BASIC Language Reference

Volume 2: O-Z

HP 9000 Series 200/300 Computers

HP Part Number 98613-90052

HEWLETT
PACKARD

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

NOTICE

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

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

WARRANTY

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

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Software clause in FAR 52.227-7013.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

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.)

August 1988...Edition 3. This edition contains new information regarding the BASIC/UX 5.5. There are no changes to the BASIC Workstation pages, and the previous update has been merged.

Table of Contents

Volume 1

Volume 2

Appendix D: Useful Tables

Appendix F: Keyword Summary

OFF CDIAL

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement disables any ON CDIAL branching currently set up.

Example Statements

```
100  OFF CDIAL
200  IF Done THEN OFF CDIAL
```

OFF CYCLE

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON CYCLE statement.

Example Statements

```
OFF CYCLE
IF Kick_stand THEN OFF CYCLE
```

Semantics

OFF CYCLE destroys the log of any CYCLE event which has already occurred but which has not been serviced.

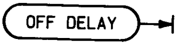
If OFF CYCLE is executed in a subprogram such that it cancels an ON CYCLE in the calling context, the ON CYCLE definition is restored upon returning to the calling context.

BASIC/UX Specifics

Resolution is 20 miliseconds. A new child process of BASIC/UX is started for the timer.

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON DELAY statement.



Example Statements

```
OFF DELAY
IF Ready THEN OFF DELAY
```

Semantics

OFF DELAY destroys the log of any DELAY event which has already occurred but which has not been serviced.

If OFF DELAY is executed in a subprogram such that it cancels an ON DELAY in the calling context, the ON DELAY definition is restored upon returning to the calling context.

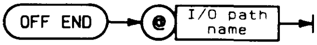
BASIC/UX Specifics

Resolution is 20 milliseconds. A new child process of BASIC/UX is started for the timer.

OFF END

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement cancels event-initiated branches previously enabled and defined by an ON END statement.



Item	Description	Range
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)

Example Statements

```
OFF END @File
IF Special THEN OFF END @Source
```

Semantics

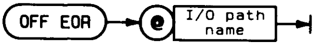
If OFF END is executed in a subprogram and cancels an ON END in the context which called the subprogram, the ON END definitions are restored when the calling context is restored.

If there is no ON END definition in a context, end-of-file and end-of-record are reported as errors.

OFF EOR

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON EOR statement.



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

Example Statements

```
OFF EOR @File
OFF EOR @Device_selector
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file or pipe; however, if the I/O path is assigned to a BUFFER, an error is reported when the OFF EOR statement is executed.

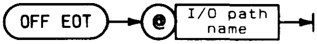
OFF EOR destroys the log of any EOR event which has already occurred but which has not been serviced.

If OFF EOR is executed in a subprogram such that it cancels an ON EOR in the calling context, the ON EOR definition is restored upon returning to the calling context.

OFF EOT

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON EOT statement.



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

Example Statements

```
OFF EOT @File
IF Done_flag THEN OFF EOT @Info
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file or pipe; however, if the I/O path is assigned to a BUFFER, an error is reported when the OFF EOT statement is executed.

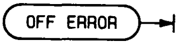
OFF EOT destroys the log of any EOT event which has already occurred but which has not been serviced.

If OFF EOT is executed in a subprogram such that it cancels an ON EOT in the calling context, the ON EOT definition is restored upon returning to the calling context.

OFF ERROR

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

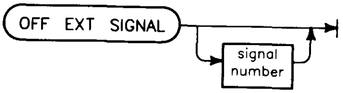
This statement cancels event-initiated branches previously defined and enabled by an ON ERROR statement. Further errors are reported to the user in the usual fashion.



OFF EXT SIGNAL

Supported On	UX
Option Required	n/a
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels event-initiated branches previously defined by an ON EXT SIGNAL statement.



Item	Description	Range
signal number	numeric expression, rounded to integer	1 thru 32 (see ON EXT SIGNAL)

Example Statements

```
OFF EXT SIGNAL 4
OFF EXT SIGNAL
```

Semantics

Not specifying a system signal number disables the event-initiated branches for all system signals. Specifying a signal number causes the OFF EXT SIGNAL to apply to the event-initiated log entry for the specified signal only.

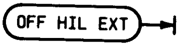
Any pending ON EXT SIGNAL branches for the affected signals are lost and further signals are vectored to the default handler for the EXT SIGNAL. See ON EXT SIGNAL for a description of the default actions for each EXT SIGNAL.

The action to be taken for an EXT SIGNAL is inherited when entering a new context (subprogram). This action stays in effect until an ON EXT SIGNAL or OFF EXT SIGNAL is executed. When an OFF EXT SIGNAL is executed within a context, the action for that external signal reverts to its default action. When the context is exited, the current action reverts to what it was in the calling context.

OFF HIL EXT

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement disables an end-of-line interrupt previously enabled by an ON HIL EXT statement. When this statement is executed, any pending ON HIL EXT branch is cancelled.



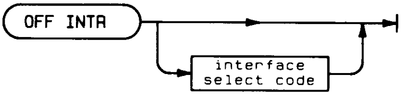
Example Statement

```
OFF HIL EXT
IF NOT Hil_active THEN OFF HIL EXT
```

OFF INTR

Supported On	WS,UX
Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels event-initiated branches previously defined by an ON INTR statement.



Item	Description	Range
interface select code	numeric expression, rounded to an integer; Default = all interfaces	5, and 7 thru 31

Example Statements

```
OFF INTR
OFF INTR Hpib
```

Semantics

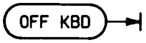
Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the OFF INTR to apply to the event-initiated log entry for the specified interface only.

Any pending ON INTR branches for the effected interfaces are lost and further interrupts are ignored.

OFF KBD

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement cancels the event-initiated branch previously defined by an ON KBD statement.



Example Statements

```
OFF KBD
IF NOT Process_keys THEN OFF KBD
```

Semantics

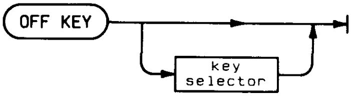
When this statement is executed, any pending ON KBD branch is cancelled, and the keyboard buffer is cleared.

If OFF KBD is executed in a subprogram such that it cancels an ON KBD in the calling context, the cancelled ON KBD definition is restored when the calling context is restored. However, the keyboard buffer's contents are not restored with the calling context, because the buffer was cleared with the OFF KBD.

OFF KEY

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON KEY statement.



Item	Description	Range
key selector	numeric expression, rounded to an integer; Default = all keys	0 thru 19

Example Statements

```
OFF KEY
OFF KEY 4
```

Semantics

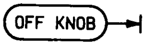
Not specifying a softkey number disables the event-initiated branches for all softkeys. Specifying a softkey number causes the OFF KEY to apply to the specified softkey only. If OFF KEY is executed in a subprogram and cancels an ON KEY in the context which called the subprogram, the ON KEY definitions are restored when the calling context is restored.

Any pending ON KEY branches for the effected softkeys are lost. Pressing an undefined softkey generates a beep.

OFF KNOB

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

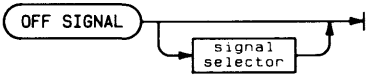
This statement cancels event-initiated branches previously defined and enabled by the ON KNOB statement. Any pending ON KNOB branches are lost. Further use of the knob will result in normal scrolling or cursor movement.



OFF SIGNAL

Supported On	WS,UX
Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

OFF SIGNAL cancels the ON SIGNAL definition with the same signal selector. If no signal selector is provided, all ON SIGNAL definitions are cancelled. OFF SIGNAL only applies to the current context.



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

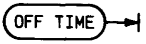
Example Statements

```
OFF SIGNAL
OFF SIGNAL 15
```

OFF TIME

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON TIME statement.



Example Statements

```
OFF TIME
IF Attended THEN OFF TIME
```

Semantics

OFF TIME destroys the log of any TIME event which has already occurred but which has not been serviced.

If OFF TIME is executed in a subprogram such that it cancels an ON TIME in the calling context, the ON TIME definition is restored upon returning to the calling context.

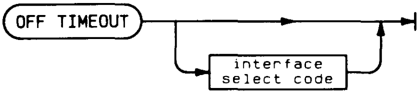
BASIC/UX Specifics

Resolution is 20 milliseconds. A new child process of BASIC/UX is started for the timer.

OFF TIMEOUT

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON TIMEOUT statement.



Item	Description	Range
interface select code	numeric expression, rounded to an integer; Default = all interfaces	7 thru 31

Example Statements

```
OFF TIMEOUT
OFF TIMEOUT Isc
```

Semantics

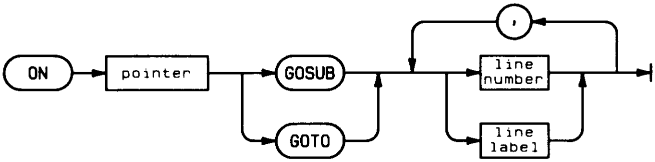
Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the OFF TIMEOUT to apply to the event-initiated branches for the specified interface only. When OFF TIMEOUT is executed, no more timeouts can occur on the effected interfaces.

BASIC/UX Specifics

All channels of MUX interfaces have timeouts disabled by OFF TIMEOUT without an interface select code.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement transfers program execution to one of several destinations selected by the value of the pointer.



Item	Description	Range
pointer	numeric expression, rounded to an integer	1 thru 74
line number	integer constant identifying a program line	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
ON X1 GOTO 100,150,170
IF Point THEN ON Point GOSUB First,Second,Third,Last
```

Semantics

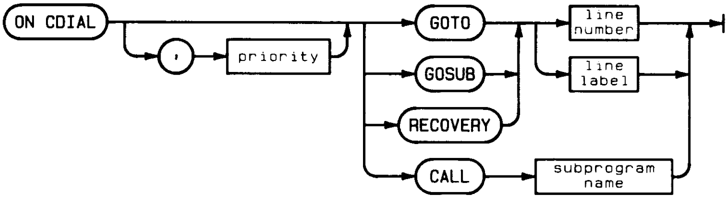
If the pointer is 1, the first line number or label is used. If the pointer is 2, the second line identifier is used, and so on. If GOSUB is used, the RETURN is to the line following the ON...GOSUB statement.

If the pointer is less than 1 or greater than the number of line labels or numbers, error 19 is generated. The specified line numbers or line labels must be in the same context as the ON statement.

ON CDIAL

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement sets up and enables a branch to be taken upon sensing rotation of one of the dials on a "control dial" device.



Item	Description	Range
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid line name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
100  ON CDIAL GOSUB Dial_serv_rtn
200  ON CDIAL,Priority CALL Dial_sub
```

Semantics

All CDIAL function registers are automatically cleared when ON CDIAL is executed.

The interrupt service routine for the branch initiated when one of the control dials is rotated should read the number of pulses with the CDIAL function.

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.

The most recent ON CDIAL (or OFF CDIAL) overrides any previous ON CDIAL branching. If the overriding branch is defined in another context (such as in a SUB subprogram or user-defined FN), then the overriding branch is canceled and the overridden branch is restored upon return to the calling context.

The ON CDIAL statement behaves like the ON KNOB and ON HIL EXT statements:

- When ON CDIAL is executed in a SUB context and program control exits that context, the pulses from control dials will continue to be accumulated (and can be read by CDIAL). No interrupts occur if there is no ON CDIAL active in the current context.
- Conversely, if an ON CDIAL has been executed in a context and then OFF CDIAL is executed in a called context, then upon returning to the calling context the pulses will be routed to the BASIC system (instead of the CDIAL function) and no interrupts will be initiated.

The priority can be specified, with the highest represented by a value of 15. (This is the highest user-specifiable priority; however, ON END and ON TIMEOUT have priorities of 16, and ON ERROR has a priority of 17.) An ON CDIAL branch can interrupt the currently executing program segment, if its priority is higher than the current SYSTEM PRIORITY (use `SYSTEM$("SYSTEM PRIORITY")` to determine the current priority).

Upon completion of the interrupt service routine, CALL and GOSUB branches are returned to the next line that would have been executed if the ON CDIAL branch had not been serviced; the system priority is returned to the value in effect before the ON CDIAL branch occurred. RECOVER forces the program to go directly to the specified line in the context containing the ON CDIAL statement; when RECOVER forces a change of context, the system priority is restored to the value which existed in the original (defining) context at the time that the context was exited.

CALL and RECOVER remain active (that is, they can initiate branches) when the context changes to a subprogram (SUB), unless the change in context is caused by a keyboard-originated CALL statement. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch is not initiated until after the calling context is restored.

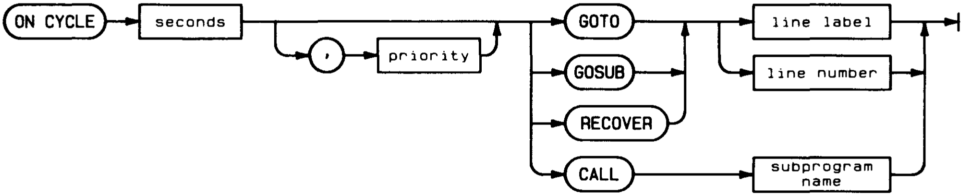
ON CDIAL branches are disabled by DISABLE, temporarily disabled when the program is executing an INPUT, LINPUT, or ENTER KBD... statement; and deactivated by OFF CDIAL.

ON CDIAL does not initiate branches for other “knob” devices (such as built-in knobs of 98203 keyboards or HIL mouse devices).

ON CYCLE

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken each time the specified number of seconds has elapsed.



Item	Description	Range
seconds	numeric expression, rounded to the nearest 0.02 second	0.01 thru 167 772.16
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON CYCLE 1 GOSUB One_second
ON CYCLE 3600,12 CALL Chime
```

Semantics

The most recent ON CYCLE (or OFF CYCLE) definition overrides any previous ON CYCLE definition. If the overriding ON CYCLE definition occurs in a context different from the one in which the overridden ON CYCLE occurs, the overridden ON CYCLE is restored when the calling context is restored, but the time value of the more recent ON CYCLE remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON CYCLE can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON CYCLE priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON CYCLE statement. CALL and GOSUB will return to the next line that would have been executed if the CYCLE event had not been serviced, and the system priority is restored to that which existed before the ON CYCLE branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON CYCLE statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

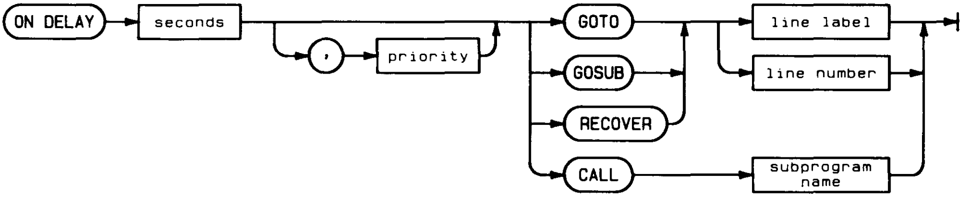
ON CYCLE is disabled by DISABLE and deactivated by OFF CYCLE. If the cycle value is short enough that the computer cannot service it, the interrupt will be lost.

BASIC/UX Specifics

Resolution is 20 milliseconds. A new child process of BASIC/UX is started for the timer.

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken after the specified number of seconds has elapsed.



Item	Description	Range
seconds	numeric expression, rounded to the nearest 0.02 second	0.01 thru 167 772.16
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Examples

```
ON DELAY 10 GOTO Default
ON DELAY 3,2 GOSUB Low_level
```

Semantics

The most recent ON DELAY (or OFF DELAY) definition overrides any previous ON DELAY definition. If the overriding ON DELAY definition occurs in a context different from the one in which the overridden ON DELAY occurs, the overridden ON DELAY is restored when the calling context is restored, but the time value of the more recent ON DELAY remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON DELAY can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON DELAY priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON DELAY statement. CALL and GOSUB will return to the next line that would have been executed if the DELAY event had not been serviced, and the system priority is restored to that which existed before the ON DELAY branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON DELAY statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated

call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

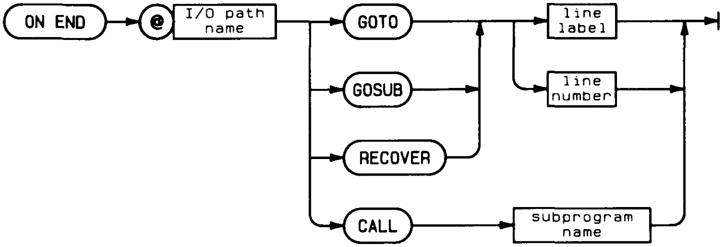
ON DELAY is disabled by DISABLE and deactivated by OFF DELAY.

BASIC/UX Specifics

Resolution is 20 milliseconds. A new child process of BASIC/UX is started for the timer.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement defines and enables an event-initiated branch to be taken when end-of-file is reached on the mass storage file associated with the specified I/O path.



Item	Description	Range
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON END @Source GOTO Next_file
ON END @Dest CALL Expand
```

Semantics

The ON END branch is triggered by any of the following events:

- When the physical end-of-file is encountered.
- When an ENTER statement reads the byte at EOF or beyond.
- When a random access OUTPUT or ENTER requires more than one defined record.
- When a random access OUTPUT is attempted beyond the next available record. (If EOF is the first byte of a record, then that record is the next available record. If EOF is not at the first byte of a record, the following record is the next available record.)

