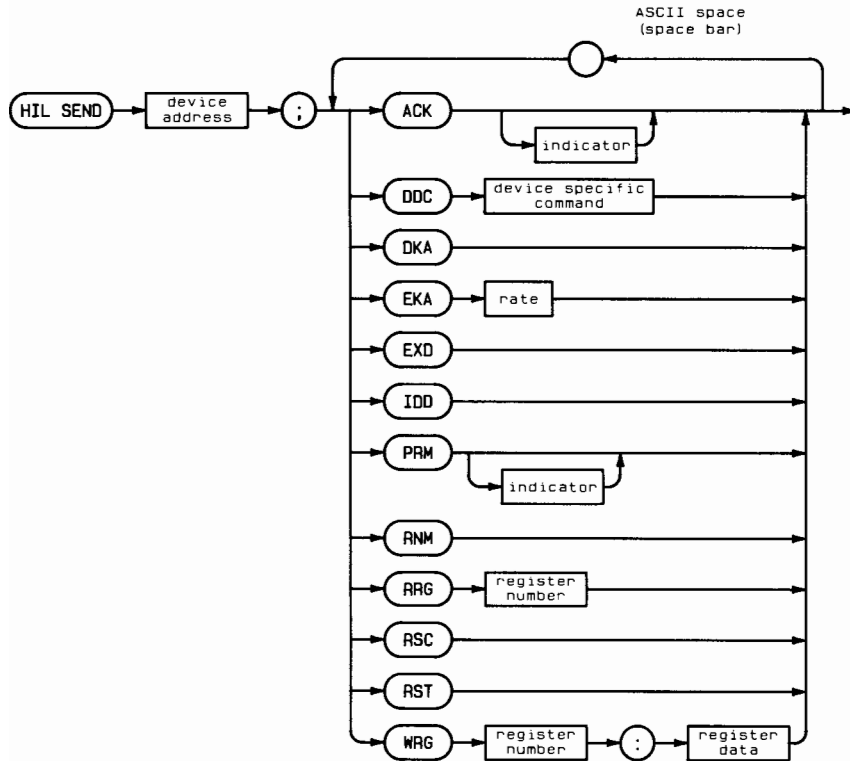
where 11 is the packet length and 4 is the device address. The remaining characters make up the data list (which in this example is a product/exchange number and serial number). The first character of the packet is the *packet length*. The *packet length* tells you how many string characters are left in the packet (including this character). *Packet lengths* range from 3 to 19 characters. The second character in the string is the *device address*. This tells you the position of the device within the HP-HIL link. There can only be a total of 7 addresses in the HP-HIL link. The remaining characters in the packet make up the *data list*. This list contains information which is dependent on the HP-HIL device polled (ON HIL EXT) or on the HP-HIL device and the command sent (HIL SEND). For more information on packets read the chapter "HP-HIL Interface" found in the *BASIC Interfacing Techniques* manual.

If the HIL SEND statement results in data being returned from the device, the data is put into HILBUF\$ even if HP-HIL interrupts are not enabled (i.e. ON HIL EXT is not currently active). Note that no interrupt is generated, even if HP-HIL interrupts are enabled (i.e. ON HIL EXT is currently active), for data placed in HILBUF\$ as a result of HIL SEND. However, care should be taken in this case, since executing ON HIL EXT clears HILBUF\$.

# HIL SEND

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

This statement allows a selected subset of the HP-HIL Command Set to be transmitted to specific devices in the HP-HIL link.



Item	Description/Default	Range Restrictions
device address	numeric expression representing the HP-HIL device's position in the HP-HIL link	1 thru 7
rate	numeric expression indicating a keyswitch auto-repeat rate of 20 or 40 milliseconds	1 or 2
indicator	numeric expression representing which of several prompts/acknowledges on the device to use	1 thru 7
device specific command	numeric expression whose meaning is device dependent	128 thru 239
register number	numeric expression	0 thru 127 (RRG); 0 thru 255 (WRG)
register data	numeric expression	0 thru 255

### Example Statement

```
HIL SEND Dev_address;DKA PRM Led
HIL SEND 3;IDD
HIL SEND 7;ACK 6
```

### Semantics

HP-HIL commands must be sent to a specific HP-HIL device, they may not be sent to several devices at once.

The IDD (Identify and Describe) command can be sent to all HP-HIL devices. For all other commands, HP-HIL devices which can use the HIL SEND statement are those whose poll records are not being processed for another purpose by the BASIC system. These devices are grouped into three categories:

- Absolute positioning devices which are not the current GRAPHICS INPUT device. Examples of these devices are as follows: Touchscreen (HP 35723A), A-size Digitizer (HP 46087A), B-size Digitizer (HP 46088A).
- HP-HIL devices with Device ID's less than hexadecimal 60. Examples of these devices are as follows: Bar-code Reader (HP 92916A), ID Module (HP 46084A), Function Box (HP 46086A), Vectra Keyboard (HP 46030A).
- True keyboards which **are not** relative pointing devices, such as the HP 46020A and HP 46021A. (Poll records from these devices are processed by the Keyboard controller, rather than by BASIC.)

The main HP-HIL devices which **cannot** use this function are:

- Relative pointing devices, such as the HP Mouse (HP 46060A) and Control Dial Box (HP 46085A). Since the HP 98203C keyboard has a knob on it, it is considered a relative pointing device and cannot be used with the HP-HIL Command Set.
- Current GRAPHICS INPUT devices.

If the HIL SEND statement results in data being returned from the device, the data is put into HILBUF\$ even if HP-HIL interrupts are not enabled (i.e. ON HIL EXT is not currently active). Note that no interrupt is generated, even if HP-HIL interrupts are enabled, for data placed in HILBUF\$ as a result of HIL SEND. However, care should be taken in this case, since executing ON HIL EXT clears HILBUF\$.

The system will report an error if an attempt is made to send an HP-HIL command to an HP-HIL device at an address which was not present at the last SCRATCH A or power-up, even if a device is now present at that address. The system will not report an error if a command is sent to an address which had a device present at power-up or SCRATCH A but is now empty.

The sections which follow cover the HP-HIL commands supported by BASIC. For a detailed description of these commands, read the “HP-HIL Command Reference” located in the “HP-HIL Appendix” of the *BASIC Interfacing Techniques* manual.

#### **IDD**

Identify and Describe is used by the system to determine the type of HP-HIL devices in the HP-HIL link, as well as some general characteristics of these devices.

Sending an IDD to a relative pointing device, or to an absolute pointing device which is currently the GRAPHICS INPUT device, will result in an IDD record being reconstructed from the system’s internal configuration record. This pseudo-IDD record will be the actual IDD record stored by the system at power-up or SCRATCH A time, but no HP-HIL bus access is made (i.e. if the device has been removed, it will still show up here).

#### **RRG**

Read Register provides a means for interaction with more complex devices via HP-HIL, allowing for data transfers not generally supported by the HP-HIL devices. Device support for this command is indicated in the Extended Describe Record.

The numeric value listed after this HP-HIL command is the number of the register that is to be accessed. The range of valid register numbers is 0 to 127.

**WRG**

Write Register provides a means of setting the contents of individual registers in HP-HIL devices supporting this feature. Device support for this command is indicated in the Extended Describe Record.

There are two types of Write Register Records: Write Register Type 1 and Write Register Type 2. BASIC looks at both of these Types as functionally the same (i.e. they both write a single byte to a single register).

The numeric value listed after this HP-HIL command is the number of the register that is to be accessed. The range of valid register numbers is 0 to 255. Range 0 to 127 selects Write Register Type 1 and range 128 to 255 selects Write Register Type 2. In either case, only one data item may be transmitted per command (i.e. this implementation limits the Type 2 data list to one item).

**RNM**

Report Name is used to request a string of up to 15 characters (8-bit ASCII) which aid in describing the device to the user. Devices indicate support of the Report Name command in the Extended Describe Record.

**RST**

Report Status is used to extract device-specific status information from devices configured on the HP-HIL link. Devices indicate support of the Report Status command in the Extended Describe Record.

**EXD**

This command provides additional information concerning more advanced device features which may not be required for basic operation. Support of the Extended Describe command is indicated in the Describe Record Header.

**RSC**

The Report Security Code command is used to extract a unique product/exchange number and serial number from the HP-HIL device. Support of the command is indicated in the Describe Record Header.

Information returned when executing this command can also be obtained using the SYSTEM\$(“SERIAL NUMBER”) function. It should be noted that this function will only return the product/exchange number and serial number for the last HP 46084A ID Module in the HP-HIL link. If there are other devices in the HP-HIL link with security code information, they are ignored by the SYSTEM\$(“SERIAL NUMBER”) function. This is not the case with the RSC command when it is executed with the HIL SEND statement, as it will allow you to select the device you want to report a security code.



## **DKA**

This command is used to disable the “repeating keys” feature for the addressed HP-HIL device, reducing returned data to one report per keyswitch transition. Support of this command is not indicated in the Describe Record or Extended Describe Record. Examples of devices which support it are the: Function Box, Vectra Keyboard, ITF Keyboard. The default state for HP-HIL devices supporting this command is AutoRepeat disabled.

---

### **Note**

The auto-repeat for DKA, EKA 1, and EKA 2 is different and independent of the keyboard auto-repeat which is controlled by keyboard CONTROL registers 3 and 4. The repeated arrow keys return a code which is not recognized by the keyboard driver; hence they have no effect.

---

## **EKA 1**

EKA 1 is used to enable the “repeating key” feature in the addressed device (if the feature is supported). Support of this command is not indicated in the Describe Record or Extended Describe Record. Examples of devices which support it are the: Function Box, Vectra Keyboard, ITF Keyboard. This command will cause the HP-HIL device’s keys to repeat about every 40 milliseconds. Modifier keys (**Shift**, **CTRL**, **Extend char**, etc.) will not repeat. The cursor keys (**←**, **→**, **▲** and **▼**) on an ITF Keyboard will send repeated 02 codes after the initial Keycode.

## **EKA 2**

EKA 2 is used to enable the “repeating key” feature in the addressed device (if the feature is supported). Support of this command is not indicated in the Describe Record or Extended Describe Record. Examples of devices which support it are the: Function Box, Vectra Keyboard, ITF Keyboard. This command will cause the HP-HIL device’s keys to repeat about every 40 milliseconds. Modifier keys (**Shift**, **CTRL**, **Extend char**, etc.) will not repeat. The Cursor Keys (**←**, **→**, **▲** and **▼**) on an ITF Keyboard will send repeated 02 codes at a faster rate than EKA 1.

**PRM 1..7**

These Prompt commands are used to provide an audible or visual stimulus to the user, perhaps indicating that the System is ready for a particular type of input. Usually Prompts 1 through 7 are paired with Acknowledge 1 through Acknowledge 7.

The Prompts supported by a device are indicated in the Describe Record.

**PRM**

Prompt is intended to be a general-purpose stimulus to the user. This command is usually paired with Acknowledge. An HP-HIL device indicates support of Prompt in the Describe Record.

**ACK 1..7**

These Acknowledge commands are used to provide an audible or visual stimulus to the user, perhaps indicating that the System is ready for a particular process to be performed. Usually Acknowledges 1 through 7 are paired with Prompt 1 through Prompt 7.

The Acknowledges supported by a device are indicated in the Describe Record.

**ACK**

Acknowledge is intended to be a general-purpose stimulus to the user. This command is usually paired with Prompt. An HP-HIL device indicates support of Acknowledge in the Describe Record.

**DDC 128..239**

This is a range of 112 commands which have been reserved for use as “device-specific” commands. These commands are intended for use by devices with special requirements which the remainder of the HP-HIL protocol does not readily support.

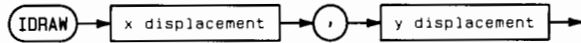
**IDN**

---

See the MAT statement.

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	Range
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.