The priority associated with ON END is higher than priority 15. ON TIMEOUT and ON ERROR have the same priority as ON END, and can interrupt an ON END service routine.

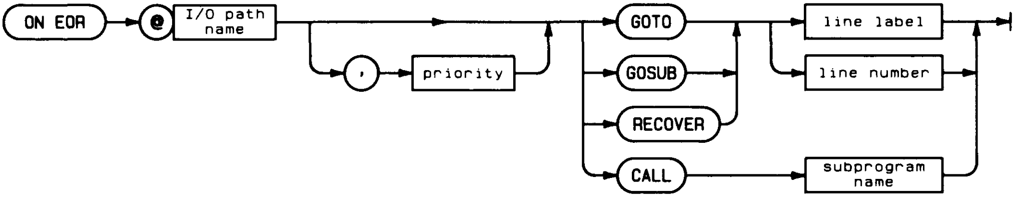
Any specified line label or line number must be in the same context as the ON END statement. CALL and GOSUB will return to the line immediately following the one during which the end-of-file occurred, and the system priority is restored to that which existed before the ON END branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON END statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, if the I/O path name is known in the new context. CALL and RECOVER do not remain active if the context changes as a result of a keyboard-originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram.

The end-of-record error (error 60) or the end-of-file error (error 59) can be trapped by ON ERROR if ON END is not active. ON END is deactivated by OFF END. DISABLE does not affect ON END.

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when an end-of-record is encountered during a TRANSFER.



Item	Description	Range
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON EOR @Gpio GOSUB Gpio_eor
ON EOR @Hpib,9 CALL Eor_sensed
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file or pipe. If the I/O path is assigned to a BUFFER, an error is reported when the ON EOR statement is executed.

If a TRANSFER statement uses an I/O path name which is local to a subprogram and the TRANSFER has not completed by the time the context is exited, returning to the original context will be deferred until the end of the TRANSFER; at that time the ON EOR event cannot be serviced. To ensure that the event will be serviced, a statement that cannot be executed in overlap with the TRANSFER must be executed before the context is exited. A `WAIT FOR EOR @Non_buf` statement is used for this purpose.

End-of-record delimiters are defined by the EOR parameters of the TRANSFER statement (i.e., DELIM, COUNT, or END). An EOR event occurs when any of the specified end-of-record delimiters is encountered during a TRANSFER. The event's occurrence is logged, and the specified branch is taken when system priority permits.

The most recent ON EOR (or OFF EOR) definition for a given I/O path name overrides any previous ON EOR definition. If the overriding ON EOR definition occurs in a context different from the one in which the overridden ON EOR occurs, the overridden ON EOR is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON EOR can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON EOR priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON EOR statement. CALL and GOSUB will return to the next line that would have been executed if the EOR event had not been serviced, and the system priority is restored to that which existed before the ON EOR branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON EOR statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

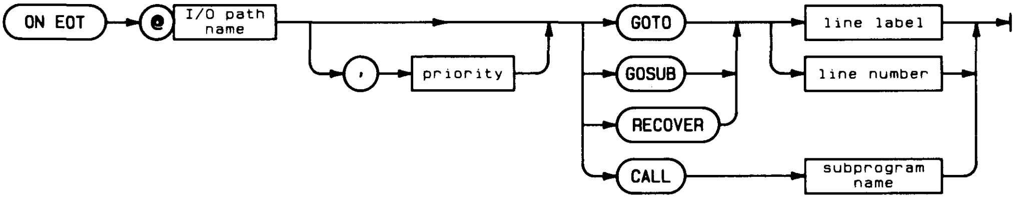
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated

call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EOR is disabled by DISABLE and deactivated by OFF EOR.

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when the last byte is transferred by a TRANSFER statement.



Item	Description	Range
I/O path name	name assigned to a device, a group of devices, or a mass storage file	any valid name
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON EOT @File GOTO Finished
ON EOT @Hpib,5 CALL More
```

Semantics

The I/O path may be assigned either to a device, a group of devices, or to a mass storage file or pipe. If the I/O path is assigned to a BUFFER, an error is reported when the ON EOT statement is executed.

If a TRANSFER statement uses an I/O path name which is local to a subprogram and the TRANSFER has not completed by the time the context is exited, returning to the original context will be deferred until the end of the TRANSFER; at that time the ON EOT event cannot be serviced. To ensure that the event will be serviced, a statement that cannot be executed in overlap with the TRANSFER must be executed before leaving the context. A `WAIT FOR EOT @Non_buf` statement is used for this purpose.

The most recent ON EOT (or OFF EOT) definition for a given path name overrides any previous ON EOT definition. If the overriding ON EOT definition occurs in a context different from the one in which the overridden ON EOT occurs. the overridden ON EOT is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON EOT can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON EOT priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON EOT statement. CALL and GOSUB will return to the next line that would have been executed if the EOT event had not been serviced, and the system priority is restored to that which existed before the ON EOT branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON EOT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

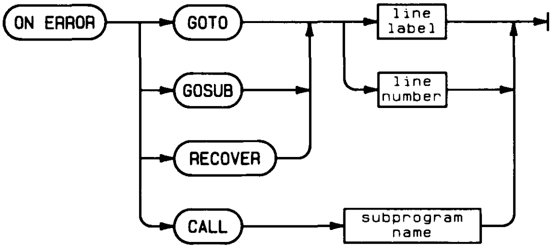
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated

call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EOT is disabled by DISABLE and deactivated by OFF EOT.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch which results from a trappable error. This allows you to write your own error-handling routines.



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

Example Statements

```
ON ERROR GOTO 1200
ON ERROR CALL Report
```

Semantics

The ON ERROR statement has the highest priority of any event-initiated branch. ON ERROR can interrupt any event-initiated service routine.

Any specified line label or line number must be in the same context as the ON ERROR statement. RECOVER forces the program to go directly to the specified line in the context containing the ON ERROR statement.

Returns via RETURN, SUBEXIT, or SUBEND from ON ERROR GOSUB or ON ERROR CALL routines are different from regular GOSUB or CALL returns. When ON ERROR is in effect, the program resumes at the beginning of the line where the error occurred. If the ON ERROR routine did not correct the cause of the error, the error is repeated. This causes an infinite loop between the line in error and the error handling routine. To avoid a retry of the line which caused the error, use ERROR RETURN instead of RETURN or ERROR SUBEXIT instead of SUBEXIT. When execution returns from the ON ERROR routine, system priority is restored to that which existed before the ON ERROR branch was taken.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. In this case, the error is reported to the user, as if ON ERROR had not been executed.

GOSUB and GOTO do not remain active when the context changes to a subprogram. If an error occurs, the error is reported to the user, as if ON ERROR had not been executed.

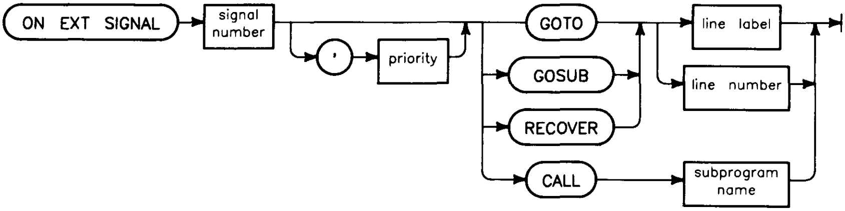
If an execution error occurs while servicing an ON ERROR CALL or ON ERROR GOSUB, program execution stops. If an execution error occurs while servicing an ON ERROR GOTO or ON ERROR RECOVER routine, an infinite loop can occur between the line in error and the GOTO or RECOVER routine.

If an ON ERROR routine cannot be serviced because inadequate memory is available for the computer, the original error is reported and program execution pauses at that point.

ON ERROR is deactivated by OFF ERROR. DISABLE does not affect ON ERROR.

Supported On	UX
Option Required	n/a
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement defines an event-initiated branch to be taken when a system generated signal is received.



Item	Description	Range
signal number	numeric expression, rounded to integer	(see below)
priority	numeric expression, rounded to integer (Default = 1)	1 thru 15
line label	name of a program line	any valid name
line number	integer const identifying a program line	1 thru 32766
subprogram name	name of a SUB or CSUB	any valid name

Example Statements

```
ON EXT SIGNAL 4 GOTO 10
ON EXT SIGNAL Sigusr2,12 GOSUB Fred
ON EXT SIGNAL Sigterm,15 CALL Terminate
```

Semantics

The ON EXT SIGNAL statement specifies a new action to be taken when a system generated signal is received by BASIC. If an ON EXT SIGNAL statement is not specified, then a default system action is be taken. The action for a specific EXT SIGNAL is specified in the table below. The two possible actions that can be taken are:

Exit   BASIC is immediately, but gracefully exited.

Error   An error [number to be determined] is generated at the next end-of-line.

All ON EXT SIGNAL actions take place at end-of-line except the default action to exit, which takes effect immediately upon receipt.

BASIC does not allow all system signals to be caught by users. The table below specifies all system signals, and indicates which can be specified in the EXT SIGNAL statements. All other values cause an error. This table also specifies the default EXT SIGNAL handling action which takes place in the absence of an ON EXT SIGNAL, or after an OFF EXT SIGNAL.

Signal Number	Signal Name	Valid Signal	Default Action	Comment
1	SIGHUP	yes	exit	hangup (lost connection)
2	SIGINT	no	-	BASIC "CLR-I/O" signal
3	SIGQUIT	no	-	BASIC "RESET" signal
4	SIGILL	no	-	illegal instruction
5	SIGTRAP	no	-	BASIC debugging signal
6	SIGIOT	yes	error	software generated (abort)
7	SIGEMT	yes	error	software generated
8	SIGFPE	no	-	floating point execution used internally by BASIC
9	SIGKILL	no	-	not catchable by anyone
10	SIGBUS	no	-	hardware bus error
11	SIGSEGV	no	-	segmentation violation
12	SIGSYS	yes	error	bad argument to system call
13	SIGPIPE	no	-	write on pipe with no reader
14	SIGALRM	yes	error	system alarm clock (used by BASIC)
15	SIGTERM	yes	exit	software termination signal
16	SIGUSR1	no	-	used by BASIC for communications
17	SIGUSR2	yes	error	user defined signal
18	SIGCLD	no	-	used by BASIC
19	SIGPWR	no	-	powerfail; never reaches user
20	SIGVTALRM	yes	error	virtual timer alarm

Signal Number	Signal Name	Valid Signal	Default Action	Comment
21	SIGPROF	yes	error	profiling timer alarm
22	SIGIO	no	-	used by BASIC
23	SIGWINDOW	no	-	window/mouse signal
24	SIGSTOP	no	-	not supported on S300
25	SIGTSTP	no	-	not supported on S300
26	SIGCONT	no	-	not supported on S300
27	SIGTTIN	no	-	not supported on S300
28	SIGTTOU	no	-	not supported on S300
29	SIGURG	no	-	urgent condition in I/O
30	-	no	-	not defined for HP-UX
31	-	no	-	not defined for HP-UX
32	-	no	-	not defined for HP-UX

EXT SIGNALS default to and remain enabled unless explicitly disabled with the DISABLE EXT SIGNAL statement.

The priority of an EXT SIGNAL can be specified in the ON EXT SIGNAL statement, with the highest priority represented by 15. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON EXT SIGNAL can interrupt service routines of other event-initiated branches which have user-definable priorities, if the ON EXT SIGNAL priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON EXT SIGNAL statement. CALL and GOSUB return to the next line that would have been executed if the EXT SIGNAL event had not been serviced, and the system priority is restored to that which existed before the ON EXT SIGNAL branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON EXT SIGNAL statement. When recover forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EXT SIGNAL is disabled by DISABLE EXT SIGNAL or DISABLE and deactivated by OFF EXT SIGNAL.

The current state of the system signal handling can be determined through the STATUS statement. EXT SIGNALS use the pseudo-select code 33 for providing status information. For each EXT SIGNAL, a status register exists with the same number, and providing the following information:

Status Number	Comment
-1	signal not catchable by user
0	signal disabled
1	signal enabled

Thus to determine the state of the SIGTERM (15) signal,

```
STATUS 33,15;A
```

When an EXT SIGNAL is enabled, and there is no ON EXT SIGNAL setup for it and the default action is an error , a program error is generated if a program is running, or if in a keyboard command (including EXECUTE). If a program is running, an ON ERROR statement can catch the error.

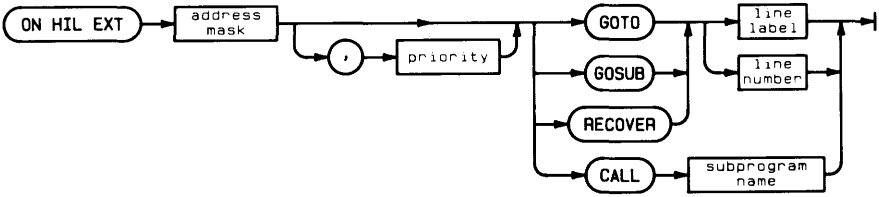
When BASIC is idle (not running a program and not executing a keyboard command) all EXT SIGNALS except SIGHUP and SIGTERM are ignored. SIGHUP and SIGTERM exit if they are enabled.

Note that all EXT SIGNALs default to being enabled.

ON HIL EXT

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement enables an end-of-line interrupt in response to receiving data from HIL devices whose poll records are not otherwise being processed by the BASIC system.



Item	Description/Default	Range Restrictions
address mask	the sum of 2 raised to the power of each of the addresses of the desired devices; Default = 254	any even number from 2 to 254
priority	numeric expression, rounded to a integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statement

```
ON HIL EXT 8 GOSUB Ser_routine
ON HIL EXT Mask,Priority CALL Sub_prog
ON HIL EXT 2,3 GOTO Label_1
```

Semantics

The *address mask* provides the capability of being able to enable polling of several devices using the same ON HIL EXT statement. This mask is obtained by raising 2 to the power of each of the addresses of desired devices, and adding these values. Suppose you want to create a mask which would allow interrupts from HP-HIL devices at addresses 1 and 3. You would take 2 and raise it to the first power and add this result to 2 raised to the third power; the final result is a mask value of 10. This indicates that end-of-line interrupts can be received from HP-HIL devices at addresses 1 and 3 in the HP-HIL link. Note that the default mask is 254 (all devices in the link).

While interrupts are enabled, poll records are accumulated and returned via the HILBUF$ function. 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$.

HP-HIL devices which can use the ON HIL EXT statement are those whose poll records are not being processed for another purpose by the BASIC system or the Keyboard controller. These devices are grouped into two 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). Note that both digitizers return data too fast to be processed using the HILBUF$ function; therefore, it is best to use the GRAPHICS INPUT IS statement with these devices along with the READ LOCATOR or DIGITIZE statement.
- 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).

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.
- Current GRAPHICS INPUT devices.
- All system Keyboards (includes HP 98203C as well as HP 46020/21A). Their poll records are processed by the Keyboard controller and the keycodes returned to BASIC via a different interface.

The *priority* can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON HIL EXT can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON HIL EXT priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON HIL EXT statement. CALL and GOSUB will return to the next line that would have been executed if the HIL EXT event had not been serviced, and the system priority is restored to that which existed before the ON HIL EXT branch was taken. RECOVER forces the program to go directly to the specified line in the context containing the ON HIL EXT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

The most recent ON HIL EXT (or OFF HIL EXT) overrides any previous ON HIL EXT definition. If the overriding ON HIL EXT occurs in another context (such as in a SUB subprogram), then the overridden ON HIL EXT branch is restored when the calling context is restored. (See below for restrictions.)

ON HIL EXT is deactivated by OFF HIL EXT.

The ON HIL EXT statement behaves like the ON CDIAL and ON KNOB statements:

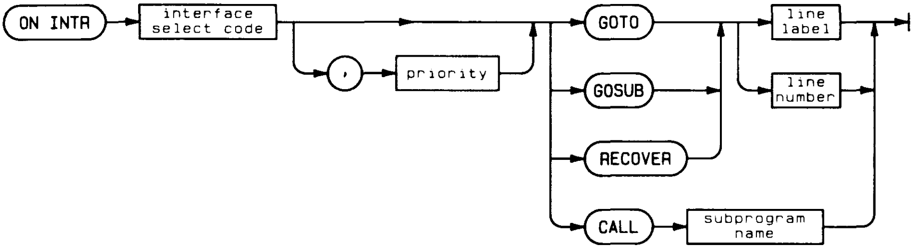
- When ON HIL EXT is executed in a SUB context and program control exits that context, the data from the enabled devices will continue to be accumulated (and can be read by HILBUF$—unless lost due to buffer overflow). No interrupts occur if there is no ON HIL EXT active in the current context.
- Conversely, if an ON HIL EXT has been executed in a context and the OFF HIL EXT is executed in a called context, upon returning to the calling context, the data **is not** accumulated for HILBUF$ and no interrupts will be initiated.

If ON HIL EXT is executed in a context with one mask value, and then another ON HIL EXT is executed in a called context with a different mask value, the former mask value **is not** restored on return to the calling context. This behavior is similar to the time parameters of ON CYCLE and ON DELAY.

ON INTR

Supported On	WS,UX
Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines an event-initiated branch to be taken when an interface card generates an interrupt. The interrupts must be explicitly enabled with an ENABLE INTR statement.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	5, 7 thru 31
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON INTR 7 GOSUB 500
ON INTR Isc,4 CALL Service
```

Semantics

The occurrence of an interrupt performs an implicit DISABLE INTR for the interface. An ENABLE INTR must be performed to re-enable the interface for subsequent event-initiated branches. Another ON INTR is not required, nor must the mask for ENABLE INTR be redefined.