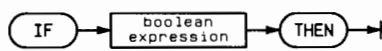
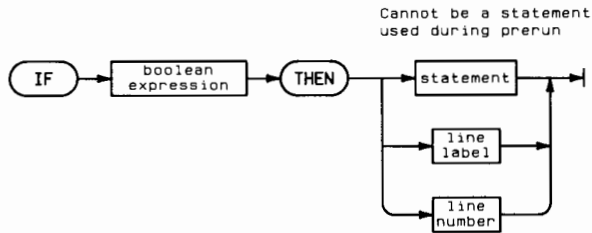
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.

Note 4: RPLOT and IPLOT are affected by PDIR.

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



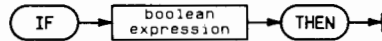
This statement provides conditional branching.



program  
segment

END IF

---



program  
segment

ELSE

program  
segment

END IF

Item	Description	Range
boolean expression	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	REM
CASE ELSE	END IF	IMAGE	REPEAT
COM	END LOOP	INTEGER	SELECT
COMPLEX	END SELECT	LOOP	SUB
DATA	END WHILE	NEXT	SUBEND
DEF FN	EXIT IF	OPTION BASE	UNTIL
DIM	FNEND	REAL	WHILE
ELSE	FOR		

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.

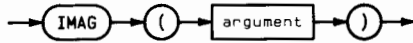
Branching into an IF...THEN construct (such as with a GOTO) results in a branch to the program line following the END IF when the ELSE statement is executed.

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).

# IMAG

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

This function returns the imaginary part of a COMPLEX number.



Item	Description/Default	Range Restrictions
argument	numeric expression	any valid INTEGER, REAL, or COMPLEX value

## Example Statements

```
X=IMAG(Complex_expr)
Y=IMAG(Real_expr)
Z=IMAG(Integer_expr)
Result=IMAG(CMPLX(2.1,-8))
```

## Semantics

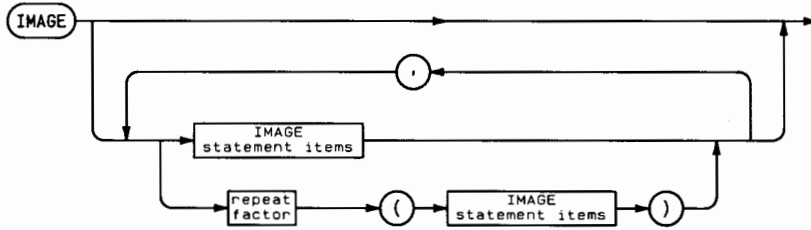
If the argument is not a COMPLEX value, the result is 0.



# 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.



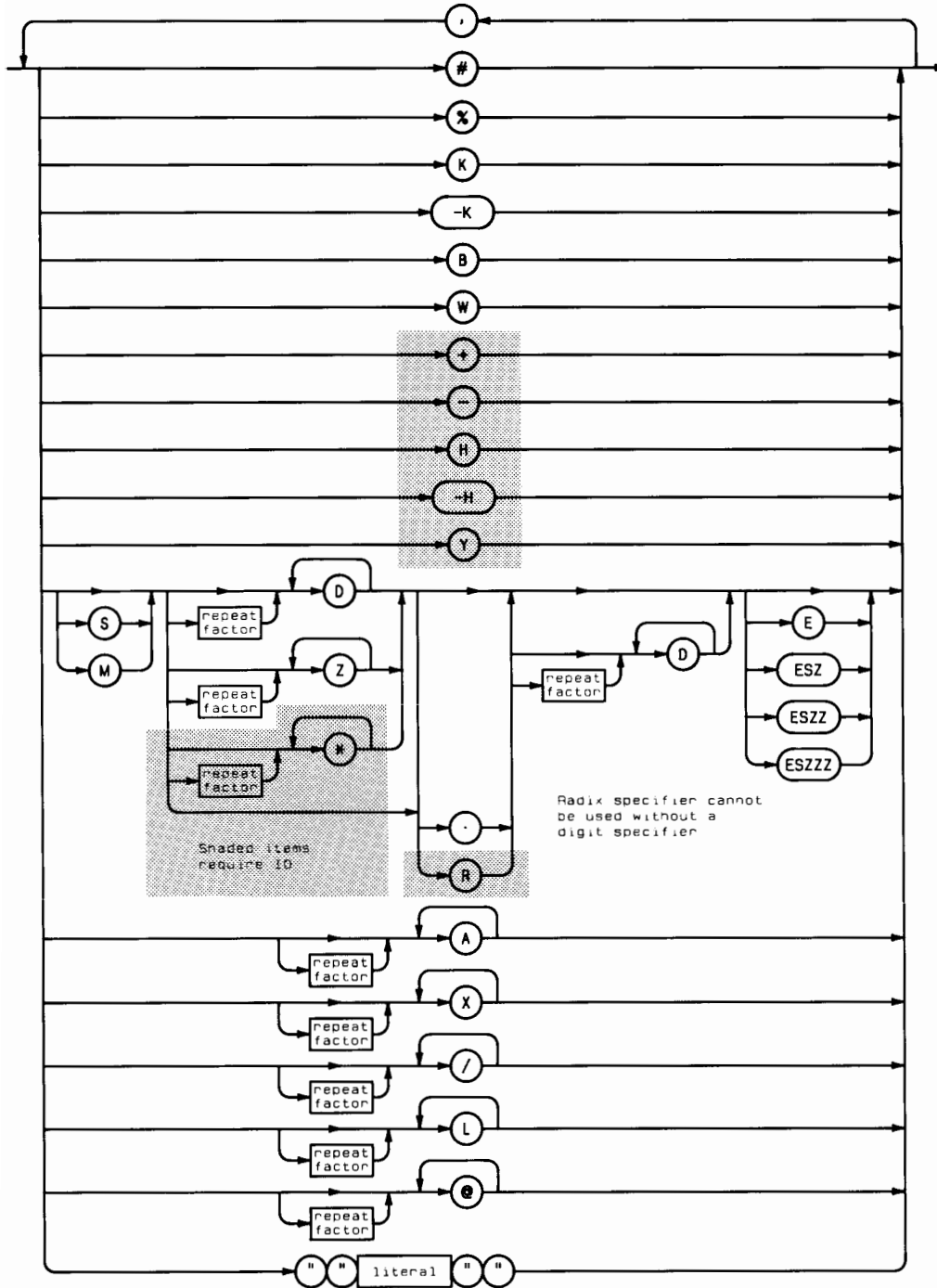
Item	Description	Range
IMAGE	literal	(see drawing)
statement items		
repeat factor	integer constant	1 thru 32 767
literal	string composed of characters from the keyboard, including those generated using the ANY CHAR key.	quote mark not allowed

## Example Statements

```
IMAGE 4Z,DD,3X,K,/  

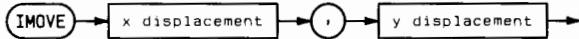
IMAGE "Result = ",SDDDE,3(XX,ZZ)  

IMAGE #,B
```



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	Range
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.

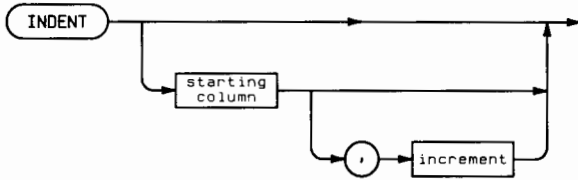
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.

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	Range
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 <sup>1</sup>	DEF FN



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.

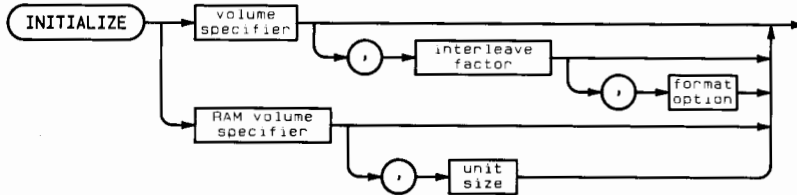
---

<sup>1</sup> This is only true for IF..THEN statements where the THEN is followed by an end-of-line or an exclamation point.

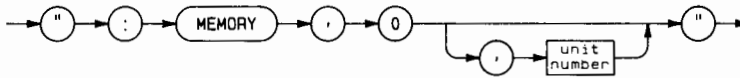
# INITIALIZE

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

This statement prepares ("formats") mass storage media for use by BASIC, and places a LIF (Logical Interchange Format) directory on the media. (To format an HFS volume, use the DISC\_UTIL program.) When INITIALIZE is executed, **any existing files on the media are destroyed.**



literal form of RAM volume specifier:



Item	Description	Range
volume specifier	string expression	(see MASS STORAGE IS)
interleave factor	numeric expression, rounded to an integer; Default = device dependent (see table)	0 thru 15
format option	numeric expression Default = 0	device dependent
RAM volume specifier	string expression	(see drawing)
RAM unit size	numeric expression, rounded to an integer; specifies number of 256-byte sectors; Default = 1056 (size of a 5 <sup>1</sup> / <sub>4</sub> -inch or <i>single</i> - sided 3 <sup>1</sup> / <sub>2</sub> -inch disc)	4 thru 32 767 memory-dependent

## Example Statements

```
INITIALIZE ":INTERNAL"  
INITIALIZE Disc$,2  
INITIALIZE ":",700",0,4  
INITIALIZE ":MEMORY,0",Sectors
```

## Semantics

Any media used by the computer must be initialized before its **first** use. Initialization creates a new LIF 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 subsequently).

## Interleave Factor

The interleave factor establishes the distance (in physical sectors) between consecutively numbered sectors. 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
HP 9121	2
HP 913X (floppy)	4
HP 913X (hard)	9
HP 9885	1
HP 9895	3
HP 8290X	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.

---

### **Format Option**

Some mass storage devices allow you to select the sector or volume size with which the disc is initialized. Omitting this parameter or specifying 0 initializes the disc to the default sizes. Refer to the disc drive manual for 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 a value of 4 (256-byte sectors, and 270K bytes total volume size).

### **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.

### **INITIALIZE and HFS Volumes**

Since INITIALIZE creates a LIF directory, it **cannot** alone be used to format an HFS disc; it will still, however, scan the volume for bad sectors. To format an HFS volume, use the System Disc Utility (DISC\_UTIL, which calls the "Mkhfs" compiled subprogram to place an HFS-format directory on the disc volume). See the "Preparing Hard Discs" chapter of *Installing, Using, and Maintaining the BASIC System* for instructions on using this utility.

### **INITIALIZE and SRM Volumes**

Since INITIALIZE creates a LIF directory, it **cannot** be used with SRM (which uses the Structured Directory Format, SDF). An error will be reported if you attempt to initialize a REMOTE volume from a workstation.

### **Recovering MEMORY Volume Space**

BASIC RAM disc memory can be reclaimed if no binaries have been loaded after initializing the memory volume. To recover this memory, you would execute a line similar to the following:

```
INITIALIZE ":,0, unit number ",0
```

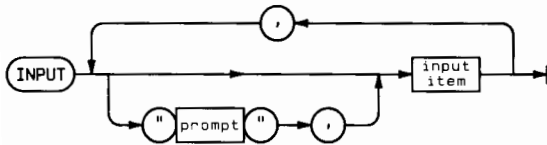
Initializing the volume to 0 sectors removes it from memory.

Memory volumes are allocated in a mark and release stack. What this means is, you get the memory back only when other subsequently created memory volumes have been reclaimed. You can re-initialize a removed memory volume in its original space provided the newly allocated space is no larger than the original space that was allocated. Otherwise, new space will be allocated for it.