The priority can be specified, with highest priority represented by l5. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON INTR can interrupt service routines of other event-initiated branches which have user-definable priorities, if the ON INTR priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON INTR statement. CALL and GOSUB will return to the next line that would have been executed if the INTR event had not been serviced, and the system priority is restored to that which existed before the ON INTR branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON INTR statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

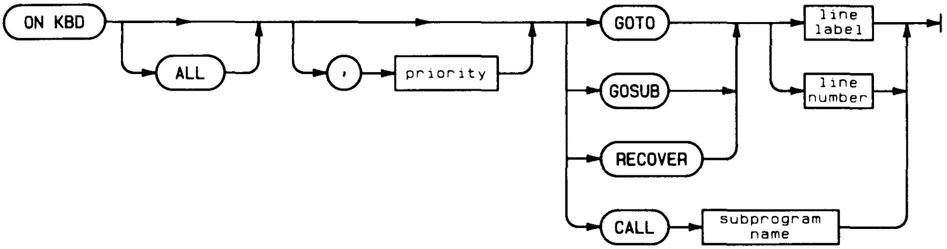
ON INTR is disabled by DISABLE INTR or DISABLE and deactivated by OFF INTR.

ON INTR and OFF INTR statements may be executed for **any** I/O card in the machine. It is not necessary to have a driver for the card.

ON KBD

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when a key is pressed.



Item	Description	Range
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KBD GOSUB 770
ON KBD,9 CALL Get_key
```

Semantics

Specifying the secondary keyword ALL causes all keys except RESET, SHIFT, and CTRL to be trapped. When ALL is omitted, the untrapped keys are those just mentioned, the softkeys, PAUSE, STOP, CLR I/O, Break, System, User, Menu, and Shift Menu. When not trapped, these keys perform their normal functions. When the softkeys are trapped, ON KBD branching overrides any ON KEY branching.

A keystroke triggers a keyboard interrupt and initiates a branch to the specified routine when priority allows. If keystrokes occur while branching is held off by priority, the keystrokes are stored in a special buffer. When keystrokes are in the buffer, branching will occur when priority allows. This buffer is read and cleared by the KBD$ function (see the KBD$ entry).

Knob rotation will generate ON KBD interrupts unless an ON KNOB statement has been executed. Clockwise rotation of the knob produces right-arrow keystrokes; counterclockwise rotation produces left-arrow keystokes. If the SHIFT key is pressed while turning the knob, then clockwise rotation of the knob produces up-arrow keystrokes; counterclockwise rotation produces down-arrow key strokes. Since one rotation of the knob is equivalent to 20 keystrokes (more with HP-HIL knobs), keyboard buffer overflow may occur if the BASIC service routine does not process the keys rapidly.

Live keyboard, editing, and display control functions are suspended during ON KBD. To restore a key's normal function the keystroke may be OUTPUT to select code 2.

The most recent ON KBD (or OFF KBD) definition overrides any previous ON KBD definition. If the overriding ON KBD definition occurs in a context different from the one in which the overridden ON KBD occurs, the overridden ON KBD is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON KBD can interrupt sevice routines of other event-initiated branches with user-definable priorities, if the ON KBD priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KBD statement. CALL and GOSUB will return to the next line that would have been executed if the KBD event had not been serviced, and the system priority is restored to that which existed before the ON KBD branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KBD statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

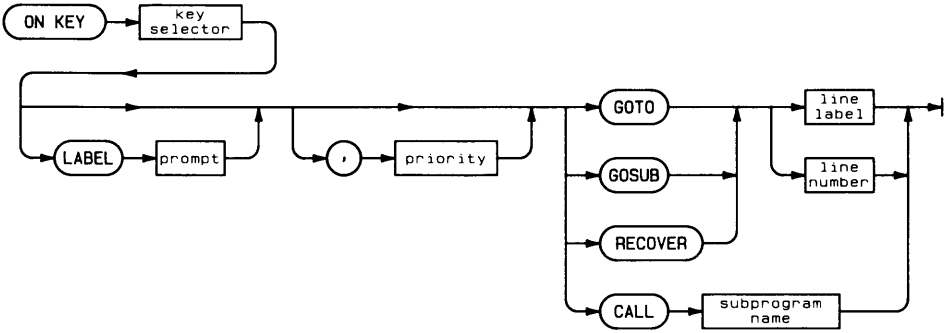
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON KBD is disabled by DISABLE, deactivated by OFF KBD, and temporarily deactivated when the program is executing LINPUT, INPUT, or ENTER KBD.

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

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when a softkey is pressed.



Item	Description	Range
key selector	numeric expression, rounded to an integer	0 thru 23

prompt	string expression;
	Default = no label
		—

priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KEY 0 GOTO 150
ON KEY 5 LABEL "Print",3 GOSUB Report
```

Semantics

The most recently executed ON KEY (or OFF KEY) definition for a particular softkey overrides any previous key definition. If the overriding ON KEY definition occurs in a context different from the one in which the overridden ON KEY occurs, the overridden ON KEY is restored when the calling context is restored.

Labels appear in the two bottom lines of the CRT. The label of any key is bound to the current ON KEY definition. Therefore, when a definition is changed or restored, the label changes accordingly. If no label is specified, that label field is blank. Refer to the *BASIC Programming Techniques* manual for a discussion of these labels.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). On KEY can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON KEY priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

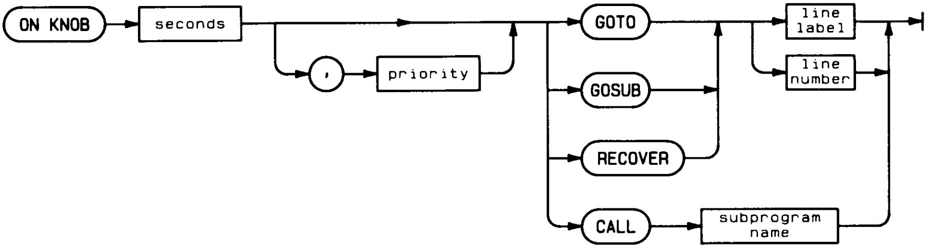
Any specified line label or line number must be in the same context as the ON KEY statement. CALL and GOSUB will return to the next line that would have been executed if the KEY event had not been serviced, and the system priority is restored to that which existed before the ON KEY branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KEY statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON KEY is disabled by DISABLE, deactivated by OFF KEY, and temporarily deactivated when the program is paused or executing LINPUT, INPUT, or ENTER KBD.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement defines and enables an event-initiated branch to be taken when the knob is turned.



Item	Description	Range
seconds	numeric expression, rounded to the nearest 0.01 second	0.01 thru 2.55
priority	numeric expression, rounded to an integer; Default=1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON KNOB .1 GOSUB 250
ON KNOB .333,Priority CALL Pulses
```

Semantics

Turning the knob (cursor wheel) generates pulses. After ON KNOB is activated (or reactivated), the first pulse received starts a sampling interval. The "seconds" parameter establishes the length of that sampling interval. At the end of the sampling interval, the ON KNOB branch is taken if the net number of pulses received during the interval is not zero and priority permits. The KNOBX and KNOBY functions can be used to determine the number of pulses received during the interval. If the ON KNOB branch is held off for any reason, the KNOBX and KNOBY functions accumulate the pulses (see KNOBX and KNOBY).

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON KNOB can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON KNOB priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KNOB statement. CALL and GOSUB will return to the next line that would have been executed if the KNOB event had not been serviced, and the system priority is restored to that which existed before the ON KNOB branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON KNOB statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

The most recent ON KNOB (or OFF KNOB) definition overrides any previous ON KNOB definition. If the overriding ON KNOB definition occurs in a context different from the one in which the overridden ON KNOB occurs, the overridden ON KNOB is restored when the calling context is restored, but the "seconds" parameter of the more recent ON KNOB remains in effect. (See below for restrictions.)

ON KNOB is disabled by DISABLE and deactivated by OFF KNOB.

You can use an HP-HIL relative pointing device, such as a mouse or knob, if the KBD binary is loaded.

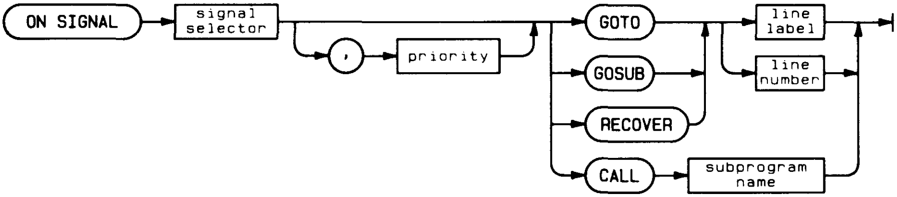
The ON KNOB statement behaves like the ON CDIAL and ON HIL EXT statements:

- When ON KNOB is executed in a SUB context and program control exits that context, the pulses from control dials will continue to be accumulated (and can be read by KNOBX and KNOBY). No interrupts occur if there is no ON KNOB active in the current context.
- Conversely, if an ON KNOB has been executed in a context and then OFF KNOB is executed in a called context, then upon returning to the calling context the pulses will be routed to the BASIC system (instead of the KNOBX and KNOBY functions) and no interrupts will be initiated.

ON SIGNAL

Supported On	WS,UX
Option Required	IO
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when a SIGNAL statement with the same signal selector is executed.



Item	Description	Range
signal selector	numeric expression, rounded to an integer	0 thru 15
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
suprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON SIGNAL 5 GOSUB 550
ON SIGNAL Bailout,15 RECOVER Bail_here
```

Semantics

The most recent ON SIGNAL (or OFF SIGNAL) definition for a given signal selector overrides any previous ON SIGNAL definition. If the overriding ON SIGNAL definition occurs in a context different from the one in which the overridden ON SIGNAL occurs, the overridden ON SIGNAL is restored when the calling context is restored.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON SIGNAL can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON SIGNAL priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON SIGNAL statement. CALL and GOSUB will return to the next line that would have been executed if the SIGNAL event had not been serviced, and the system priority is restored to that which existed before the ON SIGNAL branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON SIGNAL statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

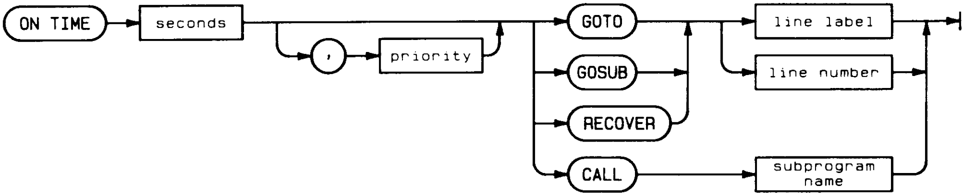
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON SIGNAL is disabled by DISABLE and deactivated by OFF SIGNAL.

ON TIME

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

This statement defines and enables an event-initiated branch to be taken when the real-time clock reaches a specified time.



Item	Description	Range
seconds	numeric expression, rounded to the nearest 0.02 second	0 thru 86 399.99
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON TIME 3600*8 GOTO Work
ON TIME (TIMEDATE+3600) MOD 86400 CALL One_hour
```

Semantics

The most recent ON TIME (or OFF TIME) definition overrides any previous ON TIME definition. If the overriding ON TIME definition occurs in a context different from the one in which the overridden ON TIME occurs, the overridden ON TIME is restored when the calling context is restored, but the time value of the more recent ON TIME remains in effect.

The priority can be specified, with the highest priority represented by 15. The highest user-defined priority (15) is less than the priority for ON ERROR, ON END, and ON TIMEOUT (whose priorities are not user-definable). ON TIME can interrupt service routines of other event-initiated branches with user-definable priorities, if the ON TIME priority is higher than the priority of the service routine (the current system priority). CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

CALL and GOSUB will return to the next line that would have been executed if the TIME event had not been serviced, and the system priority is restored to that which existed before the ON TIME branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON TIME statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

Any specified line label or line number must be in the same context as the ON TIME statement. CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

Unlike ON CYCLE, an ON TIME statement requires an exact match between the clock and the time specified in the defining statement. If the event was missed and not logged, re-executing the ON TIME statement will not result in a branch being taken.

ON TIME is disabled by DISABLE and deactivated by OFF TIME.

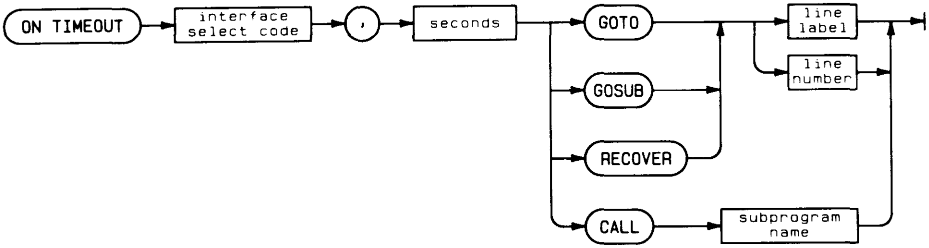
BASIC/UX Specifics

Resolution is 20 milliseconds. A new child process of BASIC/UX is started for the timer.

ON TIMEOUT

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement defines and enables an event-initiated branch to be taken when an I/O timeout occurs on the specified interface.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	7 thru 31
seconds	numeric expression, rounded to the nearest 0.001 second	0.001 thru 32.767
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB or CSUB subprogram	any valid name

Example Statements

```
ON TIMEOUT, 2.554 GOTO 770
ON TIMEOUT Printer,Time GOSUB Message
```

Semantics

There is no default system timeout. If ON TIMEOUT is not in effect for an interface, a device can cause the program to wait forever.

The specified branch occurs if an input or output is active on the interface and the interface has not responded within the number of seconds specified. The computer waits at least the specified time before generating an interrupt; however, it may wait up to an additional 25% of the specified time.

Timeouts apply to ENTER and OUTPUT statements, and operations involving the PRINTER IS, PRINTALL IS, and PLOTTER IS devices when they are external. Timeouts do not apply to CONTROL, STATUS, READIO, WRITEIO, CRT alpha or graphics I/O, real time clock I/O, keyboard I/O, or mass storage operations.

The priority associated with ON TIMEOUT is higher than priority 15. ON END and ON ERROR have the same priority as ON TIMEOUT, and can interrupt an ON TIMEOUT service routine.

Any specified line label or line number must be in the same context as the ON TIMEOUT statement. CALL and GOSUB will return to the line immediately following the one during which the timeout occurred, and the system priority is restored to that which existed before the ON TIMEOUT branch was taken. RECOVER forces the program to go directly to the specified line in the context containing that ON TIMEOUT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram. The TIMEOUT event does remain active. Unlike other ON events, TIMEOUTs are never logged, they always cause an immediate action. If a TIMEOUT occurs when the ON TIMEOUT branch cannot be taken, an error 168 is generated. This can be trapped with ON ERROR. The functions ERRN and ERRDs are set **only** when the error is generated. They are not set when the ON TIMEOUT branch can be taken.

ON TIMEOUT is deactivated by OFF TIMEOUT. DISABLE does not affect ON TIMEOUT.

ON TIMEOUT with SRM Interfaces

With SRM, ON TIMEOUT defines and enables a branch resulting from an I/O timeout on the specified SRM interface. Although ON TIMEOUT is supported on SRM, **its use should be avoided** because the asynchronous nature of the SRM system does not allow predictable results.

A TIMEOUT occurring during statements such as RE-SAVE and RE-STORE may leave a temporary file on the mass storage device. The file's name is a 10-character identifier (the first character is an alpha character, the rest are digits) derived from the value of the workstation's real-time clock when the TIMEOUT occurred. You may wish to check the contents of any such file before purging.

BASIC/UX Specifics

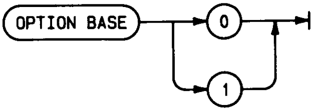
If the interface is a MUX, the interface select code must be a device selector with channel number included. For example,

- `ON TIMEOUT 16` gives an error.
- `ON TIMEOUT 1600` works.

OPTION BASE

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	No

This statement specifies the default lower bound of arrays.



Example Statements

```
OPTION BASE 0
OPTION BASE 1
```

Semantics

This statement can occur only once in each context. If used, OPTION BASE must precede any explicit variable declarations in a context. Since arrays are passed to subprograms by reference, they maintain their original lower bound, even if the new context has a different OPTION BASE. Any context that does not contain an OPTION BASE statement assumes default lower bounds of zero.

The OPTION BASE value is determined at prerun, and is used with all arrays declared without explicit lower bounds in COM, DIM, INTEGER, and REAL statements as well as with all implicitly dimensioned arrays. OPTION BASE is also used at runtime for any arrays declared without lower bounds in ALLOCATE.

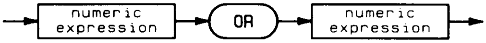
OPTIONAL

See the DEF FN and SUB statements.

OR

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

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



Example Statements