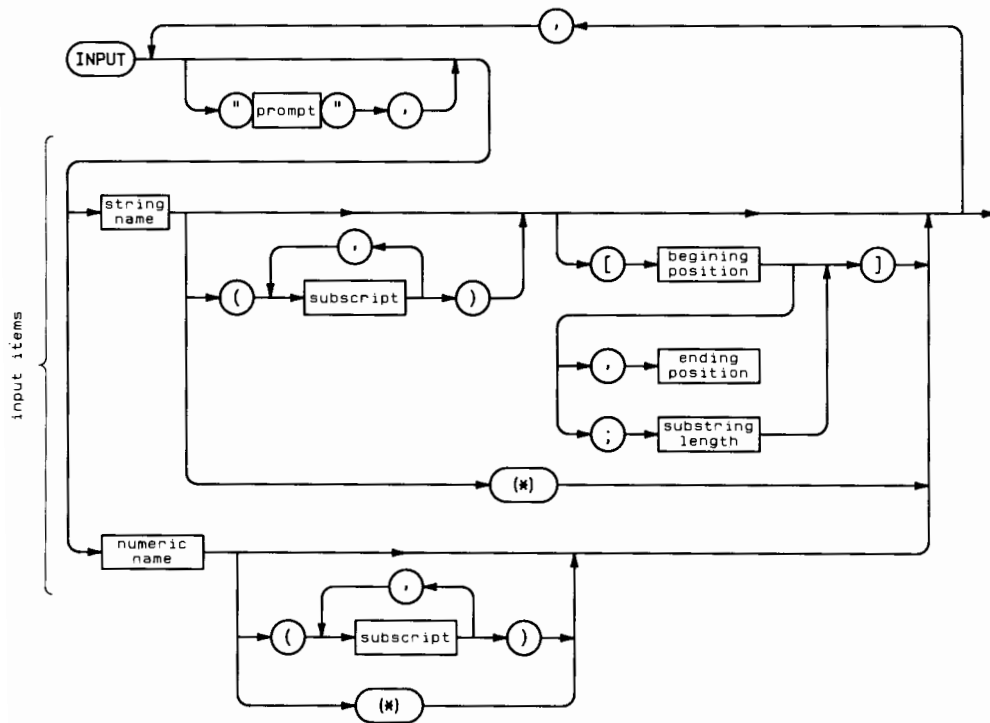


Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF... THEN	Yes

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	−32 768 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)
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 quotation marks. Placing quotes around an input string allows any character(s) to be used as input. If " is intended to be a character in a quoted string, use ". Note that when you are prompted to input a COMPLEX value, you must input two REAL values (one representing the real part and another representing the imaginary part) separated by a comma or a `Return` or `ENTER`.

Multiple values can be entered individually or separated by commas. Press the `Return`, `CONTINUE`, `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 order (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 46020 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.

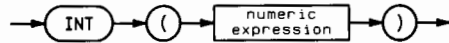
ON KBD, ON KEY and ON KNOB events are deactivated during an INPUT statement. Errors do not cause an ON ERROR branch. If an input response results in an error, re-entry begins with the variable which would have received the erroneous response.

# 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
```

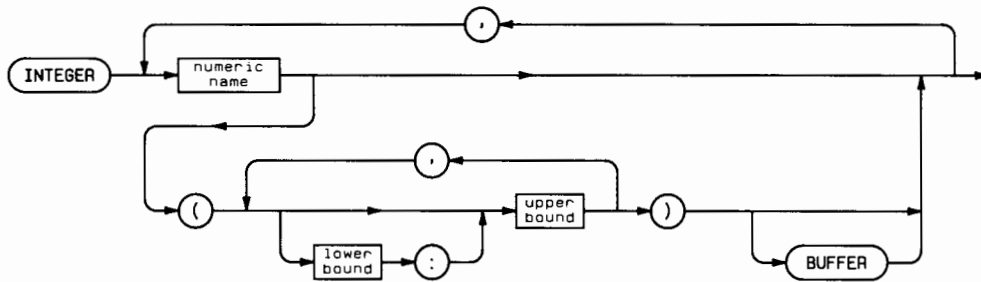
## Example Statements

COMPLEX arguments **are not** allowed with this function.

# 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	Range
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. No single dimension can have more than 32 767 total elements.

The total number of INTEGER elements is limited by the fact that the maximum memory usage for *all* variables—COMPLEX, INTEGER, REAL, and string—within any context is  $2^{24}-1$ , or 16 777 215, bytes (or limited by the amount of available memory, whichever is less).

### **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.

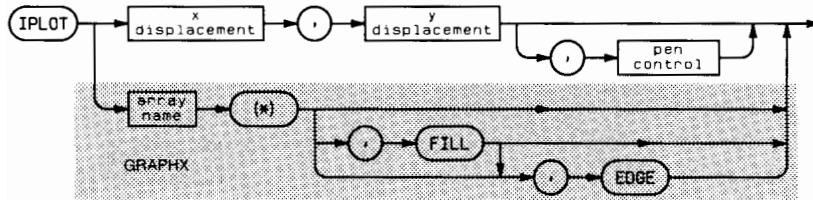
## **INV**

---

See the MAT statement.

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	Range
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 range 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 following list.

### IPLOT Array Parameter Effects

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 } Value
blue value	ignored	15	
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**

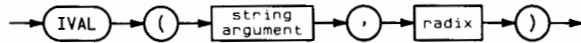
Operation selector 8 causes the IPLOT statement to be terminated. The IPLOT statement will successfully terminate if the actual end of the array has been reached, so the use of operation selector 8 is optional.

### **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 than 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.

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	Range
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	Sign 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

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	Range
"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	-32 768 thru +32 768	6 characters
16	hexadecimal	0000 thru FFFF	4 characters



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.

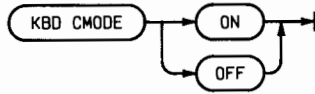
The buffer can hold 256 characters. Further keystrokes are not saved and produce beeps. An overflow flag is set after the buffer is full. This flag can be checked by reading keyboard status register 5 and is cleared by reading the status register, SCRATCH A, and a **RESET** operation.

The buffer is cleared by KBD\$, OFF KBD, SCRATCH, SCRATCH A, INPUT, LINPUT, ENTER 2, and a **RESET** operation.

# KBD CMODE

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

This statement enables/disables the HP 98203A/B/C keyboard compatibility mode on an ITF keyboard.



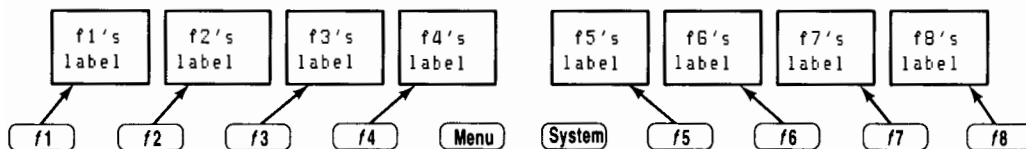
## Example Statements

```
KBD CMODE ON  
KBD CMODE OFF  
IF Change_mode THEN KBD CMODE ON
```

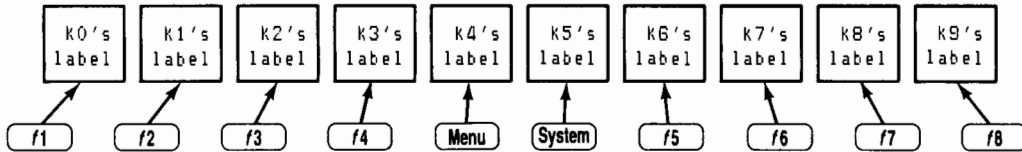
## Semantics

Executing the KBD CMODE ON statement re-defines the softkeys of the ITF keyboard so they are compatible with the HP 98203A/B/C keyboard softkeys. This means that the eight function keys located at the top of the ITF keyboard, as well as the **Menu** and **System** keys, emulate function keys **k0** thru **k9** of the HP 98203A/B/C Keyboard and their BASIC system definitions.

The following are the ITF softkey definitions before executing KBD CMODE ON:



The following are the ITF softkey definitions after executing KBD CMODE ON:



Executing the KBD CMODE OFF statement returns you back to the softkey key definitions of the ITF Keyboard.

The KBD CMODE statement does not affect KEY LABELS ON or OFF. What this means is the definitions of the softkeys change, but the softkey labels at the bottom of the display will not be turned on or off.

While in the Keyboard Compatibility mode, the System Function keys are accessed by using the `Extend char` key with the softkeys `f1` through `f8`. Note that using `Extend char` along with the `Menu` key turns the keylabels on and off in this mode. `Extend char` when pressed with the `User` or `System` keys exits the Keyboard Compatibility mode. Also, in the Keyboard Compatibility mode, `Shift` can be used with the eight softkeys and the `Menu` and `System` keys (as on an HP 98203A/B/C) to access keys `SHIFT-k0` through `SHIFT-k9`.

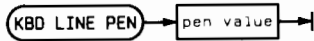
Note that the functionality of this statement can be achieved through KBD CONTROL register 15.

For further information on HP 98203A/B/C keyboard compatibility mode, read the chapter called "Porting to Series 300" in the *BASIC Programming Techniques* manual.

## KBD LINE PEN

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

This statement sets the pen color to be used for the keyboard line and other associated areas of the CRT (see "Semantics" below).



Item	Description/Default	Range Restrictions
pen value	numeric expression	see semantics

### Example Statements

```
KBD LINE PEN Pen_value
KBD LINE PEN 143
IF Color_blue THEN KBD LINE PEN 141
```

### Semantics

This statement has no effect on monochrome displays.

Pen color areas of the CRT which are associated with the keyboard line are:

- the run indicator
- all of the edit screen except the key labels
- the following annunciators:
  - softkey menu (e.g. **System**, **User 1**, **User 2**, and **User 3**)
  - CAPS indicator
  - system-activity indicator (e.g. **Idle**, **Running**, etc.)
- the system message area

The set of alpha colors for the Model 236C is given in the table below:

Value	Result
< 16	The number is evaluated MOD 8 and resulting values produce the following: 0 — black 1 — white 2 — red 3 — yellow 4 — green 5 — cyan 6 — blue 7 — magenta
16 to 135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144 to 255	Ignored

For displays with bit-mapped alpha, `KBD LINE PEN` specifies the graphics pen to be used for subsequent alpha output. The range of values allowed with this statement are 0 through 255; these values are treated as “*value* MOD ( $2^n$ )” where *n* is the number of display planes.

`KBD LINE PEN n` and `CONTROL CRT,17;n` set the value of CRT control register 17. These statements have no effect on control registers 15 and 16 which are set using `PRINT PEN` and `KEY LABELS PEN`, respectively.

Note that the functionality of this statement can be achieved through CRT CONTROL register 17.

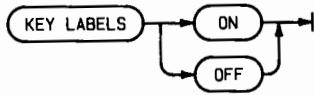
## KEY

See the `OFF KEY` and `ON KEY` statements.

# KEY LABELS

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

This statement turns the softkey labels on and off.



## Example Statements

```
KEY LABELS ON
KEY LABELS OFF
IF Key_labels_on THEN KEY LABELS OFF
```

## Semantics

With ITF keyboards, the default is KEY LABELS ON. (You can also turn the key labels on and off by using the **Menu** key.) The default softkey menu that appears with an ITF keyboard is the:

- **System** menu if the KBD binary has *not* been loaded.
- **User 1** menu if the KBD binary has been loaded.

The times when key labels are displayed depends on the current value of CRT STATUS register 12:

Value of CRT Register 12	Effect on Key Labels
0	Typing-aid key labels are <i>displayed until the program is run</i> , at which time they are turned off (until at least one ON KEY is executed). For more details on this mode, see the next table.
1	Typing-aid and softkey labels are <i>not displayed at any time</i> .
2	Typing-aid and softkey labels are <i>displayed at all times</i> .

The default value of this register is 0 for systems using 98203 keyboards, and 2 for systems using an ITF keyboard. The default is restored at power-on and when SCRATCH A is executed.

When the value of CRT register 12 is 0, softkey labels are ON or OFF as given in the following table:

<b>Condition</b>	<b>KBD Binary Not Loaded</b>	<b>KBD Binary Loaded</b>
No program running	Key labels off	Key labels on
Program running and ON KEY is not active	Key labels off	Key labels off
Program running and ON KEY is active	Key labels on	Key labels on

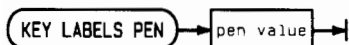
Executing a KEY LABELS ON or KEY LABELS OFF statement **does not** change the current softkey definitions.



## KEY LABELS PEN

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

This statement sets the pen color to be used for the softkey labels of the CRT.



Item	Description/Default	Range Restrictions
pen value	numeric expression	(see Semantics)

### Example Statements

```
KEY LABELS PEN Pen_value  
KEY LABELS PEN 143  
IF Color_blue THEN KEY LABELS PEN 141
```



## Semantics

This statement has no effect on monochrome displays.

The set of alpha colors for the Model 236C is given in the table below:

Value	Result
< 16	The number is evaluated MOD 8 and resulting values produce the following: 0 — black 1 — white 2 — red 3 — yellow 4 — green 5 — cyan 6 — blue 7 — magenta
16 to 135	Ignored
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black
144 to 255	Ignored

For displays with bit-mapped alpha, `KEY LABELS PEN` specifies the graphics pen to be used for subsequent alpha output. The range of values allowed with this statement are 0 through 255; these values are treated as “ $value \text{ MOD } (2^n)$ ” where  $n$  is the number of display planes.

`KEY LABELS PEN  $n$`  and `CONTROL CRT, 16;  $n$`  set the value of CRT control register 16. These statements have no effect on control registers 15 and 17 which are set using `PRINT PEN` and `KBD LINE PEN`, respectively.

Note that the functionality of this statement can be achieved through `CRT CONTROL` register 16.

## **KNOB**

---

See the OFF KNOB and the ON KNOB statements.

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 on an HP 98203A/B keyboard. HIL Knobs return a greater number of 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 at the end of each ON KNOB sampling interval. The pulse count during each sampling interval is limited to  $-127$  thru  $+128$  on an HP 98203A/B keyboard. HIL pulse counts are limited to  $-32\,768$  through  $+32\,767$  per sampling period. The limits of the KNOBX function are  $-32\,768$  thru  $+32\,767$ .

You can use a relative pointing device, such as the HP 46060A with an HP-HIL interface, if the KBD BIN is loaded.

---

### Note

KNOBX functions differently if BIN file KNB2\_0 is loaded. Refer to the Knob section of the "Porting to 3.0" chapter of the *BASIC Programming Techniques* manual for more information.

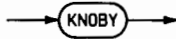
---

# 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 on an HP 98203A/B keyboard. HIL Knobs return a greater number of 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 at the end of each ON KNOB sampling interval. The pulse count during each sampling interval is limited to -127 thru +128 on an HP 98203A/B keyboard. HIL pulse counts are limited to -32 768 through +32 767 per sampling period. The limits of the KNOBY function are -32 768 thru +32 767.

You can use a relative pointing device, such as the HP 46060A with an HP-HIL interface, if the KBD BIN is loaded.

---

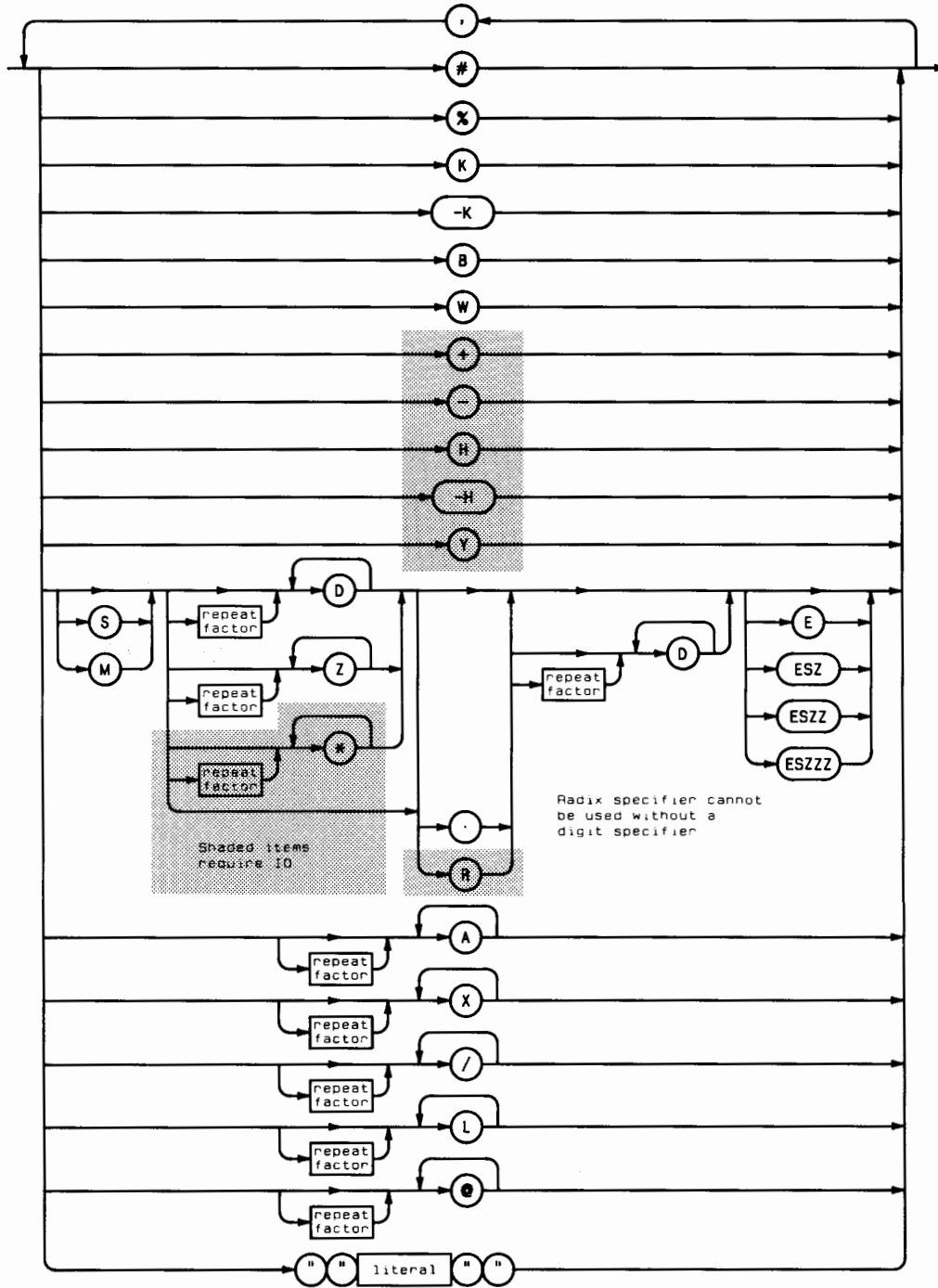
### Note

KNOBY functions differently if BIN file KNB2\_0 is loaded. Refer to the Knob section of the "Porting to 3.0" chapter of the *BASIC Programming Techniques* manual for more information.

---

# Notes







Item	Description	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
image specifier list	literal	(see diagram)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the <code>ANY CHAR</code> 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 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.

COMPLEX numbers are treated like two REAL numbers separated by a semicolon.

### 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

Character	Keystroke	Name	Action
CHR\$(8)	CTRL-H	backspace	Back up the width of one character cell.
CHR\$(10)	CTRL-J	line-feed	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**

Entire arrays may be output by using the asterisk specifier. Each element in an array is treated as an item by the LABEL 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. COMPLEX array elements are treated as if the real and imaginary parts are separated by a semicolon. The array is output in row major order (rightmost subscript varies fastest).

### **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 punctuation 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 specifier 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 there is no matching display item (and the specifier requires a display item). If the image specifiers are exhausted before the display items, they are reused, starting at the beginning.

COMPLEX values require two REAL image specifiers (i.e. each COMPLEX value is treated like two REAL values.)

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 right-most 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 LABEL statement are shown in the following table:

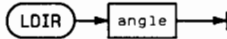
Image Specifier	Meaning
K	Compact field. Outputs a number or string as a label in standard form with no leading or trailing blanks.
-K	Same as K.
H	Similar to K, except the number is output using the European number format (comma radix). (Requires IO)
-H	Same as H. (Requires IO)
S	Outputs the number's 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	Range
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.

LDIR EXAMPLES (in Degrees)

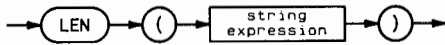
The diagram illustrates the effect of the LDIR command at various angles. The text 'LDIR' is centered, and the angle value is placed to its right. The text is rotated counter-clockwise from the horizontal line. The angles shown are 0, 45, 90, 135, 180, 225, 270, and 315 degrees.

Angle (Degrees)	LDIR Output Orientation
0	Horizontal
45	45 degrees counter-clockwise
90	Vertical
135	135 degrees counter-clockwise
180	Horizontal (inverted)
225	225 degrees counter-clockwise
270	Vertical (inverted)
315	315 degrees counter-clockwise

# 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

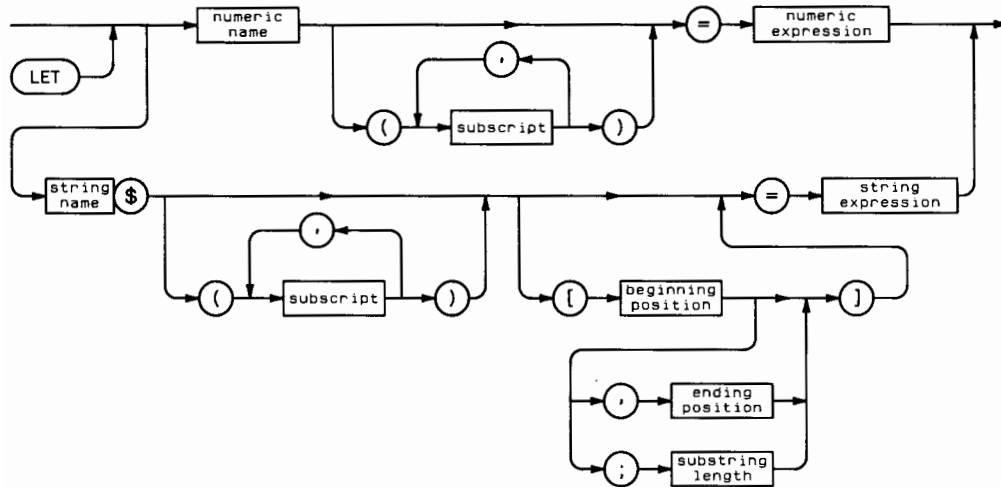
```
Last=LEN(String$)  
IF NOT LEN(A$) THEN Empty
```



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	Range
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 assignment 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. If a REAL or INTEGER value is assigned to a COMPLEX variable, the imaginary part receives the value 0. If a COMPLEX value is assigned to a REAL or INTEGER variable, the imaginary part is dropped.

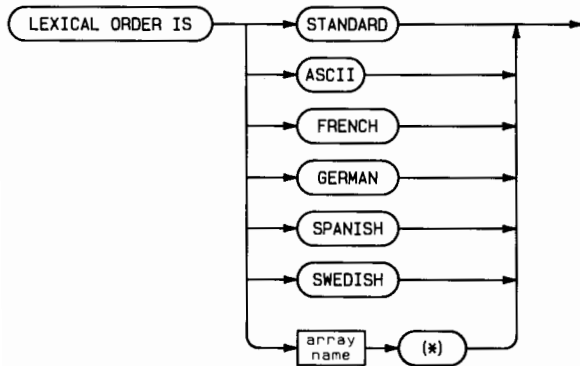
The length of the string expression must be less than or equal to the dimensioned length of the string it is being assigned to. Assignments may be made into substrings, using the normal rules for substring definition. The string expression will be truncated or blank-filled on the right (if necessary) 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	Range
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**

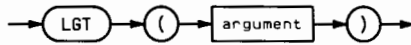
The STANDARD lexical order is determined by the internal keyboard jumper preset to match the language on the keyboard. For example, with an English language or Katakana keyboard, the STANDARD lexical order is the same as the ASCII lexical order.

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.

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	> 0 for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

**Examples Statements**