```
X=Y OR Z
IF File_type OR Device THEN Process
```

Semantics

An expression which evaluates to a non-zero value is treated as a logical 1. An expression must evaluate to zero to be treated as a logical 0.

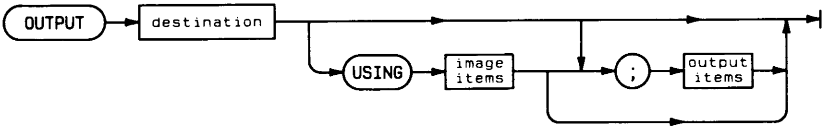
The truth table is:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

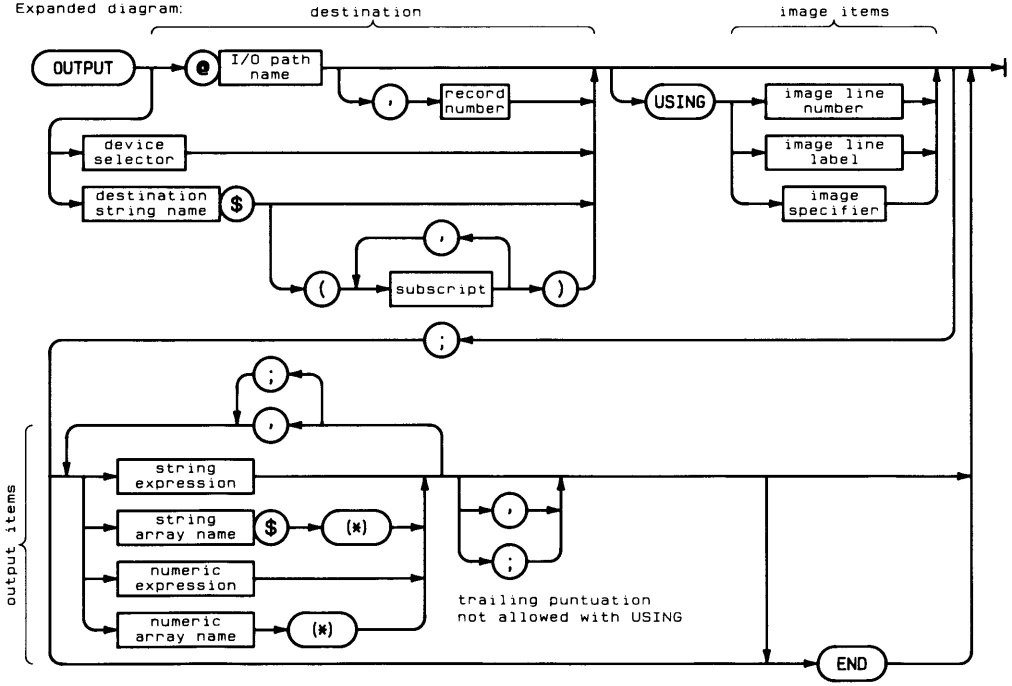
OUTPUT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

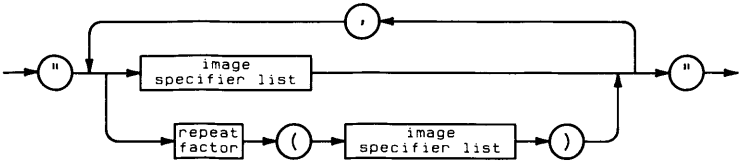
This statement outputs items to the specified destination.



Expanded diagram:

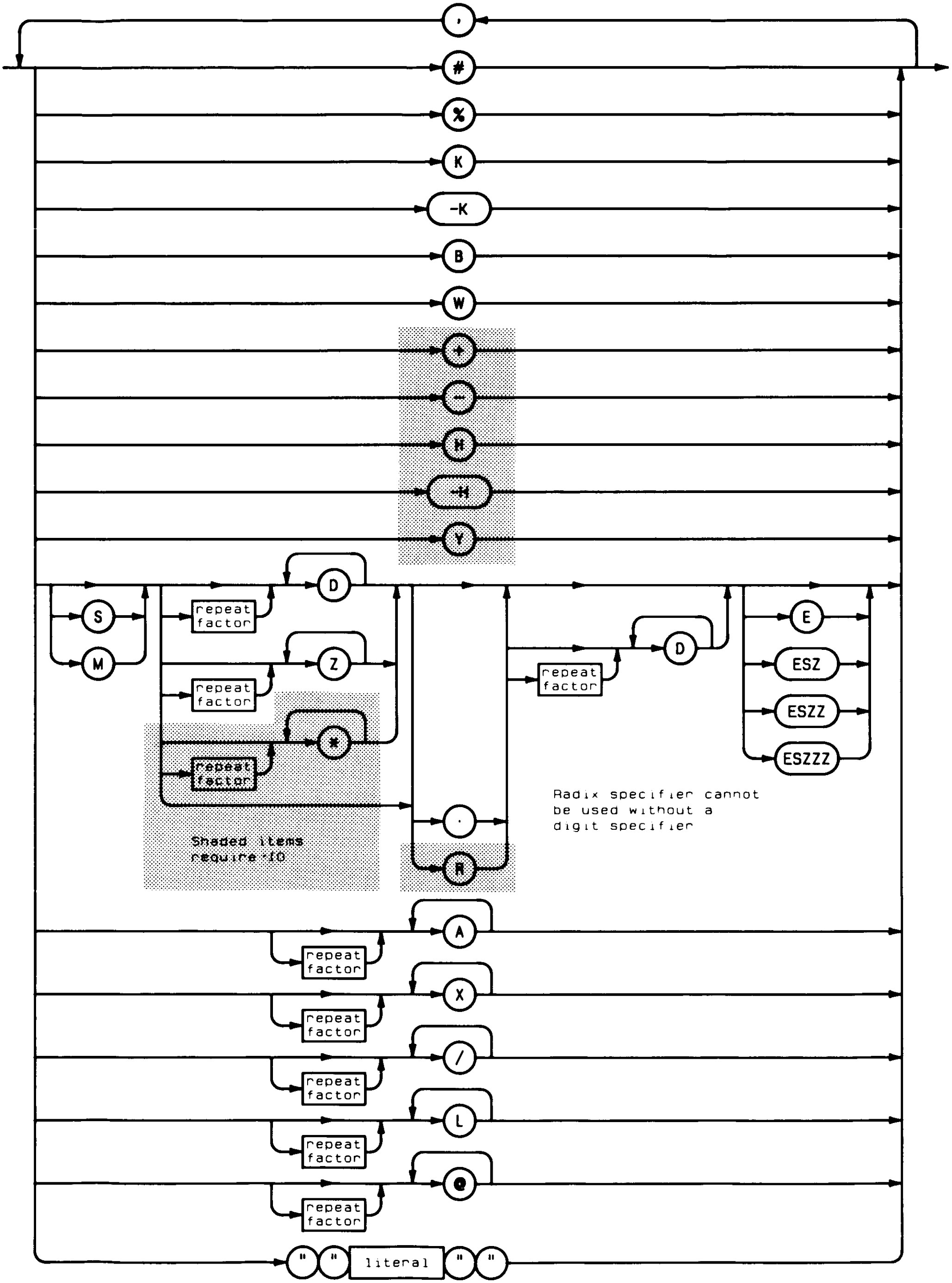


literal form of image specifier



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

image specifier list



Example Statements

```
OUTPUT 701;Number,String$;
OUTPUT @File;Array(*),END
OUTPUT @Rand,5 USING Fmt1;Item(5)
OUTPUT 12 USING "#,6A";B$[2;6]
OUTPUT @Printer;Rank;Id;Name$
```

Semantics

Standard Numeric Format

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

COMPLEX numbers are treated like two REAL numbers. The separator following the item is also used as the separator between the real and imaginary parts.

Arrays

Entire arrays may be output by using the asterisk specifier. Each element in an array is treated as an item by the OUTPUT statement, as if the items were listed separately, separated by the punctuation following the array specifier. If no punctation follows the array specifier, a comma is assumed. The array is output in row major order (rightmost subscript varies fastest). COMPLEX arrays are treated as if they were REAL arrays with twice as many elements.

Files as Destination

If an I/O path has been assigned to a file, the file may be written to with OUTPUT statements. The file must be an ASCII, BDAT, or HP-UX file. The attributes specified in the ASSIGN statement are used if the file is a BDAT or HP-UX file (ASCII files are always assigned a special case of the FORMAT ON attribute).

Serial access is available for ASCII, BDAT, and HP-UX files. Random access is available for BDAT and HP-UX files. The end-of-file marker (EOF) and the file pointer are important to both serial and random access. The file pointer is set to the beginning of the file when the file is opened by an ASSIGN. It is updated by OUTPUT operations so that it always points to the next byte to be written.

The EOF pointer is read from the media when the file is opened by an ASSIGN. On a newly created file, EOF is set to the beginning of the file. After each OUTPUT operation, the EOF pointer in the I/O path table is updated to the maximum of the file pointer or the previous EOF value. The EOF pointer on the volume is updated at the following times:

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

Random access uses the record number parameter to write items to a specific location in a file. The OUTPUT begins at the start of the specified record and must fit into one record. The record specified cannot be beyond the record containing the EOF, if EOF is at the first byte of a record. The record specified can be one record beyond the record containing the EOF, if EOF is not at the first byte of a record. Random access is always allowed to records preceding the EOF record. If you wish to write randomly to a newly created file, either use a CONTROL statement to position the EOF in the last record, or write some "dummy" data into every record.

When data is written to an ASCII file, each item is sent as an ASCII representation with a 2-byte length header. You cannot use OUTPUT with USING to ASCII files; see the following section, "OUTPUT with USING" for details.

Data sent to a BDAT or HP-UX file is sent in internal format if FORMAT OFF is currently assigned to the I/O path (this is the default FORMAT attribute for these file types), and is sent as ASCII characters if FORMAT ON has been explicitly assigned. (See "Devices as Destination" for a description of these formats.)

OUTPUT to HFS Files

You must have W (write) permission on an HFS file, as well as X (search) permission on all superior directories, to output data to the file. If you do not have these permissions, error 183 is reported.

HFS files are extensible. If the data output to the file with this statement would overflow the file's space allocation, the file is extended. The BASIC system allocates the additional space needed to store the data being output, provided the disc contains enough unused storage space.

OUTPUT to SRM Files

You must have W (write) access capability on an SRM file, as well as R (read) capability on all superior directories, to output data to the file. If this capability is not public or if a password protecting this capability was not used at the time the file was assigned an I/O path name (with ASSIGN), error 62 is reported.

SRM files are extensible. If the data output to the file with this statement would overflow the file's space allocation, the file is extended. The BASIC system allocates an additional "extent size" amount of space, provided the disc contains enough unused storage space; see one of the CREATE statements for a description of "extent size".

Devices as Destination

An I/O path or a device selector may be used to direct OUTPUT to a device. If a device selector is used, the default system attributes are used (see ASSIGN). If an I/O path is used, the ASSIGN statement used to associate the I/O path with the device also determines the attributes used. If multiple listeners were specified in the ASSIGN, the OUTPUT is directed to all of them. If FORMAT ON is the current attribute, the items are sent in ASCII. Items followed by a semicolon are sent with nothing following them. Numeric items followed by a comma are sent with a comma following them. String items followed by a comma are sent with a CR/LF following them. If the last item in the OUTPUT statement has no punctuation following it, the current end-of-line (EOL) sequence is sent after it. Trailing punctuation eliminates the automatic EOL.

If FORMAT OFF is the current attribute, items are sent to the device in internal format. Punctuation following items has no effect on the OUTPUT. Two bytes are sent for each INTEGER, eight bytes for each REAL, and sixteen bytes for each COMPLEX value. Each string output consists of a four byte header containing the length of the string, followed by the actual string characters. If the number of characters is odd, an additional byte containing a blank is sent after the last character.

CRT as Destination

If the device selector is 1, the OUTPUT is directed to the CRT. OUTPUT 1 and PRINT differ in their treatment of separators and print fields. The OUTPUT format is described under "Devices as Destination." See the PRINT keyword for a discussion of that format. OUTPUT 1 USING and PRINT USING to the CRT produce similar actions.

Keyboard as Destination

Outputs to device selector 2 may be used to simulate keystrokes. ASCII characters can be sent directly (i.e. "hello"). Non-ASCII keys (such as EXECUTE) are simulated by a two-byte sequence. The first byte is CHR$(255), and the second byte can be found in the "Second Byte of Non-ASCII Key Sequences" table in the back of this book.

When simulating keystrokes, unwanted characters (such as the EOL sequence) can be avoided with an image specifier (such as "#,B" or "#,K"). See "OUTPUT with USING."

Strings as Destination

If a string is used for the destination, the string is treated similarly to a file. However, there is no file pointer; each OUTPUT begins at the beginning of the string, and writes serially within the string.

Buffers as Destination (Requires TRANS)

When the destination is an I/O path name assigned to a buffer, data is placed in the buffer beginning at the location indicated by the buffer's fill pointer. As data is sent, the current number-of-bytes

register and fill pointer are adjusted accordingly. Encountering the empty pointer (buffer full) produces an error unless a continuous outbound TRANSFER is emptying the buffer. In this case, the OUTPUT will wait until there is more room in the buffer for data.

If an I/O path is currently being used in an inbound TRANSFER, and an OUTPUT statement uses it as a destination, execution of the OUTPUT is deferred until the completion of the TRANSFER. An OUTPUT can be concurrent with an outbound TRANSFER only if the destination is the I/O path assigned to the buffer.

An OUTPUT to a string variable that is also a buffer will not update the buffer's pointers and will probably corrupt the data in the buffer.

Pipes as Destination (BASIC/UX only)

If an I/O path has been assigned to a pipe, the pipe may be written to with OUTPUT statements. The attributes specified in the ASSIGN statement are used. Data is sent in internal format if FORMAT OFF is currently assigned to the I/O path, and is sent as ASCII characters if FORMAT is currently assigned (this is the default FORMAT attribute). (See "Devices as Destination" for a description of these formats.)

Using END with Devices

The secondary keyword END may be specified following the last item in an OUTPUT statement. The result, when USING is not specified, is to suppress the EOL (End-of-Line) sequence that would otherwise be output after the last byte of the last item. If a comma is used to separate the last item from the END keyword, the corresponding item terminator is output (CR/LF for string items or comma for numeric items).

With HP-IB interfaces, END specifies an EOI signal to be sent with the last data byte of the last item. However, if no data is sent from the last output item, EOI is not sent. With Data Communications interfaces, END specifies an end-of-data indication to be sent with the last byte of the last output item.

OUTPUT With USING

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

The processing of image specifiers stops when 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 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 right-most characters are lost. If it is shorter than the specifier, trailing blanks are used to fill out the field.

OUTPUT with USING cannot be used with output to ASCII files. Instead, direct the OUTPUT with USING to a string variable, and then OUTPUT this variable to the file. For instance, `OUTPUT String$ USING "5A,X,6D.D";Chars$,Number` and then `OUTPUT @File;String$`.

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

Image Specifier	**Meaning**
K	Compact field. Outputs a number or string in standard form with no leading or trailing blanks.
−K	Same as K.
H	Similar to K, except the number is output using the European number format (comma radix). (Requires IO)
−H	Same as H. (Requires IO)
S	Outputs the number's sign (+ or −).
M	Outputs the number's sign if negative, a blank if positive.
D	Outputs one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is output, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are output.
*	Like D, except that asterisks are output instead of leading zeros. (Requires IO)
.	Outputs a decimal-point radix indicator.
R	Outputs a comma radix indicator (European radix). (Requires IO)
E	Outputs an E, a sign, and a two-digit exponent.
ESZ	Outputs an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Outputs an E, a sign, and a three-digit exponent.
A	Outputs a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Outputs a blank.
literal	Outputs the characters contained in the literal.
B	Outputs the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.

Image Specifier	Meaning
W	Outputs a 16-bit word as a two's-complement integer. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is sent; if it is less than −32 768, then −32 768 is sent. If either an I/O path name with the BYTE attribute or a device selector is used to access an 8-bit interface, two bytes will be output; the most-significant byte is sent first. If an I/O path name with the BYTE attribute is used to access a 16-bit interface, the BYTE attribute is overridden, and one word is output in a single operation. If an I/O path name with the WORD attribute is used to access a 16-bit interface, a null pad byte is output whenever necessary to achieve alignment on a word boundary. If the destination is a BDAT file, string variable, or buffer, the BYTE or WORD attribute is ignored and all data are sent as bytes; however, pad byte(s) will be output when necessary to achieve alignment on a word boundary. The pad character may be changed by using the CONVERT attribute; see the ASSIGN statement for further information.
Y	Like W, except that no pad bytes are output to achieve word alignment. If an I/O path with the BYTE attribute is used to access a 16-bit interface, the BYTE attribute is not overridden (as with the W specifier above). (Requires IO)
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence following the last output item.
%	Ignored in OUTPUT images.
+	Changes the automatic EOL sequence that normally follows the last output item to a single carriage-return. (Requires IO)
−	Changes the automatic EOL sequence that normally follows the last output item to a single line-feed. (Requires IO)
/	Outputs a carriage-return and a line-feed.
L	Outputs the current end-of-line (EOL) sequence. The default EOL characters are CR and LF; see ASSIGN for information on re-defining the EOL sequence. If the destination is an I/O path name with the WORD attribute, a pad byte may be sent after the EOL characters to achieve word alignment.
@	Outputs a form-feed.

END with OUTPUT...USING

Using the optional secondary keyword END in an OUTPUT...USING statement produces results which differ from those in an OUTPUT statement without USING. Instead of always suppressing the EOL sequence, the END keyword only suppresses the EOL sequence when no data is output from the last output item. Thus, the # image specifier generally controls the suppression of the otherwise automatic EOL sequence.

With HP-IB interfaces, END specifies an EOI signal to be sent with the last byte output. However, no EOI is sent if no data is sent from the last output item or the EOL sequence is suppressed. With Data Communications interfaces, END specifies an end-of-data indication to be sent at the same times an EOI would be sent on HP-IB interfaces.

BASIC/UX Specifics

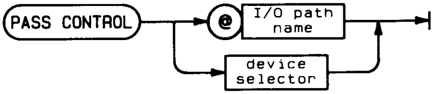
You can specify a window number or unnamed pipe as the output destination to OUTPUT.

PARITY

See the ASSIGN statement.

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used to pass the capability of Active Controller to a specified HP-IB device.



Item	Description	Range
I/O path name	name assigned to an HP-IB device	any valid name
device selector	numeric expression, rounded to an integer	must contain primary address (see Glossary)

Example Statements