```
Decibel=20*LGT(Volts)
PRINT "Log base 10 of ";X;"=";LGT(X)
```

**Semantics**

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

To compute the LGT of a COMPLEX value, the COMPLEX binary must be loaded.

**Range Restriction Specifics**

The formula used for computing the LGT of a COMPLEX value is:

$$\text{LOG}(\text{Argument})/\text{LOG}(10)$$

where **Argument** is a COMPLEX argument to the LGT function. Some values of a COMPLEX argument may cause errors in this computation. For example,

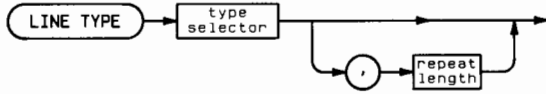
```
LGT(CMPLX(MAXREAL,MAXREAL))
```

will cause error 22 due to the LOG(**Argument**) calculation.

# 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	Range
type selector	numeric expression, rounded to an integer; Default = 1	1 thru 10
repeat length	numeric expression, rounded to an integer; Default = 5	greater than 0

## Example Statements

```
LINE TYPE 1
LINE TYPE Select,20
```

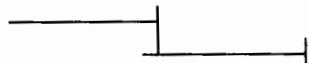
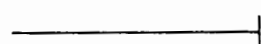
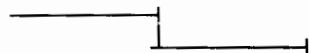

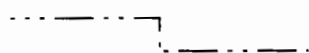

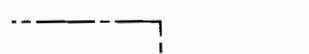

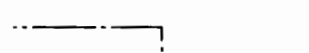

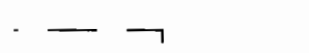

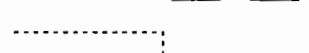

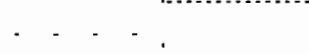



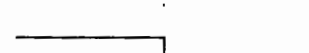

## Semantics

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 segment 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.

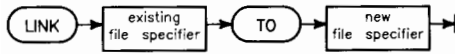
	LINE TYPE 10	
	LINE TYPE 9	
	LINE TYPE 8	
	LINE TYPE 7	
	LINE TYPE 6	
	LINE TYPE 5	
	LINE TYPE 4	
	LINE TYPE 3	
	LINE TYPE 2	
	LINE TYPE 1	



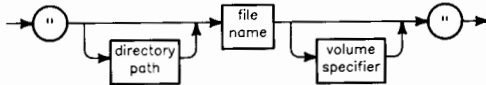
# LINK

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

This statement links a destination file on an HFS volume to a source file on the same volume.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see diagram)
volume specifier	string expression	(see MASS STORAGE IS)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)

## Example Statements

```
LINK "Sor_file" TO "Des_file"
LINK "Letter1:CS80,700" TO "Letter2"
LINK "/Home_dir/File1:CS80,700" TO "/Home_dir/Sub_dir/File2"
```



## Semantics

The LINK statement works only with HFS.

Executing a LINK between files that are on different volumes is not possible. Since the files that are linked **must** be on the same volume, you need to give the volume specifier only with the first file used by the link statement. For example,

```
LINK "File1:INTERNAL,4" TO "File2"
```

is a legal statement because the second file specifier's default msvs is the first file specifier's msvs. However, the following statement:

```
LINK "News1:CS80,700,1" TO "News2:9133,702,0"
```

will give an error because the files are on different volumes.

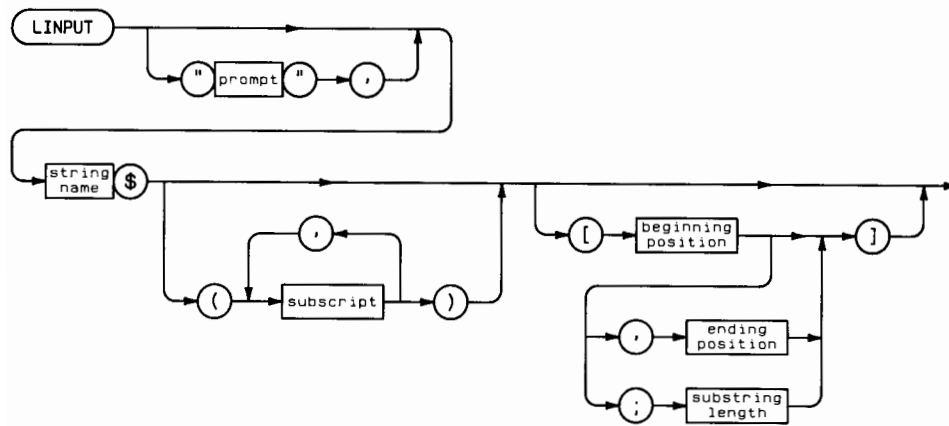
If you RE-STORE or RE-SAVE to a file that has links to it, the links will be broken.

If you OUTPUT to a file that has links to it, the linked files will all have the same contents.

# 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	Range
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.)

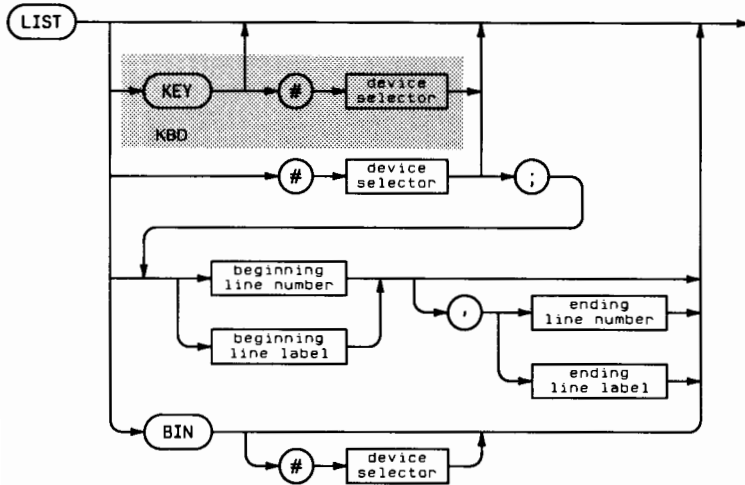
Live keyboard operations are not allowed while a LINPUT is waiting for data entry. [PAUSE] (or [Stop] on an ITF keyboard) can be pressed so live keyboard operations can be performed. The LINPUT statement is re-executed from the beginning when [CONTINUE] or [STEP] is pressed.

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	EDIT
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	Range
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.

Note that the default width of the PRINTER IS device is 80 characters, which means that a carriage-return (CR) and line-feed (LF) character will be sent after 80 characters are printed on any one line. You can change this, however, with the WIDTH attribute of the PRINTER IS statement.

LIST *#device selector* is always done as if the WIDTH OFF printer attribute was in effect.

### **LIST KEY (Requires KBD and Does Not Require EDIT)**

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 (Does Not Require EDIT)**

The LIST BIN statement lists the BINs currently loaded in memory. The name, version and brief description of the BIN is listed. For example:

<b>NAME</b>	<b>VERSION</b>	<b>DESCRIPTION</b>
GRAPH	5.0	Graphics
MAT	5.0	Matrix Statements

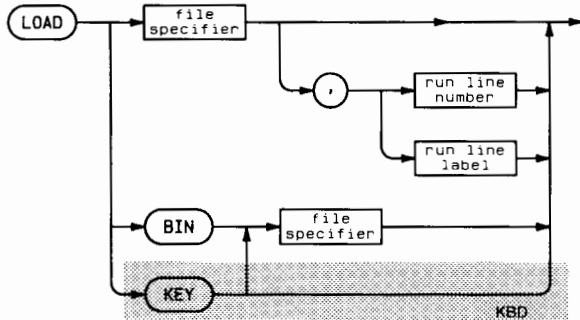
**LISTEN**

---

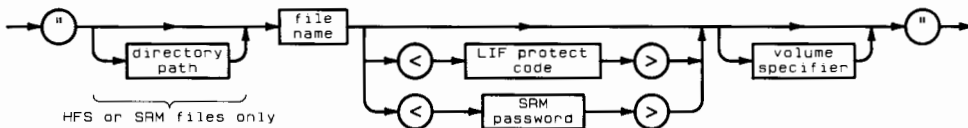
See the SEND statement.

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 softkey definitions.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	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 File_name$&&Volume$
LOAD "UTIL",120
LOAD BIN "MAT"
LOAD KEY "KEYS:INTERNAL,4,1"

LOAD BIN Dir$&&File$&&Volume$
LOAD "/Dir1/Dir2/Prog2",500
LOAD "Dir3/Prog_1:REMOTE"

LOAD BIN "dir1/dir2/bin_file<SRM_READ_pass>:REMOTE 21,5;LABEL Disc"

LOAD KEY "KEYS:REMOTE"
LOAD KEY "/Dir1/Dir2/Keyfile"
```

## 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 not compatible with the current version of BASIC, the binary is skipped, a warning is printed, and the program is loaded. If it contains a binary that is compatible with the current version of BASIC, the binary 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.



### **Autostart Program Files**

BASIC automatically loads and runs a file called AUTOST if the file exists on the boot mass storage device (and if it is of type PROG). If the system is loaded from an HFS volume, the autostart file is /WORKSTATIONS/AUTOST. If the system is loaded from SRM, the autostart file is /SYSTEMS/AUTOST $nn$ , where  $nn$  is the node number of the SRM interface installed in the computer you are booting; if this file does not exist, BASIC looks for /AUTOST.

### **HFS Permissions**

In order to LOAD a file from an HFS volume, you need to have R (read) permission on the file, and X (search) permission on the immediately superior directory as well as all other superior directories.

### **LOAD with SRM Volumes**

In order to LOAD a file from an SRM volume, you need to have READ access capability on the file, on the immediately superior directory, and on all other superior directories.

LOAD from an SRM volume 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 Requirements for Device Drivers**

With HFS volumes, you must have all drivers for the disc from which you are loading the program. For instance, if you are loading from a CS80-type disc, you must have HPIB, CS80, and HFS binaries currently in memory. (This is not like other volumes. It is required because the Boot ROM does not contain drivers for HFS volumes, but it does contain drivers for LIF and SRM volumes.)

### **LOAD BIN**

LOAD BIN is used to load BIN files. (A BIN file contains either language extensions, such as MAT or GRAPH, or drivers, such as DISC and HPIB.) A BIN file may contain more than one of the extensions or drivers; if so only the extensions or drivers not already present in memory are loaded. Executing a LOAD BIN does not affect the currently loaded BASIC program or any variables.



**LOAD BIN Requirements for Device Drivers**

BIN files can usually be loaded from a mass storage device even though the BIN(s) to access that device are absent (as long as the Boot ROM has the required drivers). Most Boot ROMs have drivers for LIF and SRM volumes and HPIB interfaces, but not for HFS volumes. If the Boot ROM does not have the drivers for the device from which you want to load a BIN file, then you will need to first install all required drivers for that device (such as HFS, CS80, and HPIB).

**LOAD BIN with SRM Volumes**

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 and DCOMM binaries present in the workstation (as described in the preceding paragraph). However, you cannot perform operations like STORE without these binaries.

**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.

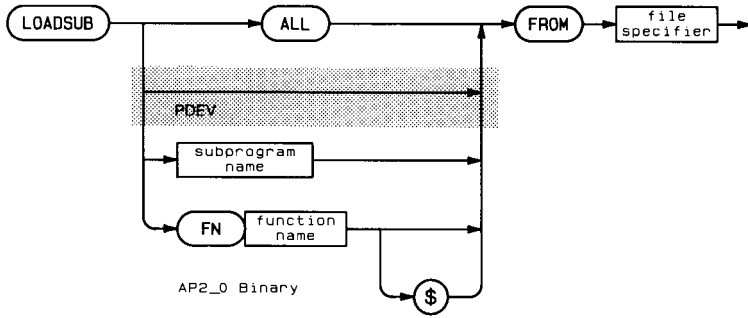
**LOAD KEY with SRM Volumes**

In order to LOAD KEY a file on an SRM volume, you need to have READ access capability on the immediately superior directory, as well as READ capability on all other superior directories.

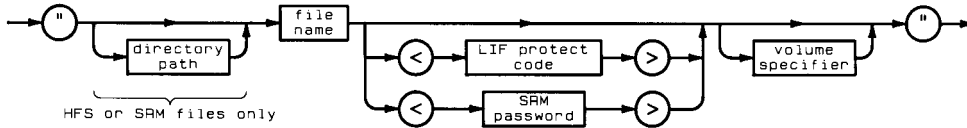
LOAD KEY is executed in shared mode, which means that several users can perform a LOAD KEY from a BDAT file at the same time. Files being stored with the STORE KEY or RE-STORE KEY statements are locked during that operation and cannot be accessed for loading.