```
PASS CONTROL 719
PASS CONTROL @Controller_19
```

Semantics

Executing this statement first addresses the specified device to talk and then sends the Take Control message (TCT), after which Attention is placed in the False state. The computer then assumes the role of a bus device (a non-active controller).

The computer must currently be the active controller to execute this statement, and primary addressing (but not multiple listeners) must be specified.

Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL TAD TCT $\overline{\text{ATN}}$	Error	ATN UNL TAD TCT $\overline{\text{ATN}}$
Not Active Controller	Error			

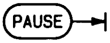
BASIC/UX Specifics

You cannot pass control on an interface containing a swap device or mounted file system.

PAUSE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement suspends program execution. (Also see TRACE PAUSE.)



Semantics

PAUSE suspends program execution before the next line is executed, until the CONTINUE key is pressed or CONT is executed. If the program is modified while paused, RUN must be used to restart program execution.

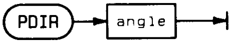
When program execution resumes, the computer attempts to service any ON INTR events that occurred while the program was paused. ON END, ON ERROR, or ON TIMEOUT events generate errors if they occur while the program is paused. ON KEY and ON KNOB events are ignored while the program is paused.

Pressing the PAUSE (or Stop on an ITF keyboard) key, or typing **PAUSE** and pressing EXECUTE, ENTER or Return will suspend program execution at the end of the line currently being executed.

PDIR

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement specifies the angle with which IPLOT, RPLOT, POLYGON, POLYLINE, and RECTANGLE output are rotated.



Item	Description	Range
angle	numeric expression in current units of angle; Default = 0	—

Example Statements

```
PDIR 20
PDIR ACS(Side)
```

Semantics

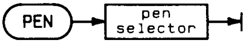
The rotation is about the local origin of the RPLOT, POLYGON, POLYLINE or RECTANGLE.

The angle is interpreted as counter-clockwise rotation from the X-axis.

PEN

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement selects a pen value to be used for all subsequent lines. (For information about PEN as a secondary keyword, see the AREA statement.)



Item	Description	Range
pen selector	numeric expression, rounded to an integer	−32 768 thru +32 767 (device dependent)

Example Statements

```
PEN 4
PEN Select
PEN Pen_number(I,J)
```

Semantics

For devices which support more than one line color (color CRT), or physical pen (external hard copy plotters), this statement specifies the line color or physical pen to be used for all subsequent lines until the execution of another PEN statement or until the execution of a PLOT, IPLOT, RPLOT, or SYMBOL statement with an array argument which changes the pen color (see Operation Selector 3 of these statements). The sign of the pen selectors affects the drawing mode.

In color map mode, specifying PEN 14 actually means "write a 14 into the frame buffer." The value of the frame buffer specifies the entry in the color map to be used, which in turn describes the actual color to be used.

The PEN statement can also be used to specify that the current drawing mode is to erase lines on all devices which support such an operation. This is specified with a negative pen number. An alternate mode of operation which allows non-dominant and complementing drawing may be accessed through the GESCAPE function.

When the PEN statement is executed, the pen used is mapped into the appropriate range, retaining the sign. For example, if you specify pen +8 on a device whose pens range from −7 through 7, it would actually use pen +1. The formulae used are as follows:

For monochromatic displays:

If pen selector > 0 then use PEN 1 (draw lines)
If pen selector = 0 then use PEN 0 (complement[1] lines)
If pen selector < 0 then use PEN −1 (erase lines)

For color displays *not* in COLOR MAP mode, and the HP 98627A:

If pen selector > 0 then use PEN (pen selector − 1) MOD 7 + 1
If pen selector = 0 then use PEN 0 (complement)
If pen selector < 0 then use PEN − ((ABS(pen selector) − 1) MOD 7 + 1)

For color displays in COLOR MAP mode:

If pen selector>0 then use PEN (pen selector − 1) MOD MaxPen + 1
If pen selector=0 then use PEN 0
If pen selector<0 then use PEN − ((ABS(pen selector) − 1) MOD MaxPen + 1)
Where MaxPen is the highest pen number (the lowest is 0). Four planes: MaxPen=15; six planes: MaxPen=63; eight planes: MaxPen=255.

For an HPGL Plotter: use PEN pen selector

On an HPGL plotter, no checking is done to determine if the requested pen actually exists. Pen 0 puts away any pen if the plotter supports such an operation.

[1] "Complement" means to change the state of pixels; that is, to draw lines where there are none, and to erase where lines already exist.

Non-Color-Map Mode

The value written into the frame buffer depends not only on what pen is being used, but whether or not the computer is in color map mode. The colors for the default (non-color map) mode are given because the color map cannot be changed in this mode.

The meanings of the different pen values are shown in the table below. The pen value can cause either a 1 (draw), a 0 (erase), no change, or invert the value of each location in the frame buffer.

Non-Color-Map Mode

Pen	Color	Plane 1 (Red)	Plane 2 (Green)	Plane 3 (Blue)
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1

Drawing with the pen numbers indicated in the above table results in the frame buffer planes being set to the indicated values. Drawing with the negatives of the pen numbers while in *normal pen mode* causes the bits to be cleared where there are 1s in the table. Drawing with the negatives of the pen numbers while in *alternate pen mode* causes the bits to be inverted where there are 1s in the table. In either case, no change will take place where there are 0s in the table. Although complementing lines can be drawn, complementing area fills cannot be executed.

Positive pen numbers in alternate drawing mode allows non-dominant drawing. (Non-dominant drawing causes the values in the frame buffer to be inclusively ORed with the value of the pen.) Pen 0 in normal mode complements. Pen 0 in alternate mode draws in the background color. Since the table represents the computer in non-color map mode, table entries for any additional frame buffer planes are all zeros.

Color Map Mode

When operating the color display in color map mode, pen colors can be redefined at will. For this reason, no colors are mentioned in the following table. Unlike non-color-map mode, the fourth bit in the frame buffer is used when in color map mode. Also, memory planes 1, 2, and 3 are not associated with red, green, and blue.

Drawing with a pen merely puts the pen number into that pixel's location. The computer looks into the corresponding entry in the color map to determine what the actual color the pixel is to exhibit.

Pen	Action	Plane 1	Plane 2	Plane 3	Plane 4
0	Background	0	0	0	0
1	Draw Pen 1	1	0	0	0
2	Draw Pen 2	0	1	0	0
3	Draw Pen 3	1	1	0	0
4	Draw Pen 4	0	0	1	0
5	Draw Pen 5	1	0	1	0
6	Draw Pen 6	0	1	1	0
7	Draw Pen 7	1	1	1	0
8	Draw Pen 8	0	0	0	1
9	Draw Pen 9	1	0	0	1
10	Draw Pen 10	0	1	0	1
11	Draw Pen 11	1	1	0	1
12	Draw Pen 12	0	0	1	1
13	Draw Pen 13	1	0	1	1
14	Draw Pen 14	0	1	1	1
15	Draw Pen 15	1	1	1	1

Drawing with the negatives of the pen numbers while in *normal pen mode* causes the bits to be cleared where there are 1s in the table. Drawing with the negatives of the pen numbers while in *alternate pen mode* causes the bits to be inverted where there are 1s in the table. In either case, no change will take place where there are 0s in the table.

Pen 0 merely draws in the background color. Although complementing lines can be drawn, complementing area fills cannot be executed.

Default Colors

The RGB and HSL values for the default pen colors while in color map mode are shown below. These can be changed by the SET PEN statement. First, the RGB (red/green/blue) values:

Pen	Color	Red	Green	Blue
0	Black	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1
8	Black	0	0	0
9	Olive Green	.80	.73	.20
10	Aqua	.20	.67	.47
11	Royal Blue	.53	.40	.67
12	Maroon	.80	.27	.40
13	Brick Red	1.00	.40	.20
14	Orange	1.00	.47	0.00
15	Brown	.87	.53	.27

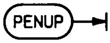
The same default color map colors are represented below in their HSL (hue/saturation/luminosity) representations:

Pen	Color	Hue	Sat.	Lum.
0	Black	0	0	0
1	White	0	0	1
2	Red	0	1	1
3	Yellow	.17	1	1
4	Green	.33	1	1
5	Cyan	.50	1	1
6	Blue	.67	1	1
7	Magenta	.83	1	1
8	Black	0	0	0
9	Olive Green	.15	.75	.80
10	Aqua	.44	.75	.68
11	Royal Blue	.75	.36	.64
12	Maroon	.95	.65	.78
13	Brick Red	.04	.80	1.00
14	Orange	.08	1.00	1.00
15	Brown	.08	.70	.85

PENUP

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

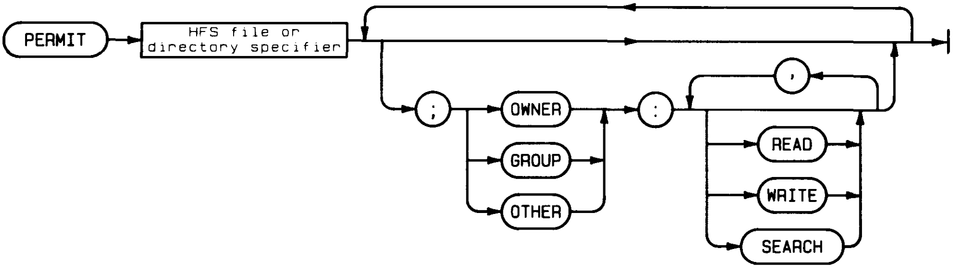
This statement lifts the pen on the current plotting device.



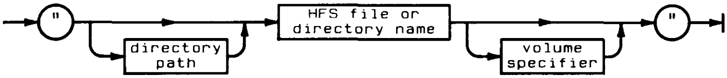
PERMIT

Supported On	WS,UX
Option Required	HFS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement modifies the owner, group, or public access permissions of an HFS file or directory.



literal form of HFS file or directory specifier:



Item	Description	Range
HFS file or directory specifier	string expression	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file or directory name	literal	1 to 14 characters (see Glossary)
volume specifier	literal	(see MASS STORAGE IS)

Example Statements

```
PERMIT Dir_path$& File$& Volume$
PERMIT "/DirPath/HFSfile";OWNER:READ,WRITE; GROUP:READ
PERMIT "/DirPath/Dir";OTHER:SEARCH
PERMIT "File"; OWNER:READ,WRITE; OTHER:READ
PERMIT "Dir"; GROUP:READ; OTHER:
PERMIT "File"
PERMIT "Directory"
```

Semantics

The PERMIT statement is used to:

- change the permissions (access rights) of a file or directory on an HFS disc,
- permit or restrict access to files and directories by the file owner, a member of the file-owner's group, or by all others.

Restricting access is useful, for instance, to prevent accidental purges of files or to prevent others from reading or writing to a file.

You must be the current owner of the file or directory in order to execute PERMIT.

There are 9 bits of "permission" for HFS files.

OWNER			GROUP			OTHER		
READ	WRITE	SEARCH	READ	WRITE	SEARCH	READ	WRITE	SEARCH

These bits are shown in the PERMISSION column of a CAT listing of the directory in which the file or directory resides (R for READ; W for WRITE; X for SEARCH; - for "no permission"):

```
                FILE    NUM   REC     MODIFIED
FILE NAME       TYPE   RECS   LEN DATE       TIME PERMISSION OWNER GROUP
============== ===== ====== ===== =============== ========== ===== =====
File                   8192     1 7-Nov-86   9:23 RW-RW-RW-     18     9
Directory               256     1 7-Nov-86   9:24 RWXRWXRWX     18     9
```

The default permission bits for directories are: RWXRWXRWX.
The default permission bits for files are: RW-RW-RW-.

There are three **classes of users**:

- OWNER—initially the person who created the file (ownership can be changed with the CHOWN statement). All BASIC Workstation files are created with an owner identifier of 18. BASIC/UX files default to the owner's user id.
- GROUP—initially the "group" to which the file's/directory's "owner" belongs (but the group can be changed with the CHGRP statement). All BASIC Workstation files are created with a group identifier of 9. BASIC/UX files default to the user's group id.
- OTHER—all other users who are not the owner and are not in the same group as the owner (known as "public" on the HP-UX system).

Each class of users has three **types of permissions** for accessing a file or directory:

- READ—allows reading the file (such as with ASSIGN, ENTER, and GET).
- WRITE—allows a user to modify the file's contents (such as with OUTPUT or RE-STORE).
- SEARCH—an operation on directories which allows you to include the directory in a directory path (such as with CAT and MASS STORAGE IS).

When a user class is specified, all permission bits for that class are changed:

- If a permission is *specified*, then the corresponding permission bit is *set*;
- If a permission is *omitted*, the corresponding permission bit is *cleared.*

For example, executing:

```
PERMIT "Div";Other:
```

gives the following permission:

```
RWXRWX---
```

If no user class is specified, the default permissions for that file are restored.

For example executing:

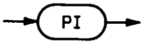
```
PERMIT "File"
```

gives the following permission:

```
RW-RW-RW-
```

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns 3.141 592 653 589 79, which is an approximate value for π.



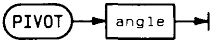
Example Statements

```
Area=PI*Radius^2
PRINT X,X*2*PI
```

PIVOT

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement specifies a rotation of coordinates which is applied to all subsequently drawn lines.



Item	Description	Range
angle	numeric expression in current units of angle	(same as COS)

Example Statements

```
PIVOT 30
IF Special THEN PIVOT Radians
```

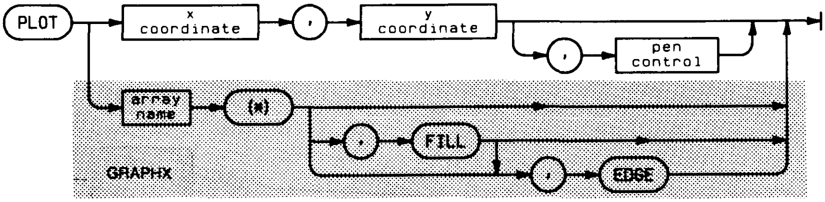
Semantics

The specified angle is interpreted according to the current angle mode (RAD or DEG).

The specified angular rotation is performed about the logical pen's position at the time the PIVOT is executed. This rotation is applied only to lines drawn subsequent to the PIVOT; logical pen movement is **not** affected by PIVOT. Consequently, PIVOT generally causes the logical and physical pens to be left at different positions. Other operations which cause similar effects are attempts to draw outside clip limits and direct HPGL output to plotters.

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement moves the pen from the current pen position to the specified X and Y coordinates. It can be used to move without drawing, or to draw a line, depending on the pen control value.



Item	Description	Range
x coordinate	numeric expression, in current units	—
y coordinate	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move)	−32 768 thru +32 767
array name	name of two-dimensional, two-column or three-column numeric array. (Requires GRAPHX)	any valid name

Example Statements

```
PLOT X,Y,-1
PLOT -5,12
PLOT Shape(*),FILL,EDGE
```

Semantics

Non-Array Parameters

The specified X and Y position information is interpreted according to the current unit-of-measure. Lines are drawn using the current pen color and line type.

PLOT is affected by the PIVOT transformation.

The line is clipped at the current clipping boundary. If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

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

Note 4: RPLOT and IPLOT are affected by PDIR.

The optional pen control parameter specifies the following plotting actions; the default value is +1 (down after move).

Pen Control Parameter

Pen Control	Resultant Action
−Even	Pen up before move
−Odd	Pen down before move
+Even	Pen up after move
+Odd	Pen down after move

The above table is summed up by: even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Array Parameters

When using the PLOT statement with an array, either a two-column or a three-column array may be used. If a two-column array is used, the third parameter is assumed to be +1; pen down after move.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the range 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the PLOT statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the PLOT statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the PLOT statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

When using a PLOT statement with an array, the following table of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth. See the following list.

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 PLOT statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens

An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types

An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color

Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation selector 14 is used in conjunction with operation selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessarily immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word; that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities (which range from 0 thru 1) for red, green, and blue in the variables `R`, `G`, and `B`, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored, and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME

Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the PLOT statement, so one probably would not have more than one operation selector 12 in an array to PLOT, since the last FRAME will overwrite all the previous ones.

Premature Termination

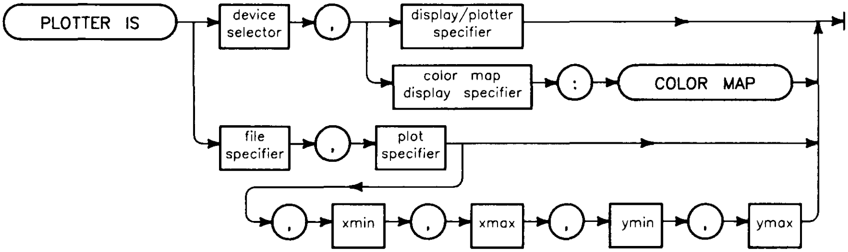
Operation selector 8 causes the PLOT statement to be terminated. The PLOT statement will successfully terminate if the actual end of the array has been reached, so the use of operation selector 8 is optional.

Ignoring Selected Rows in the Array

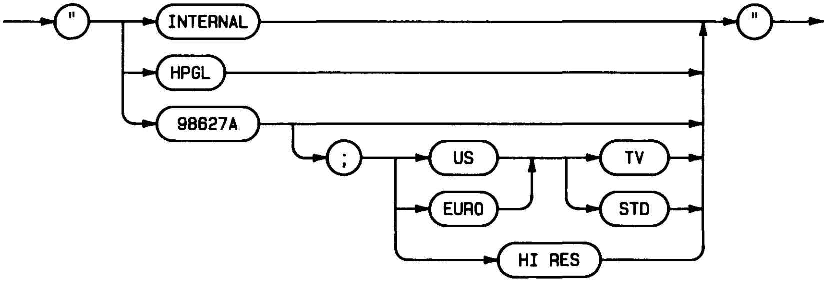
Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement selects a plotting device, file, or pipe.



literal form of display/plotter specifier:



Item	Description	Range
device selector	numeric expression, rounded to an integer	(see Glossary)
display/plotter specifier	string expression	(see drawing)
color map display specifier	string expression	INTERNAL or WINDOW
file specifier	string expression	(see drawing)
plot specifier	string expression	HPGL
window specifier	numeric expression	WINDOW

xmin	numeric expression;
	Default = −392.75mm
		device dependent

xmax	numeric expression;
	Default = 392.75mm
		device dependent

ymin	numeric expression;
	Default = −251.5mm
		device dependent

ymax	numeric expression;
	Default = 251.5mm
		device dependent

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)

Example Statements

```
PLOTTER IS 3,I$
PLOTTER IS CRT,"INTERNAL";COLOR MAP
PLOTTER IS Dsg,"HPGL"
PLOTTER IS "Newfile","HPGL"
PLOTTER IS "/PL/PlotFile"
PLOTTER IS "PlotFile:REMOTE","HPGL",6.25,256.25,6.975,186.975
PLOTTER IS 601,"WINDOW";COLOR MAP              (BASIC/UX only)
```

Semantics

Plotters

The hard clip limits of the plotter are read in when this statement is executed. Therefore, the specified device must be capable of responding to this interrogation.

Files

This statement causes all subsequent plotter output to go to the specified file.

Xmin, Xmax, Ymin, Ymax are the hard clip limits of the plotter in millimeters.

This assumes 0.025 mm per plotter unit. The default size is for an HP 7580 or HP 7585 D-size drawing. See the plotter manual for more information on plotter limits.

The PLOTTER IS statement positions the file pointer to the beginning of the file.

The file is closed when another PLOTTER IS statement is executed or SCRATCH A, GINIT or Reset is executed.

If you want to send HPGL commands to a file that is currently the PLOTTER IS device, use the GSEND statement. (See the GSEND entry of this reference for details.)

An end-of-file error occurs when the end of a LIF file is reached.

SRM and HFS Files

In order to write to a PLOTTER IS file on an HFS volume, you need to have R (read) and W (write) permission on the file, and X (search) permission on all superior directories.

In order to write to a PLOTTER IS file in an SRM volume, you need to have R (read) and W (write) permissions on the file, as well as R permission on all superior directories.

No end-of-file errors occur on SRM or HFS files, because these files are extensible. That is, if the data output to the file with this statement would overflow the file's space allocation, the file is automatically extended provided the disc contains enough unused storage space.

SRM Plotter Spoolers

If the specified file is in the SRM plotter spooler directory and the file contains data, then the SRM system sends the data to the plotting device (when the file is closed) and then purges the file. You may close the file by executing another PLOTTER IS statement, GINIT, SCRATCH A or SCRATCH BIN, or by pressing RESET (SHIFT-PAUSE or Shift-Break).

Displays

The statement `PLOTTER IS CRT, "INTERNAL"` is executed whenever a graphics statement is executed which needs a plotter (see GINIT) and no plotter is active. The plotter activated is the first device encountered in the following order:

1. The alpha display, if it has graphics capabilities
2. Internal 98542A, 98543A, 98544A, 98545A, 98547A, 98548A, 98549A, 98550A, 98700, or 98720 at select code 6
3. Non-bit-mapped alpha display with graphics capabilities at select code 3 (BASIC/UX supports the 98546A compatibility interface only)
4. External 98700 or 98720 at select code > 7
5. 98627A at select code > 7 (BASIC/WS only).

If the COLOR MAP option is specified and the plotting device has a color map, the capability of changing the color map is enabled (see SET PEN). Also, the values written into the frame buffer are different than they would be if color map mode was not enabled.

If the COLOR MAP option is not included and the plotting device is the Model 236 color display, the 4th memory plane is cleared (BASIC/WS only).

Non-Color Map Mode

Executing a PLOTTER IS statement *without* the COLOR MAP keyword causes the color map to be defined as follows, where 0 is zero intensity and 1 is full intensity. This emulates the HP 98627A non-color-mapped device on a color bit-mapped display.

Pen	Color	Red	Green	Blue
0	Complement	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1

On a display with bit-mapped alpha, the non-color map mode affects the ALPHA PEN, PRINT PEN, KEY LABELS PEN, and KBD LINE PEN statements as follows: 8 is black (the same as 0) and 9 through 15 are white (the same as 1).

The complementing cursor will be white on top of all colors except white, in which case it will be black.

COLOR MAP

In the COLOR MAP mode, the color map is initialized so that the first eight colors are the same as they were in the default mode, and the second eight colors simulate HP's designer colors of plotter pen ink.

Although the pen numbers select the same color in color map mode as in non-color map mode (for the first eight pens), the actual values written to the frame buffer are different. This results from the different interpretation of the values in the frame buffer: in non-color map mode, the values are RGB values; in color-map mode, the values are indices into the color map. This means that a picture drawn in non-color map mode will change colors if a PLOTTER IS with the COLOR MAP option is executed. The reverse is also true.

On a console or a terminal, when the `PLOTTER IS` statement is executed, the color map is initialized to a default state. If the graphics write-enable mask is left in the default mode, the entire color map will be initialized as before. Otherwise, the following algorithm is used: all color map entries whose binary representation has `1`'s in non-graphics planes will remain unchanged. This is done to insure that only pens dedicated to graphics are initialized. For example, with a graphics write mask of 7 (binary 0000 0111), only pens 0 through 7 are initialized. Higher numbered pens would remain unchanged since their binary representation would have 1s in non-graphics planes.

In windows, the color map is initialized to whatever the color map was when BASIC was booted.

Display Specifiers

There are several values which can be used when specifying the display on which graphics operations are done:

PLOTTER IS CRT,"INTERNAL" or
PLOTTER IS 1,"INTERNAL"

This is the safest of the possibilities. "CRT" is a built-in function which returns the value 1, and the value 1 is interpreted by the graphics system as "the default display." The default display may be an external display if no internal display exists.

PLOTTER IS 3,"INTERNAL"

This specifies a non-bit-mapped display if there is one; otherwise, the action is equivalent to "PLOTTER IS 1,"INTERNAL"". Specifying a value of 3 makes sense for all Series 200 displays except the Model 237.

PLOTTER IS 6,"INTERNAL"

Always specifies a bit-mapped display. If one is not found, an error results.

PLOTTER IS ⟨*device selector*⟩,"INTERNAL"

With the 98700 and 98720 displays, it is possible to configure the display card so that it is at an external select code. For example, if you set the select code to 25, you would say:

```
PLOTTER IS 25,"INTERNAL"
```

PLOTTER IS (*window id*),"WINDOW"
(BASIC/UX only)

This specifier works only in a windowing environment. A window id of 600 is equivalent to PLOTTER IS CRT,"INTERNAL" in the windowing environment.

`PLOTTER IS` <*device selector*>`,"98627A"`[1] (BASIC/WS only)	This specifies a color graphics display connected through the 98627A interface card. This may have any one of several options specifying television format, etc. See the following table.

HP 98627A Display Specifiers

Desired Display Format	Display Specifier
Standard Graphics	

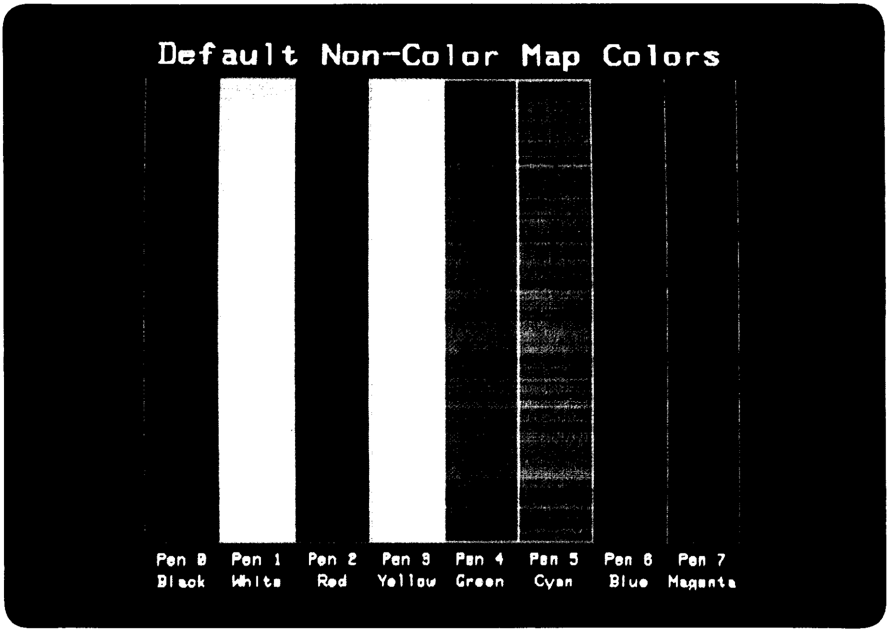
512 by 390 pixels,
60 Hz, non-interlaced
	`"98627A" or`
	`"98627A;US STD"`

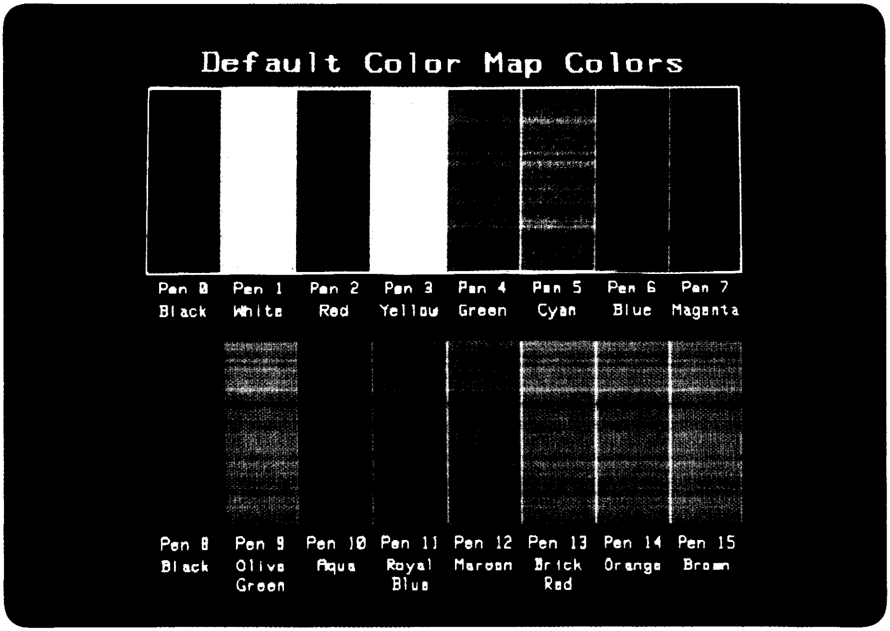
512 by 390 pixels, 50 Hz, non-interlaced	`"98627A;EURO STD"`
High-Resolution Graphics	
512 by 512 pixels 46.5 Hz, non-interlaced	`"98627A;HI RES"`
TV Compatible Graphics	
512 by 474 pixels, 60 Hz, interlaced (30 Hz refresh rate)	`"98627A;US TV"`
512 by 512 pixels, 50 Hz, interlaced (25 Hz refresh rate)	`"98627A;EURO TV"`

Default Pen Colors

The PLOTTER IS statement defines the color map to default values in a non-windowing environment. These values are different depending on whether or not the COLOR MAP option was selected. The two color plates on the next page show eight default colors available with non-color map mode, and the sixteen default colors in color map mode.

[1] `PLOTTER IS <device selector>, "INTERNAL" is also accepted.`


Default Non-Color Map Colors
Pen 0 Black
Pen 1 White
Pen 2 Red
Pen 3 Yellow
Pen 4 Green
Pen 5 Cyan
Pen 6 Blue
Pen 7 Magenta


Default Color Map Colors
Pen 0 Black
Pen 1 White
Pen 2 Red
Pen 3 Yellow
Pen 4 Green
Pen 5 Cyan
Pen 6 Blue
Pen 7 Magenta
Pen 8 Black
Pen 9 Olive Green
Pen 10 Aqua
Pen 11 Royal Blue
Pen 12 Maroon
Pen 13 Brick Red
Pen 14 Orange
Pen 15 Brown

The values, both in RGB and HSL, of the sixteen default pen colors are given below:

Color Map Default Color Definitions (RGB)

Pen	Color	Red	Green	Blue
0	Black	0	0	0
1	White	1	1	1
2	Red	1	0	0
3	Yellow	1	1	0
4	Green	0	1	0
5	Cyan	0	1	1
6	Blue	0	0	1
7	Magenta	1	0	1
8	Black	0	0	0
9	Olive Green	.80	.73	.20
10	Aqua	.20	.67	.47
11	Royal Blue	.53	.40	.67
12	Maroon	.80	.27	.40
13	Brick Red	1.00	.40	.20
14	Orange	1.00	.47	0.00
15	Brown	.87	.53	.27

The same default color map colors are represented below in their HSL (hue/saturation/luminosity) representations:

Color Map Default Color Definitions (HSL)

Pen	Color	Hue	Sat.	Lum.
0	Black	0	0	0
1	White	0	0	1
2	Red	0	1	1
3	Yellow	.17	1	1
4	Green	.33	1	1
5	Cyan	.50	1	1
6	Blue	.67	1	1
7	Magenta	.83	1	1
8	Black	0	0	0
9	Olive Green	.15	.75	.80
10	Aqua	.44	.75	.68
11	Royal Blue	.75	.36	.64
12	Maroon	.95	.65	.78
13	Brick Red	.04	.80	1.00
14	Orange	.08	1.00	1.00
15	Brown	.08	.70	.85

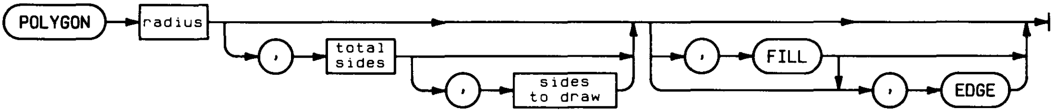
Eight-plane machines have 256-entry color maps. In these machines, pens 16 through 255 are defined to a variety of shades. For exact values, interrogate the color map with GESCAPE.

BASIC/UX Specifics

BASIC/UX treats output to a pipe as it would output to a file. The pipe must be explicitly closed before any output becomes permanent (or takes place). Output to a spooled device will not be sent to the spooler until the pipe has been closed. The closing of pipes can be achieved with a subsequent PLOTTER IS, QUIT, or SCRATCH command.

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement draws all or part of a closed regular polygon. The polygon can be filled and/or edged.



Item	Description	Range
radius	numeric expression, in current units	—
total sides	numeric expression, rounded to an integer. Default = 60	3 thru 32 767
sides to draw	numeric expression, rounded to an integer. Default = all sides	1 thru 32 767

Example Statements