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 memory.



literal form of file specifier:



Item	Description	Range
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)
directory path	literal	(see MASS STORAGE IS)
file name	literal	depends on volume's format (see Glossary)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
volume specifier	literal	(see MASS STORAGE IS)

### Example Statements

```
LOADSUB FROM "Subs_File"           (Not Programmable)
LOADSUB FNReplace$ FROM "Subs_File"
LOADSUB ALL FROM Subfile$
LOADSUB ALL FROM "Dir3/Progfile<SRM_READ_pass>"
LOADSUB ALL FROM "/Dir1/Dir2/Prog23"
```

## 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 in 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.

### HFS Permissions

In order to LOADSUB from a file on an HFS volume, you need to have R (read) access permission on the file, and X (search) permission on the immediately superior directory and on all other superior directories.

### LOADSUB with SRM Volumes

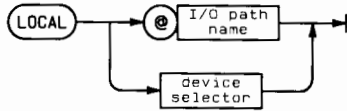
In order to LOADSUB from a file on an SRM volume, you need to have READ access capability on the immediately superior directory, as well as READ capability on all other superior directories.

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.

# 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	Range
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.

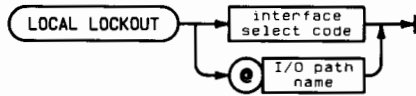
### Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{REN}$ 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	Range
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			

## LOCATOR

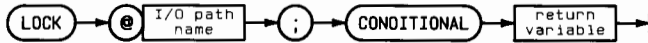
See the READ LOCATOR and SET LOCATOR statements.



# LOCK

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement prevents other SRM workstations from accessing the SRM file to which the I/O path name is currently assigned (see ASSIGN).



Item	Description	Range
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

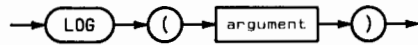
This statement establishes sole access to an SRM file that has been opened with an ASSIGN statement. This exclusive access remains assigned to the workstation executing the LOCK statement until an UNLOCK statement is executed by that workstation. The UNLOCK function is also a result of SCRATCH A, **RESET** 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.

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

This function returns the logarithm (base e) of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	> 0 for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

### Examples Statements

```
Time=-1*Rc*LOG(Volts/Emf)
PRINT "Natural log of ";Y;"=";LOG(Y)
```

### Semantics

If the argument is REAL or INTEGER, the value returned is REAL. If the argument is COMPLEX, the value returned is COMPLEX.

To compute the LOG of a COMPLEX value, the COMPLEX binary must be loaded.

### Range Restriction Specifics

The formula used for computing the LOG of a COMPLEX value is:

$$\text{CMPLX}(\text{LOG}(\text{ABS}(\text{Argument})), \text{ARG}(\text{Argument}))$$

where **Argument** is a COMPLEX argument to the LOG function. The imaginary part of the result (**ARG(Argument)**) is **always given in radians** and is between  $-\pi$  and  $+\pi$ . Some values of a COMPLEX argument may cause errors in this computation. For example:

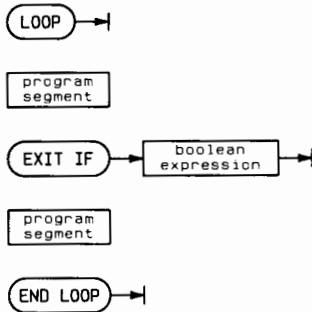
$$\text{LOG}(\text{CMPLX}(\text{MAXREAL}, \text{MAXREAL}))$$

will cause error 22 due to the **ABS(Argument)** computation.

# 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	Range
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.

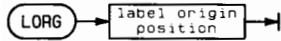
### **Nesting Constructs Properly**

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.



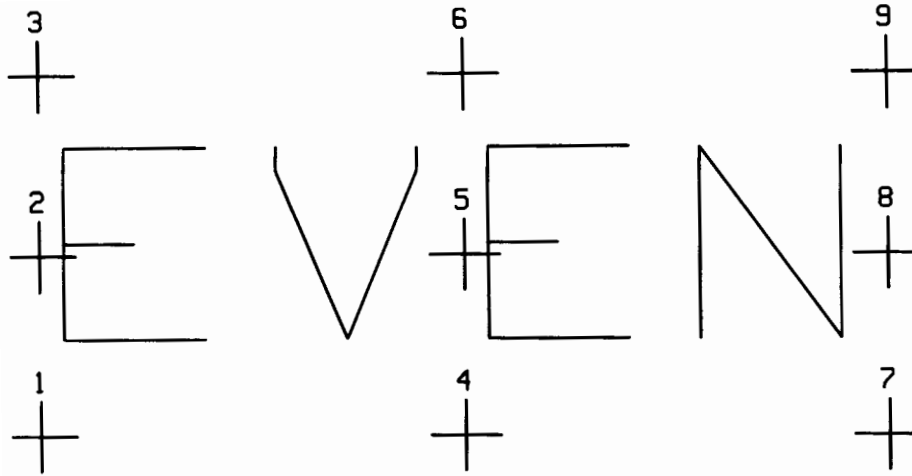
Item	Description	Range
label origin position	numeric expression, rounded to an integer; Default = 1	1 thru 9

## Example Statements

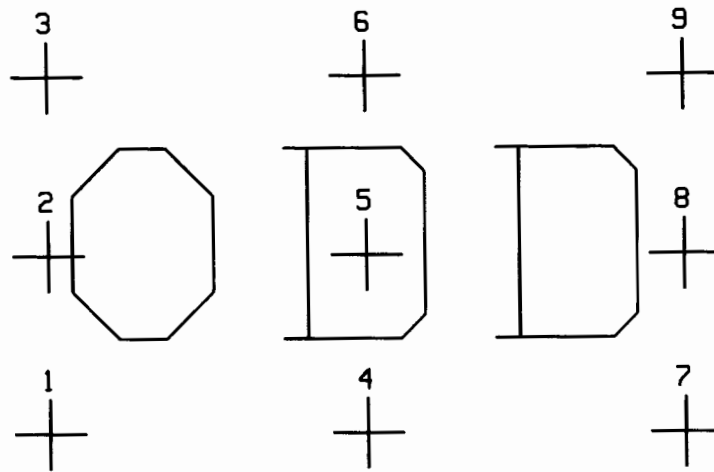
```
LORG 4  
IF Y>Limit THEN LORG 3
```

## 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**

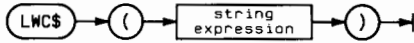


**Label Origins for Labels with an Odd Number of Characters**

# 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.

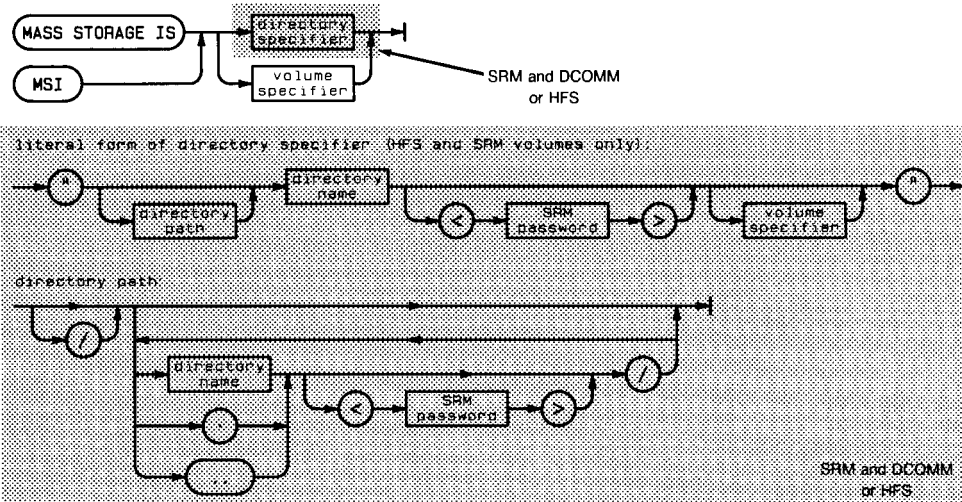
The corresponding characters for the Roman Extension alphabetic characters are determined by the current lexical order. When the lexical order is a user-defined table, the correspondence is determined by the STANDARD lexical order.



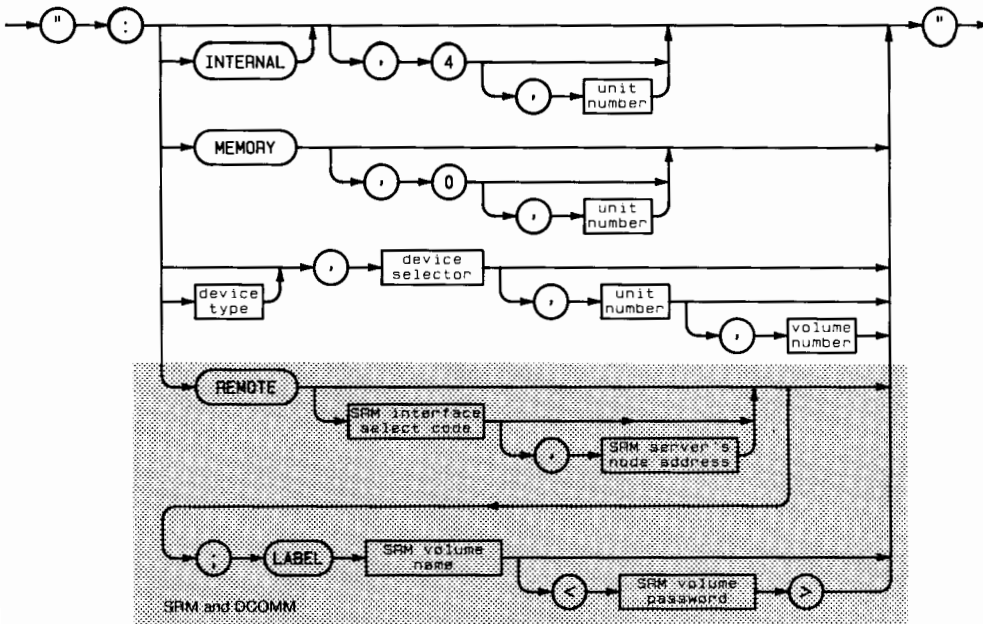
# MASS STORAGE IS

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

This statement specifies the system mass storage device.



literal form of volume specifier:



Item	Description	Range
directory specifier	string expression	(see drawing)
volume specifier	string expression	(see drawing)
directory path	literal	(see drawing)
directory name	literal	depends on volume's format (14 characters for HFS; 16 characters for SRM; see Glossary for details)
SRM password	literal; first 16 non-blank characters are significant	> not allowed
device type	literal	(see Semantics)
device selector	integer constant	(see Glossary)
unit number	integer constant; Default = 0	0 thru 255 (device-dependent)
volume number	integer constant; Default = 0	(device-dependent)
SRM interface select code	integer constant identifying the SRM interface <i>in the workstation</i>	8 through 31
SRM server's node address	literal	0 through 63
SRM volume name	literal	any valid SRM volume name (see Glossary)
SRM volume password	literal	any valid SRM volume password (see Glossary)

### Example Statements

```

MASS STORAGE IS Vol_specifier$
MSI " : ,700"
MSI " : INTERNAL,4,1"
MSI " : X,12"
MSI " : REMOTE"

```

```

MASS STORAGE IS Dir_path$&Vol_specifier$
MSI "/Dir1/Dir2/MyDir"
MSI "../.."
MSI ".<SRM_READ_pass>"

```



## Semantics

All mass storage operations which do not specify a source or destination by either an I/O path name or volume specifier 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.