```
POLYGON 1,5,5,4,FILL,EDGE
POLYGON 4
```

Semantics

The radius is the distance that the vertices of the polygon will be from the logical pen position. The first vertex will be at a distance specified by "radius" in the direction of the positive X-axis. Specifying a negative radius results in the figure being rotated 180°. POLYGON is affected by the PIVOT and the PDIR transformations.

The total sides and the number of sides drawn need not be the same. Thus:

```
POLYGON 1.5,8,5
```

will start to drawn an octagon whose vertices are 1.5 units from the current pen position, but will only draw five sides of it before closing the polygon at the first point. If the number of sides to draw is greater than the specified total sides, sides to draw is treated as if it were equal to total sides.

POLYGON forces polygon closure, that is, the first vertex is connected to the last vertex, so there is always an inside and an outside area. This is true even for the degenerate case of drawing only one side of a polygon, in which case a single line results. This is actually two lines, from the first point to the last point, and back to the first point.

Polygon Shape

The shape of the polygon is affected by the viewing transformation specified by SHOW or WINDOW. Therefore, anisotropic scaling causes the polygon to be distorted; stretched or compressed along the axes. If a rotation transformation is in effect, the polygon will be rotated first, then stretched or compressed along the unrotated axes.

The pen status also affects the final shape of a polygon if sides to draw is less than total sides. If the pen is up at the time POLYGON is specified, the first vertex specified is connected to the last vertex specified, *not* including the center of the polygon, which is the current pen position. If the pen is down, however, the center of the polygon is also included in it. If sides to draw is less than total sides, piece-of-pie shaped polygon segments are created.

FILL and EDGE

FILL causes the interior of the polygon or polygon segment to be filled with the current fill color as defined by AREA PEN, AREA COLOR, or AREA INTENSITY. EDGE causes the edges of the polygon to be drawn using the current pen and line type. If both FILL and EDGE are specified, the interior will be filled, then the edge will be drawn. If neither FILL nor EDGE is specified, EDGE is assumed.

Polygons sent to an HPGL plotter are edged but not filled regardless of any FILL or EDGE directives in the statement.

After POLYGON has executed, the pen is in the same position it was before the statement was executed, and the pen is up. The polygon is clipped at the current clip limits.

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

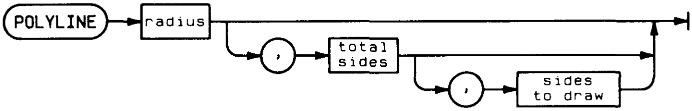
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.

Note 4: RPLOT and IPLOT are affected by PDIR.

POLYLINE

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement draws all or part of an open regular polygon.



Item	Description	Range
radius	numeric expression, in current units	—
total sides	numeric expression, rounded to an integer. Default = 60	3 thru 32 767
sides to draw	numeric expression, rounded to an integer. Default = all sides	1 thru 32 767

Example Statements

```
POLYLINE Radius,Sides,Sides_to_draw
POLYLINE 12,5
```

Semantics

The radius is the distance that the vertices of the polygon will be from the current pen position. The first vertex will be at a distance specified by "radius" in the direction of the positive X-axis. Specifying a negative radius results in the figure being rotated 180°. POLYLINE is affected by the PIVOT and the PDIR transformations.

The total sides and the number of sides drawn need not be the same. Thus:

```
POLYLINE 1.5,8,5
```

will start to drawn an octagon whose vertices are 1.5 units from the current pen position, but will only draw five sides of it. If the number of sides to draw is greater than the total sides specified, it is treated as if it were equal to the total sides.

Shape of Perimeter

POLYLINE does not force polygon closure, that is, if sides to draw is less than total sides, the first vertex is not connected to the last vertex, so there is no "inside" or "outside" area.

The shape of the polygon is affected by the viewing transformation specified by SHOW or WINDOW. Therefore, anistropic scaling causes the perimeter to be distorted; stretched or compressed along the axes. If a rotation transformation is in effect, the polygon will be rotated first, then stretched or compressed along the unrotated axes.

The pen status also affects the way a POLYLINE statement works. If the pen is up at the time POLYLINE is specified, the first vertex is on the perimeter. If the pen is down, the first point is the current pen position, which is connected to the first point on the perimeter.

After POLYLINE has executed, the current pen position is in the same position it was before the statement was executed, and the pen is up. The polygon is clipped at the current clip limits.

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

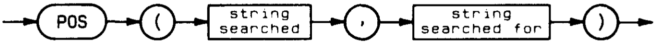
Note 3: The starting point for labels drawn after lines or axes is affected by PIVOT.

Note 4: RPLOT and IPLOT are affected by PDIR.

POS

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the first position of a substring within a string.



Item	Description	Range
string searched	string expression	—
string searched for	string expression	—

Example Statements

```
Point=POS(Big$,Little$)
IF POS(A$,CHR$(10)) THEN Line_end
```

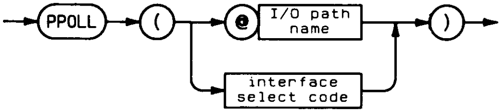
Semantics

If the value returned is greater than 0, it represents the position of the first character of the string being searched for in the string being searched. If the value returned is 0, the string being searched for does not exist in the string being searched (or the string searched for is the null string).

Note that the position returned is the relative position within the string expression used as the first argument. Thus, when a substring is searched, the position value refers to that substring, not to the parent string from which the substring was taken.

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns a value representing eight status-bit messages of devices on the HP-IB.



Item	Description	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

```
Stat=PPOLL(7)
IF BIT(PPOLL(@Hpib),3) THEN Respond
```

Semantics

The computer must be the active controller to execute this function.

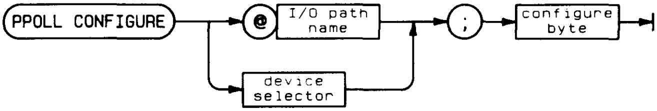
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error
Not Active Controller	Error			

PPOLL CONFIGURE

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement programs the logical sense and data bus line on which a specified device responds to a parallel poll.



Item	Description	Range
I/O path name	name assigned to a device or devices	any valid name
device selector	numeric expression, rounded to an integer	must contain a primary address (see Glossary)
configure byte	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
PPOLL CONFIGURE 711;2
PPOLL CONFIGURE @Dvm;Response
```

Semantics

This statement assumes that the device's response is bus-programmable. The computer must be the active controller to execute this statement.

The configure byte is coded. The three least significant bits determine the data bus line for the response. The fourth bit determines the logical sense of the response.

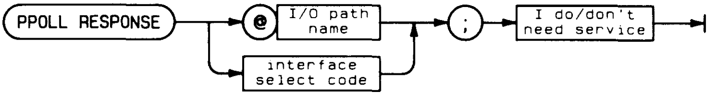
Summary of Bus Actions

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

PPOLL RESPONSE

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement defines a response to be sent when an Active Controller performs a Parallel Poll on an HP-IB Interface. The response indicates whether this computer does or does not need service.



Item	Description	Range
I/O path name	name assigned to an interface select code	any valid name
interface select code	numeric expression, rounded to an integer	7 thru 31
I do/don't need service	numeric expression, rounded to an integer	0 or 1

Examples

```
PPOLL RESPONSE @Hp_ib;I_need_service
PPOLL RESPONSE Interface;0
```

Semantics

This statement defines the computer's response to a Parallel Poll (ATN & EOI) performed by the current Active Controller on the specified HP-IB Interface. This statement only sets up a potential response; no actual response is generated when the statement is executed.

If the value of the "I do/don't need service" parameter is 0, the computer is directed to place a logical false on the bit on which it has been defined to respond; this response will tell the Active Controller that this (non-active) controller does not need service. Any non-zero, positive value of this parameter (within the stated range) directs the computer to set up a true response, which will tell a polling Active Controller that the computer requires service.

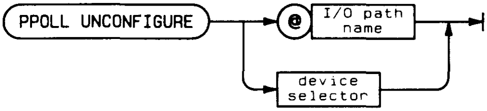
The bit on which the computer is to place its Parallel Poll response is determined by the value of the last "configure byte" written to CONTROL Register 5 of the corresponsing HP-IB Interface. In general, this configure byte can be read from HP-IB STATUS Register 7 by the service routine that responds to Parallel-Poll-Configuration-Change interrupts (Bit 14 of the Interrupt Enable Register). This configure byte may then be written into HP-IB CONTROL Register 5, and the response desired by the Active Controller will be sent when a Parallel Poll is conducted.

This statement may be executed by either an Active Controller or a non-active controller.

PPOLL UNCONFIGURE

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement disables the parallel poll response of a specified device or devices.



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

Example Statements

```
PPOLL UNCONFIGURE 7
PPOLL UNCONFIGURE @Plotter
```

Semantics

The computer must be the active controller to execute PPOLL UNCONFIGURE.

If multiple devices are specified by an I/O path name, all specified devices are deactivated from parallel poll response. If the device selector or I/O path name refers only to an interface select code, all devices on that interface are deactivated from parallel poll response.

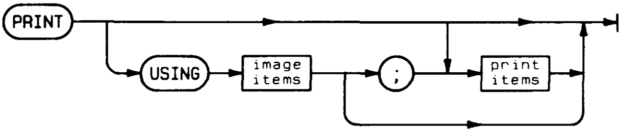
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			

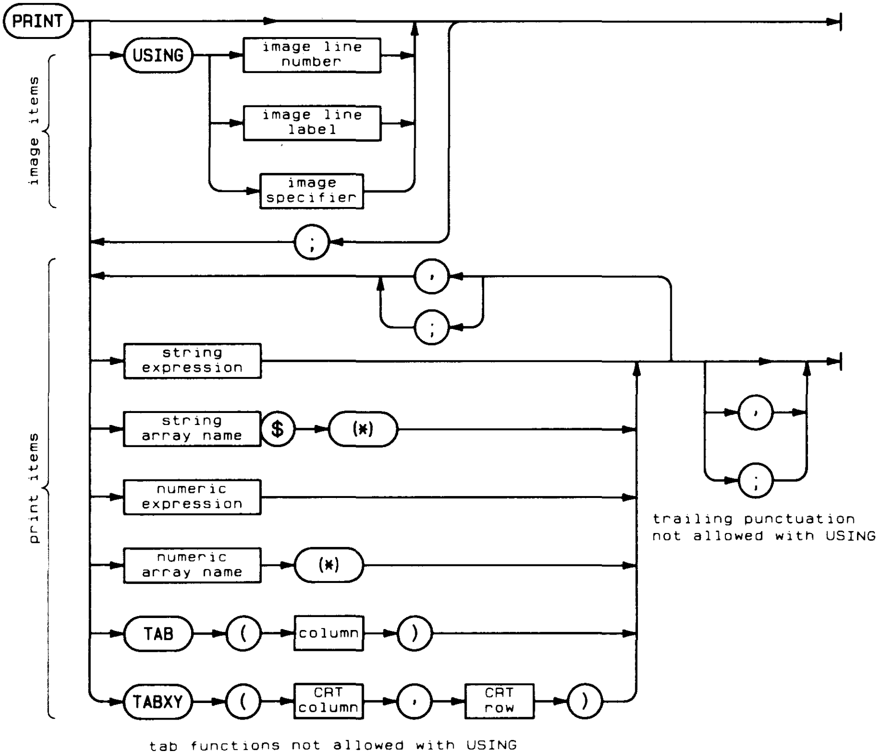
PRINT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

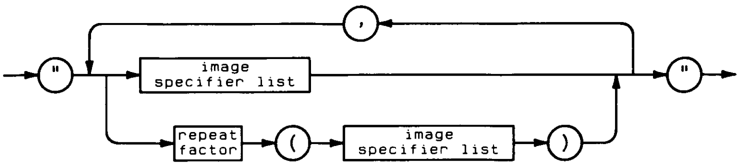
This statement sends items to the PRINTER IS device.



Expanded diagram:



literal form of image specifier

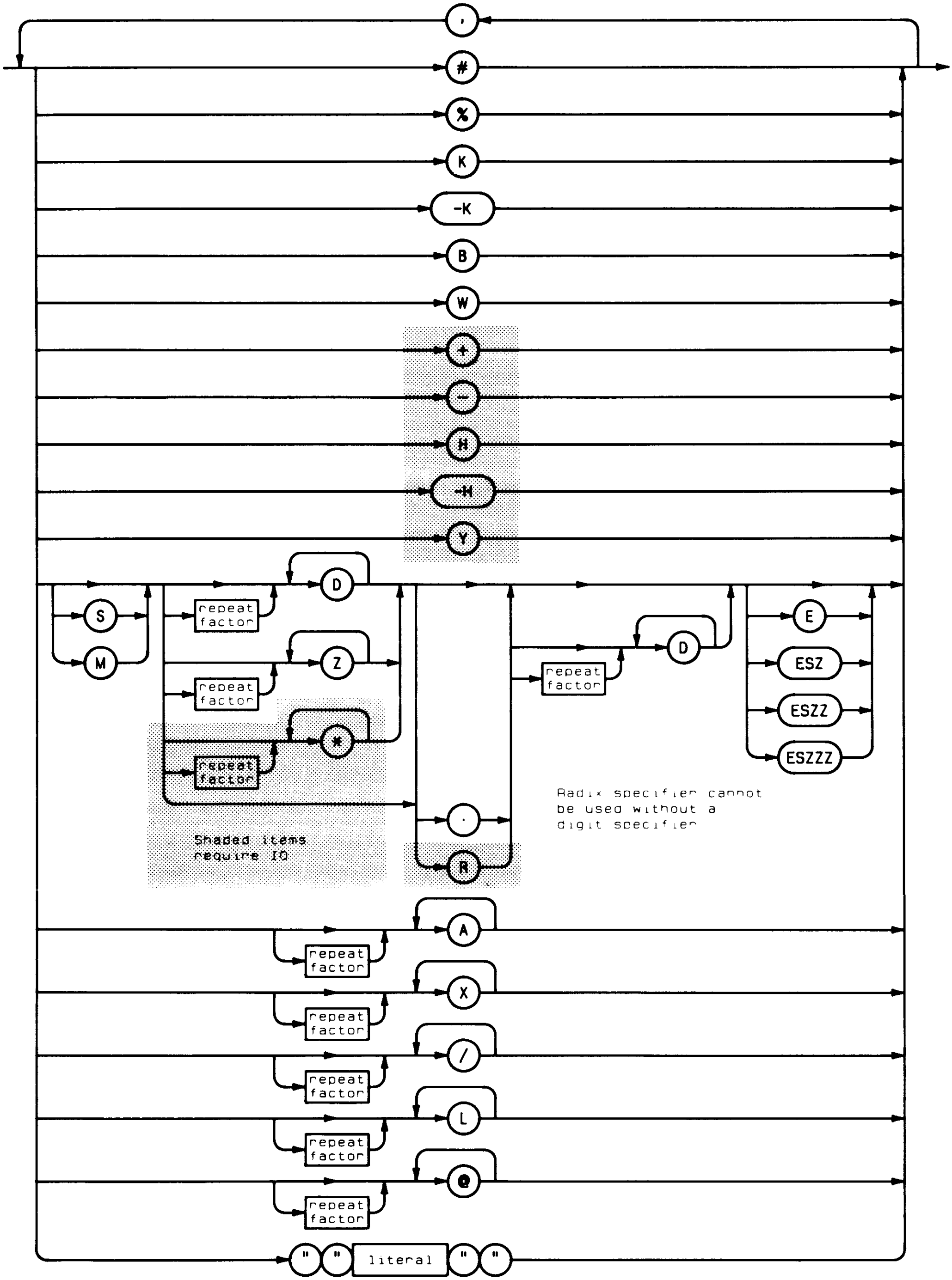


Item	Description	Range
image line number	integer constant identifying an IMAGE statement	1 thru 32 766
image line label	name identifying an IMAGE statement	any valid name
image specifier	string expression	(see drawing)
string array name	name of a string array	any valid name
numeric array name	name of a numeric array	any valid name
column	numeric expression, rounded to an integer	device dependent
CRT column	numeric expression, rounded to an integer	1 thru screen width
CRT row	numeric expression, rounded to an integer	1 thru alpha height
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

Example Statements

```
PRINT "LINE";Number
PRINT Array(*);
PRINT TABXY(1,1),Header$,TABXY(Col,3),Message$
PRINT USING "5Z.DD";Money
PRINT USING Fmt3;Id,Item$,Kilograms/2.2
```

image specifier list



Radix specifier cannot be used without a digit specifier

Shaded items require IO

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 print list is exhausted, an End-Of-Line (EOL) sequence is sent to the PRINTER IS device, unless it is suppressed by trailing punctuation or a pound-sign (#) image specifier. The printer width for EOL sequences generation is set to the screen width (50, 80 or 128 characters) for CRTs and to 80 for external devices unless the WIDTH attribute of the PRINTER IS statement was specified. WIDTH is off for files. This "printer width exceeded" EOL is not suppressed by trailing punctuation, but can be suppressed by the use of an image specifier.

Control Codes

Some ASCII control codes have a special effect in PRINT statements if the PRINTER IS device is the CRT (device selector=1):

Character	Keystroke	Name	Action
CHR$(7)	CTRL-G	bell	Sounds the beeper
CHR$(8)	CTRL-H	backspace	Moves the print position back one character.
CHR$(10)	CTRL-J	line-feed	Moves the print position down one line.
CHR$(12)	CTRL-L	form-feed	Prints two line-feeds, then advances the CRT buffer enough lines to place the next item at the top of the CRT.
CHR$(13)	CTRL-M	carriage-return	Moves the print position to column 1.