Device Type	Binary Required
INTERNAL (Models 226 and 236 only)	none
MEMORY	
HP 9121 HP 9133 HP 9134 HP 9135 (5 <sup>1</sup> / <sub>4</sub> -inch disc requires HPIB not FHPIB) HP 913X HP 9895	DISC & HPIB or FHPIB
HP 82901 HP 82902 HP 8290X	DISC & HPIB
All "CS/80" and "SS/80" drives; for instance: HP 7908 HP 7914 HP 9122 HP 9153 HP 7946 etc.	CS80 & HPIB or FHPIB
HP 9885	HP9885 & GPIO
REMOTE (SRM)	SRM & DCOMM
BUBBLE	BUBBLE
EPROM	EPROM

---

**Note**

The 98625 Card (which requires the FHPIB binary) **cannot** be used with external 5<sup>1</sup>/<sub>4</sub>-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 assumed 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.

**MSI with SRM and HFS Volumes**

With hierarchical volumes (such as SRM and HFS), MASS STORAGE IS can also be used to specify the current working directory.

In order to specify an HFS directory as the current working directory, you need to have X (search) permission of the immediately superior directory as well as on all other superior directories.

In order to specify an SRM directory as the current working directory, you need to have READ access capability on all superior directories.

If you specify an SRM volume password in an MSI statement, that password is automatically applied to all accesses that use the default volume (that is, when no volume specifier is included in the file specifier) until a mass storage volume specifier is included in a subsequent MSI.

**MSI and Driver Binaries**

With BASIC 5.0, executing MASS STORAGE IS requires accessing the volume (since the system cannot determine whether a volume is a LIF or an HFS format). This operation requires that all drivers for the interface, disc, and directory format be present; for instance, you must have HPIB, CS80, and HFS binaries currently loaded in order to MSI to an HFS directory. If media is not present (hence cannot be accessed), BASIC assumes the volume is LIF format.

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	Range
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)
MAT Des_array(-1:0,2:4)= Sor_array
MAT Array_1= Array_2(-4:1)
MAT Destination(3,*,*)= Source(*,2,*)
MAT Result= REAL(Complex_array)
MAT New_array= IMAG(Complex_array)
MAT Return_ary= CMPLX(Array1,Array2)
MAT Des_ary= CONJG(Complex_array)
MAT Array= ARG(Complex_array)
MAT Matrix_1= (CMPLX(34.56,-23.78))
MAT Magnitudes= ABS(Complex_array)

```

## 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.
- Copy two REAL arrays into a COMPLEX array with the first REAL array representing the real part of the COMPLEX elements and the second REAL array representing the imaginary parts.
- Copy the contents of one array or subarray into another array or subarray.
- 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.
- Extract the real part or imaginary parts of a COMPLEX array.
- Compute the COMPLEX conjugate, argument or absolute value (magnitude) of COMPLEX array elements.
- Calculate the absolute value of an INTEGER, REAL, or COMPLEX array.

---

### 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). Note that “size” refers to the number of elements in the array and “shape” refers to the same number of dimensions and elements in each dimension, respectively (e.g. both of these subscript specifiers have the same shape: (-2:1, -1:10) and (1:4, 9:20)). 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. Note that the **only** comparison operators allowed between COMPLEX expressions or arrays are: = and <>.

### **Complex Operations**

Complex functions can be used by the MAT statement to:

- extract the real and/or imaginary parts of a COMPLEX array and assign them to a resultant array,
- create a COMPLEX array from two REAL or INTEGER arrays,
- compute the COMPLEX conjugate, argument or absolute value (magnitude) of COMPLEX array elements and assign them to a resultant array.

The complex functions available with the MAT statement are given below. The arguments used with these functions can be INTEGER, REAL or COMPLEX arrays.

<b>REAL(Array)</b>	assigns the real parts of each array element to the corresponding element in the resultant array.
<b>IMAG(Array)</b>	assigns the imaginary parts of each array element to the corresponding element in the result array.
<b>CMPLX(Array1,Array2)</b>	creates COMPLEX values from two numeric arrays by assigning each element of the first array ( <b>Array1</b> ) to the corresponding real part of the result array element, and each element of the second array ( <b>Array2</b> ) to the corresponding imaginary part of the result array element (if it is COMPLEX). If either of the array arguments are COMPLEX arrays, only the real part of that array is used in the creation of the values.
<b>CONJG(Array)</b>	computes the conjugate of each COMPLEX array element and assigns this result to the corresponding element in the result array.
<b>ARG(Array)</b>	computes the argument of each COMPLEX array element and assigns this result to the corresponding element in the result array.
<b>ABS(Array)</b>	computes the absolute value of each COMPLEX array element and assigns this result to the corresponding element in the result array.

### 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.

If either array is COMPLEX, the operation is done in COMPLEX math. Otherwise, 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 matrices 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 matrices.

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.

### Copying Subarrays

A subarray is a subset of an array (an array within an array). A subarray is indicated by a specifier after the array name as follows:

```
Array_name(subarray_specifier)  
Array_name$(subarray_specifier)
```

For example if you wanted to specify the entire second column of a 3×3 two-dimensional array, you would use the following subarray:

```
Array_name(*,2)
```

Copying subarrays is useful when you need to:

- Copy a Subarray into an Array
- Copy an Array into a Subarray
- Copy a Subarray into a Subarray
- Copy a Portion of an Array into Itself

For a complete discussion of this subject, read the section entitled “Copying Subarrays” found in the “Numeric Arrays” chapter of the *BASIC Programming Techniques* manual. (Note that you can also copy subarrays of string arrays.)

Before discussing the rules for subarrays the concept of range needs to be understood as it appears in this text. There are two types of ranges to be considered when using subarrays they are the: *subscript range* and *default range*. The *subscript range* is used to specify a set of elements starting with a beginning element position and ending with a final element position. For example, 5:8 represents a range of four elements starting with element 5 and ending at element 8. The *default range* is denoted by an asterisk (\*) and represents all of the elements in a dimension from the dimension’s lower bound to its upper bound. For example, suppose you wanted to copy the entire first column of a two dimensional array, you would use the following subarray specifier: (\*,1), where \* represents all the rows in the array and 1 represents **only** the first column.

Some rules to follow when copying subarrays are as follow:

- Subarray specifiers **must not** contain all subscript expressions (i.e. (1,2,3) is not allowed, it will produce a syntax error). This rule applies to all subarray specifiers.
- Subarray specifiers **must not** contain all asterisks (\*) or default ranges (i.e. (\*,\*,\*) is not allowed, it will produce a syntax error). This rule applies to all subarray specifiers.
- If two subarrays are given in a MAT statement, there **must be** the same number of ranges in each subarray specifier. For example,

```
MAT Des_array1(1:10,2:3)= Sor_array(5:14,*,3)
```

is the correct way of copying a subarray into another subarray provided the default range given in the source array (Sor\_array) has only two elements in it. Note that the source array is a three-dimensional array. However, it still meets the criteria of having the same number of ranges as the destination array because two of its entries are ranges and one is an expression.

- If two subarrays are given in a **MAT** statement, the subscript ranges in the source subarray **must be** the same shape as the subscript ranges in the destination subarray. For example, the following is **legal**:

```
MAT Des_array(1:5,0:1)= Sor_array(3,1:5,6:7)
```

however, the one below is **not** legal:

```
MAT Des_array(0:1,1:5)= Sor_array(1:5,0:1)
```

because both of its subarray specifiers do not have the same shape (i.e. the rows and columns in the destination subarray do not match the rows and columns in the source subarray).



## 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 1s along the main diagonal and 0s 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.

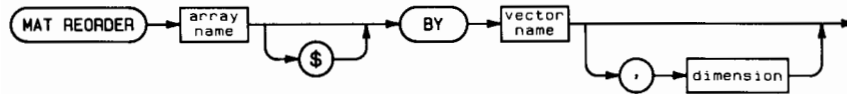
The transpose of an N-by-M matrix is an M-by-N matrix, and each element is defined by switching the subscripts. That is,  $\mathbf{A}(m,n)$  in the argument matrix equals  $\mathbf{B}(n,m)$  in the result matrix. (This description assumes that the array subscripts run from 1 through M and 1 through N; 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	Range
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

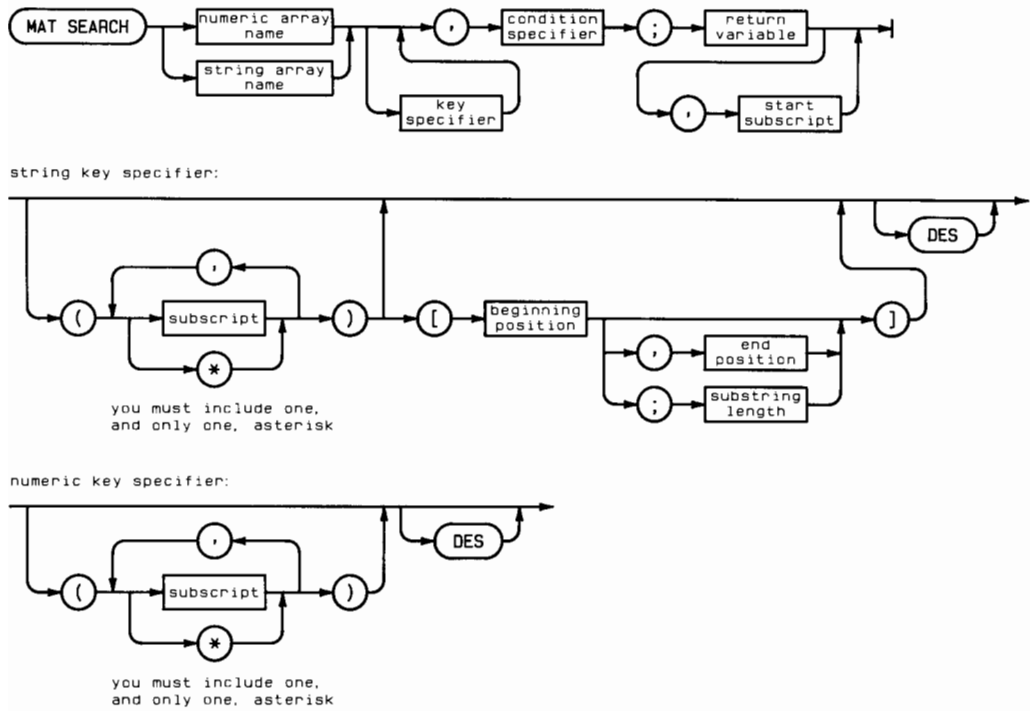
The *dimension* parameter is used to specify which dimension in a multidimensional array is to be reordered. If no dimension is specified, the computer defaults to dimension 1. The vector must be the same size as the specified dimension and it should contain integers corresponding to the subscript range of that dimension (no duplicate numbers, or numbers out of range). The vector used cannot be a COMPLEX vector, but COMPLEX arrays can be re-ordered.

Vectors generated by a MAT SORT TO statement are of the proper form for reordering (see MAT SORT).

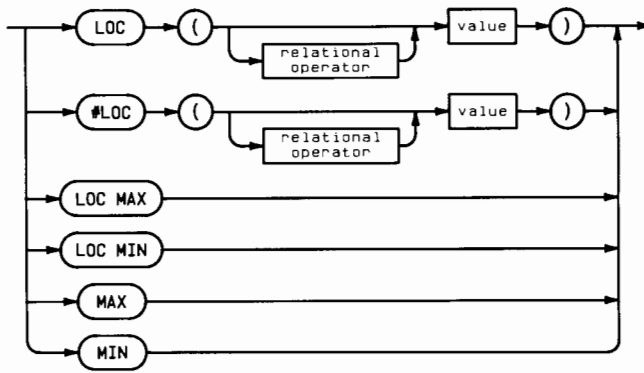
# MAT SEARCH

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

This statement searches for user-defined conditions within a numeric or string array and places information returned into a variable for recall and examination.



condition specifier:



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
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 "string" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "string" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "string" in Glossary)
return variable	name of a numeric or string variable (as appropriate)	see "Semantics"
starting subscript	starting location within the vector being searched	-32 768 thru +32 767 (see "array" in Glossary)

### Example Statements

```

MAT SEARCH Number_array,LOC MAX;Loc_max
MAT SEARCH Source,LOC(<3);Location,4
MAT SEARCH Array(1,*) DES,LOC MIN;Loc_min,6
MAT SEARCH Vector,#LOC(<>2);Non_two_values
MAT SEARCH String$(*,2,3) DES,MAX;Max_value$
MAT SEARCH Word$(*,2),MIN;Min_plus$[2;10],Start_pos
  
```

## Semantics

Numeric searching comparisons are performed in the `INTEGER` mode for integer-precision arrays, in the `REAL` mode for real-precision arrays, and in the `COMPLEX` mode for complex arrays (which use real-precision for their real and imaginary parts). Note that the only numeric comparisons allowed in the `COMPLEX` mode are `=` and `<>` relational operators.

String arrays can be searched using all of the condition specifiers mentioned above. (If you want to search a substring within each element, see the subsequent discussion of searching substrings.)

The default search order for arrays is from the lower bound to the upper bound. This is considered to be an ascending search order. You can search an array in a descending search order (upper bound to lower bound) by using the secondary keyword `DES` in the key specifier. The following table clarifies the two types of search orders:

Search Order	Starting Subscript Given	No Starting Subscript Given
ascending (default)	starting subscript → upper bound	lower bound → upper bound
descending	starting subscript → lower bound	upper bound → lower bound

The remaining sections mainly talk about numeric arrays; however, these sections also apply to searching string arrays.

### DES

To search an array by descending subscript values, use the secondary keyword `DES` in the array's key specifier:

```
MAT SEARCH Array(1,*) DES,MAX;Max_value,6
```

This secondary keyword causes a search to begin at the upper subscript bound and proceed toward the lower bound of that same dimension. If a *starting subscript* is specified in the `MAT SEARCH` statement, then the search will begin at that specified location in the dimension being searched and proceed toward the lower bound.

**LOC(relational comparison)**

LOC is used in the **MAT SEARCH** process for numeric and string arrays to scan the specified locations until it finds the first value which makes the comparison true. The *relational comparison* it uses is made up of two parts: an operator and a string or numeric expression (e.g. (>10) or (<>"CAT")). Operators determine the type of comparisons made. The following operators may be used: >, <, =, >=, <=, <>. The default relational operator is =.

When LOC is executed the value returned to the *return variable* is the subscript of the first location found that satisfied the LOC condition.

**#LOC(relational comparison)**

The condition #LOC is used in the **MAT SEARCH** process for numeric and string arrays to scan the specified locations and return a count of the number of locations whose contents satisfy the condition. The explanation for the *relational comparison* for this condition is the same as that given in the previous section.

**MAX**

MAX is used in the **MAT SEARCH** process for numeric and string arrays to scan all specified locations to find and return the maximum value found in the search. If the array is a string array, a string value is returned.

**LOC MAX**

LOC MAX is used in the **MAT SEARCH** process for numeric and string arrays to scan all specified locations to find and return the subscript of the first location in which the maximum value was found.

**MIN**

MIN is used in the **MAT SEARCH** process for numeric and string arrays to scan all specified locations to find and return the minimum value found in the search. If the array is a string array, a string value is returned.

## LOC MIN

LOC MIN is used in the MAT SEARCH process for numeric and string arrays to scan all specified locations to find and return the subscript of the first location in which the minimum value was found.

### Searching Substrings

To search a substring of each string array element, specify that substring (in square brackets) as part of the key specifier. For example:

```
MAT SEARCH A$(*,1)[3,5],LOC("CAT");B$
```

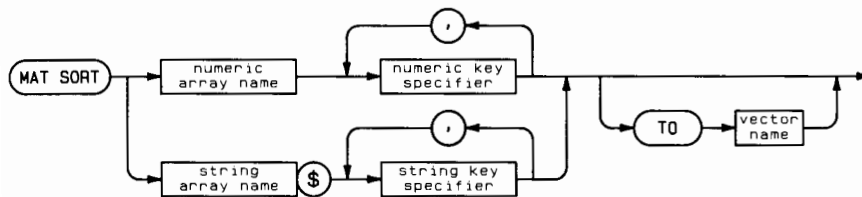
searches the 3rd thru 5th characters of each string for the string value CAT. Note that a MAT SEARCH of string arrays allows you not only to define the elements to be searched, but also to define substrings within each element. Substrings may lie anywhere within the dimensioned size of the string. If a substring lies outside the current string length, the null string is used as the searching key.



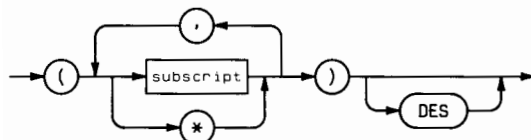
# MAT SORT

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.



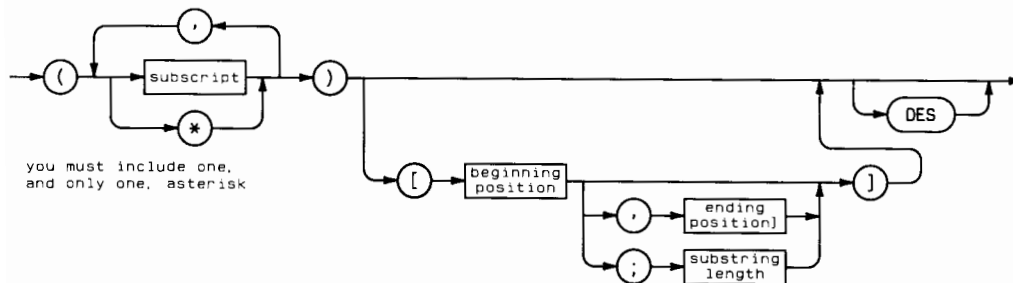
numeric key specifier:



you must include one,  
and only one, asterisk



string key specifier:



you must include one,  
and only one, asterisk

Item	Description	Range
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 (\*), (\*) **DES**, **DES**, or a blank specifier appears in the list following the array name, no other items can be added. Note that **COMPLEX** arrays **cannot** be **MAT SORTed**, because this statement's sorting routine uses < and > relational comparisons and these comparisons **are not** allowed with **COMPLEX** data types.

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.

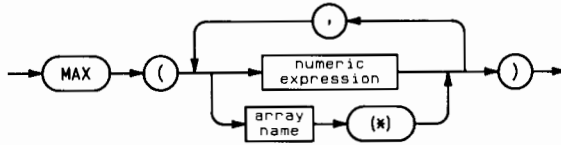
**MAT SORT** of string arrays allows you not only to define the elements to be sorted, but also to define substrings within each element. Substring specifiers refer only to the key specifier that immediately precedes them. Substrings may lie anywhere within the dimensioned size of the string. If a substring 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	Range
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))
```

## Semantics

COMPLEX arguments are not allowed with this function.

---

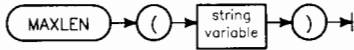
### Note

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.

---

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

This function returns the maximum (declared) length of a string variable (that is, the length specified in the DIM, COM, or ALLOCATE statement that declared the variable, or an implicitly declared of length 18).



Item	Description	Range
string variable	any simple string variable or subscripted string array element	(see the "Semantics" section given below)

## Example Statements

```

MAXLEN(String$)
N_columns=MAXLEN(String_array$(0))
  
```

## Semantics

If the length of a string variable is not explicitly declared (such as by COM or DIM) before it appears in a program, it will automatically have a length of 18 characters.

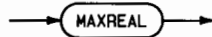
This function does not return the *current* length of the variable; use the LEN function for that purpose.

# 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/(10^N) THEN GOTO 100
```

## Semantics

The value of MAXREAL is approximately 1.797 693 134 862 32 E+308.

# MERGE ALPHA WITH GRAPHICS

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

This statement is used to merge the “simulated” separate alpha and graphics rasters set up by the SEPARATE ALPHA FROM GRAPHICS statement.



## Example Statements

MERGE ALPHA

IF Done THEN MERGE ALPHA WITH GRAPHICS



## Semantics

This statement is used to return the simulated separate alpha and graphics rasters on multi-plane bit-mapped alpha displays to their “overlapped” (default) mode. If the display is *not* a bit-mapped alpha display, an error will be reported.

The statement performs the following actions:

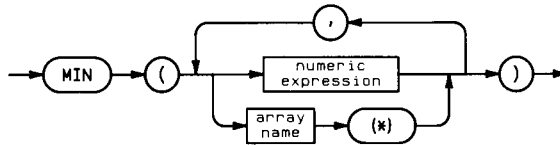
1. PLOTTER IS CRT, “INTERNAL” is executed.
2. If the display is a multi-plane, bit-mapped alpha display, then the following actions are also taken:
  - a. The alpha mask is set to its maximum value ( $2^n - 1$ , where  $n$  is the number of display planes).
  - b. The alpha pen is set to its default color (green).
  - c. The display is cleared (CLEAR SCREEN).
  - d. The graphics mask is set to its maximum value ( $2^n - 1$ , where  $n$  is the number of display planes).
  - e. The color map is re-initialized (to the default entries)

Here is a BASIC program that performs similar configuration of a 4-plane, bit-mapped alpha display:

```
100 PLOTTER IS CRT,"INTERNAL"    ! To enable GESCAPE.
110 ALPHA MASK 15                ! Restore default.
120 CLEAR SCREEN
130 ALPHA ON                     ! Display alpha plane.
140 GRAPHICS ON                  ! Display graphics planes.
150 INTEGER Gm(O)                ! Set up array for GESCAPE.
160 Gm(O)=15                     !
170 GESCAPE CRT,7,Gm(*)          ! Restore default.
180 PLOTTER IS CRT,"INTERNAL"    ! To reset color map.
190 ALPHA PEN 4                  ! Restore default.
200 END
```

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	Range
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))
```

## Semantics

COMPLEX arguments are not allowed with this function.

---

### 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*(10-N) THEN GOTO 100
```

## Semantics

The value of MINREAL is approximately 2.225 073 858 507 2 4E-308.



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

This operator returns the remainder of a division.



Item	Description	Range
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.

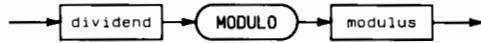
For INTEGERS, MOD is equivalent to  $X - Y \times (X \text{ DIV } Y)$ . This may return a different result from the modulus function on other computers when negative numbers are involved.

COMPLEX arguments are not allowed with this function.

# 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	Range
dividend	numeric expression	range of REAL
modulus	numeric expression	range of REAL, $\neq 0$

## Example Statements

```
Remainder=Dividend MODULO Modulus  
A=B MODULO C
```

## Semantics

$X \text{ MODULO } Y$  is equivalent to  $X - Y \times \text{INT}(X/Y)$ .

The result satisfies:

$$0 \leq (X \text{ MODULO } Y) < Y \text{ if } Y > 0$$
$$Y < (X \text{ MODULO } Y) \leq 0 \text{ if } Y < 0$$

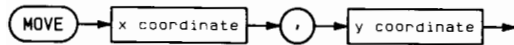
The type of the result is the higher of the types of the two operands. If the modulus is zero error 31 occurs.

MODULO returns the remainder of a division.

COMPLEX arguments are not allowed with this function.

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	Range
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.

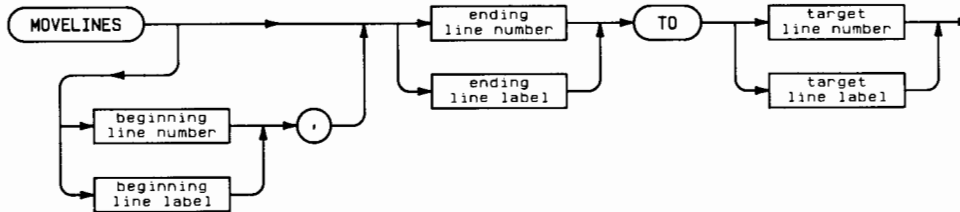
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.

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	Range
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 MOVE LINES (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.

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.



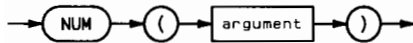
## Example Statements

```
IF NPAR>3 THEN Extra  
Factors=NPAR-2
```



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	Range
argument	string expression	not a null string

## Example Statements

```
Letter=NUM(String$)
A$(I;1)=CHR$(NUM(A$(I))+32)
```



## Notes