The effect of ASCII control codes on a printer is device dependent. See your printer manual to find which control codes are recognized by your printer and their effects.

CRT Enhancements

There are several character enhancements (such as inverse video and underlining) available on some 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 printed using the asterisk specifier. Each element in an array is treated as a separate item by the PRINT statement, as if the items were listed separately, separated by the punctuation following the array specifier. If no punctation 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).

PRINT Fields

If PRINT is used without USING, the punctuation following an item determines the width of the item's print field; a semicolon selects the compact field, and a comma selects the default print field. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the print field to be used for the print item preceding it.

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

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

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

TAB

The TAB function is used to position the next character to be printed on a line. In the TAB function, a column parameter less than one is treated as one. A column parameter greater than zero is subjected to the following formula: TAB position = ((column − 1) MOD width) + 1; where "width" is 50 for the Model 226 CRT, 128 for Model 237 and other hi-resolution displays, and 80 for all other devices. If the TAB position evaluates to a column number less than or equal to the number of characters printed since the last EOL sequence, then an EOL sequence is printed, followed by (TAB position − 1) blanks. If the TAB position evaluates to a column number greater than the number of characters printed since the last EOL, sufficient blanks are printed to move to the TAB position.

TABXY

The TABXY function provides X-Y character positioning on the CRT. It is ignored if a device other than the CRT is the PRINTER IS device. TABXY(1,1) specifies the upper left-hand corner of the CRT. If a negative value is provided for CRT row or CRT column, it is an error. Any number greater than the screen width for CRT column is treated as the last column on the screen. Any number greater than the height of the output area for CRT row is treated as the last line of the output area. If 0 is provided for either parameter, the current value of that parameter remains unchanged.

Display Type	Output Area Height	Display Width
226	18	50
216, 220, 236, and 98546	18	80
98542 and 98543	19	80
237, 98544, 98545, 98547, 98549, and 98700	41	128
98548 and 98550	44	128

PRINT With Using

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

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

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

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

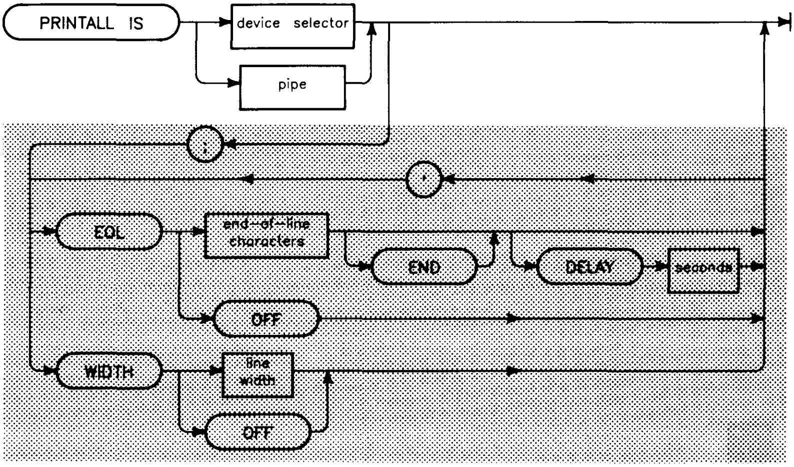
Image Specifier	Meaning
K	Compact field. Prints a number or string in standard form with no leading or trailing blanks.
−K	Same as K.
H	Similar to K, except the number is printed using the European number format (comma radix). (Requires IO)
−H	Same as H. (Requires IO)
S	Prints the number's sign (+ or −).
M	Prints the number's sign if negative, a blank if positive.
D	Prints one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is printed, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are printed.
*	Like Z, except that asterisks are printed instead of leading zeros. (Requires IO)
.	Prints a decimal-point radix indicator.
R	Prints a comma radix indicator (European radix). (Requires IO)
E	Prints an E, a sign, and a two-digit exponent.
ESZ	Prints an E, a sign, and a one-digit exponent.
ESZZ	Same as E.
ESZZZ	Prints an E, a sign, and a three-digit exponent.
A	Prints a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.

Image Specifier	Meaning
X	Prints a blank.
literal	Prints the characters contained in the literal.
B	Prints the character represented by one byte of data. This is similar to the CHR$ function. The number is rounded to an INTEGER and the least-significant byte is sent. If the number is greater than 32 767, then 255 is used; if the number is less than −32 768, then 0 is used.
W	Prints two characters represented by the two bytes in a 16-bit, two's-complement integer word. The corresponding numeric item is rounded to an INTEGER. If it is greater than 32 767, then 32 767 is used; if it is less than −32 768, then −32 768 is used. On an 8-bit interface, the most-significant byte is sent first. On a 16-bit interface, the two bytes are sent as one word in a single operation.
Y	Same as W. (Requires IO)
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence following the last print item.
%	Ignored in PRINT images.
+	Changes the automatic EOL sequence that normally follows the last print item to a single carriage-return. (Requires IO)
−	Changes the automatic EOL sequence that normally follows the last print item to a single line-feed. (Requires IO)
/	Sends a carriage-return and a line-feed to the PRINTER IS device.
L	Sends the current EOL sequence to the PRINTER IS device. The default EOL characters are CR and LF; see PRINTER IS for information on re-defining the EOL sequence. If the destination is an I/O path name with the WORD attribute, a pad byte may be sent after the EOL characters to achieve word alignment.
@	Sends a form-feed to the PRINTER IS device.

PRINTALL IS

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement assigns a logging device, file or pipe for recording operator interaction and troubleshooting messages.



Item	Description	Range
file specifier	string expression	-
device selector	numeric expression, rounded to an integer; Default = CRT	(see Glossary)
end-of-line characters	string expression; Default = CR/LF	0 thru 8 characters
seconds	numeric expression, rounded to the nearest 0.001 seconds; Default = 0	0.001 thru 32.767
line width	numeric expression, rounded to an integer; Default = infinity (see text)	1 thru 32 767

Example Statements

```
PRINTALL IS 701
PRINTALL IS Gpio
PRINTALL IS 701;EOL CHR$(13) END,WIDTH 65
PRINTALL IS 614                          (BASIC/UX in X Windows only)
PRINTALL IS "debug.out"                  (BASIC/UX only)
PRINTALL IS "| fold | pr -e -o8 | lp"    (BASIC/UX only)
```

Semantics

The printall device or file must be enabled by the PRT ALL key on the computer. The PRT ALL key is a toggle action device or file, enabling and disabling the printall operation. When the printall mode is enabled, all items generated by DISP, all operator input followed by the Return, ENTER, CONTINUE, or EXECUTE key, and all error messages from the computer are logged on the printall device or file. All TRACE activity is logged on the printall device or file if tracing is enabled.

An asterisk (*) is displayed on the PRINTALL softkey label of models with ITF keyboards, if printall mode is enabled.

At power-on and SCRATCH A, the default printall device is the CRT (select code 1).

The EOL Attribute (Requires IO)

The EOL attribute re-defines the end-of-line (EOL) sequence, which is sent at the following times: after the number of characters specified by *line width* and after each line of text. Up to eight characters may be specified as the EOL characters; an error is reported if the string contains more than eight characters. If END is included in the EOL attribute, an interface-dependent END indication is sent with the last character of the EOL sequence. If DELAY is included, the computer delays the specified number of seconds (after sending the last character) before continuing. The default EOL sequence consists of a carriage-return and a line-feed character with no END indication and no delay period.

The WIDTH Attribute (Requires IO)

The WIDTH attribute specifies the maximum number of characters which will be sent to the printing device or file before an EOL sequence is automatically sent. The EOL characters are not counted as part of the line width. The default width for the Model 226 CRT is 50, Model 237 and other high-resolution displays is 128, and the default for all other devices or file is 80. Specifying WIDTH OFF sets the width to infinity. If the default is desired, it must be restored explicitly. If the USING clause is included in the PRINT statement, the WIDTH attribute is ignored.

PRINTALL IS file

The file must be a BDAT or HP-UX file.

The PRINTALL IS file statement positions the file pointer to the beginning of the file.

The file is closed when another PRINTALL IS statement is executed and at SCRATCH A.

You can read the file with ENTER if it is ASSIGNed with FORMAT ON.

An end-of-file error occurs when the end of a LIF file is reached.

SRM and HFS Files

In order to write to a PRINTALL IS file on an HFS volume, you need to have R (read) and W (write) permission on the file, and X (search) permission on all superior directories.

In order to write to a PRINTALL IS file on an SRM volume, you need to have READ and WRITE capabilities on the immediately superior directory, as well as READ capabilities on all other superior directories.

No end-of-file error occurs when writing to a file on an SRM or HFS volume because these files are extensible. That is, if the data output to the file with this statement would otherwise overflow the file's space allocation, the BASIC system automatically allocates the additional space needed (provided the media contains enough unused storage space).

If the specified file is in the SRM printer spooler directory, is of type BDAT[1], and contains data, then the SRM system sends the data to the printer (after the file is closed) and then purges the file. You may close the file by executing another PRINTALL IS statement, or a SCRATCH A or SCRATCH BIN command.

BASIC/UX Specifics

On HP-UX systems, the line-printer is a spooled device. Writing directly to the printer as 701 may interfere with other spooled output. It is recommended that PRINTALL IS output be directed to either a file or the line-printer spooler by, for example, the statement:

```
PRINTALL IS "|lp"
```

BASIC/UX treats output to a pipe as it would output to a file. The pipe must be explicitly closed before any output becomes permanent (or takes place). Output to a spooled device will not be sent to the spooler until the pipe has been closed. The closing of pipes can be achieved with a subsequent PRINTALL IS, QUIT, or SCRATCH command.

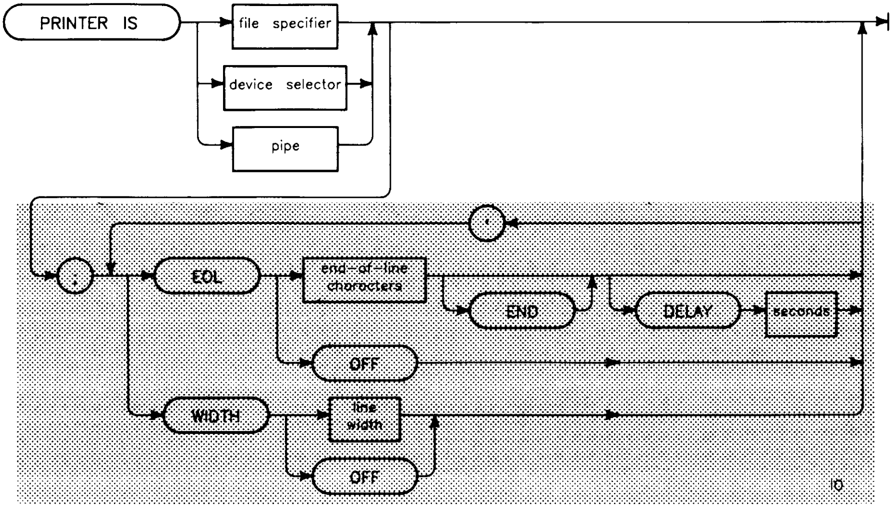
If PRINTALL IS device is a window and that window is destroyed (with DESTROY WINDOW), PRINTALL IS is undefined and generates an error.

[1] The SRM printer spooler will also spool ASCII files, which can be written by BASIC using OUTPUT, SAVE or RE-SAVE.

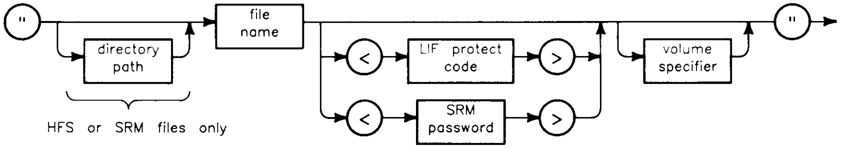
PRINTER IS

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement specifies the system printing device, file, or pipe.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	-
device selector	numeric expression, rounded to an integer	(see Glossary)

end-of-line characters	string expression;
	Default = CR/LF
		0 thru 8 characters

seconds	numeric expression, rounded to the nearest 0.001 seconds; Default=0	0.001 thru 32.767
line width	numeric expression, rounded to an integer; Default = (see text)	1 thru 32 767
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)

Example Statements

```
PRINTER IS 701
PRINTER IS 614                         (Windows only)
PRINTER IS Gpio
PRINTER IS "debug.out"
PRINTER IS 701;EOL CHR$(13) END,WIDTH 65
PRINTER IS "Myfile";WIDTH 80
PRINTER IS "Spooler:REMOTE"
PRINTER IS "My_dir/Temp_print";WIDTH 80
PRINTER IS " | fold | pr -e -o8| lp"    (BASIC/UX only)
```

Semantics

The system printing device or file receives all data sent by the PRINT statement and all data sent by CAT, LIST, and XREF statements in which the destination is not explicitly specified.

The default printing device is the CRT (select code 1) at power-on and after executing SCRATCH A.

The EOL Attribute (Requires IO)

The EOL attribute re-defines the end-of-line (EOL) sequence, which is sent at the following times: after the number of characters specified by *line width*, after each line of text, and when an "L" specifier is used in a PRINT USING statement. Up to eight characters may be specified as the EOL characters; an error is reported if the string contains more than eight characters. If END is included in the EOL attribute, an interface-dependent END indication is sent with the last character of the EOL sequence. If DELAY is included, the computer delays the specified number of seconds (after sending the last character) before continuing. The default EOL sequence consists of a carriage-return and a line-feed character with no END indication and no delay period. END and DELAY are ignored for files.

The WIDTH Attribute (Requires IO)

The WIDTH attribute specifies the maximum number of characters which will be sent to the printing device before an EOL sequence is automatically sent. The EOL characters are not counted as part of the line width. The default width for the Model 226 CRT is 50, Model 237 and other high-resolution displays is 128, and the default for all other devices is 80. Specifying WIDTH OFF sets the width to infinity. If the default is desired, it must be restored explicitly. If the USING clause is included the PRINT statement, the WIDTH attribute is ignored. Default WIDTH for files is OFF.

PRINTER IS file

The file must be a BDAT or HP-UX file.

The PRINTER IS file statement positions the file pointer to the beginning of the file.

The file is closed when another PRINTER IS statement is executed and at SCRATCH A.

You can read the file with ENTER if it is ASSIGNed with FORMAT ON.

An end-of-file error occurs when the end of a LIF file is reached.

SRM and HFS Files

In order to write to a PRINTER IS file on an HFS volume, you need to have R (read) and W (write) permission on the file, and X (search) permission on all superior directories.

In order to write to a PRINTER IS file on an SRM volume, you need to have READ and WRITE capabilities on the immediately superior directory, as well as READ capabilities on all other superior directories.

No end-of-file error occurs when writing to a file on an SRM or HFS volume because these files are extensible. That is, if the data output to the file with this statement would otherwise overflow the file's space allocation, the BASIC system automatically allocates the additional space needed (provided the media contains enough unused storage space).

If the specified file is in the SRM printer spooler directory, is of type BDAT[1], and contains data, then the SRM system sends the data to the printer (after the file is closed) and then purges the file. You may close the file by executing another PRINTER IS statement, or a SCRATCH A or SCRATCH BIN command.

BASIC/UX Specifics

On HP-UX systems, the line-printer is a spooled device. Writing directly to the printer as 701 may interfere with other spooled output. It is recommended that printer output be directed to either a file or the line-printer spooler by, for example, the statement:

```
PRINTER IS "|lp"
```

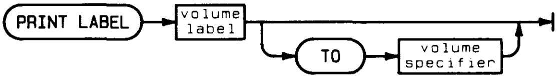
BASIC/UX treats output to a pipe as it would output to a file. The pipe must be explicitly closed before any output becomes permanent (or takes place). Output to a spooled device will not be sent to the spooler until the pipe has been closed. The closing of pipes can be achieved with a subsequent PRINTER IS, QUIT, or SCRATCH command.

[1] The SRM printer spooler will also spool ASCII files, which can be written by BASIC using OUTPUT, SAVE or RE-SAVE.

PRINT LABEL

Supported On	WS,UX
Option Required	MS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement gives a name to a mass storage volume.



Item	Description	Range
volume label	name to be given to the volume	—
volume specifier	string expression; Default=the default mass storage unit	(see MASS STORAGE IS)

Example Statements

```
PRINT LABEL "Vers3" TO ":INTERNAL,4,0"
PRINT LABEL Vol_label$ TO Vol_specifier$
```

Semantics

The new name overwrites any previous name for the volume.

The volume label can be zero to six characters in length consisting of letters and numbers. For maximum interchange, the characters should be limited to upper-case letters (A-Z) and digits (0-9) with the first character being a letter.

You cannot use PRINT LABEL with SRM volumes; instead, you will have to name the volume at the SRM console.

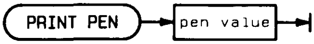
BASIC/UX Specifics

PRINT LABEL does not work in BASIC/UX for HFS.

PRINT PEN

Supported On	WS,UX
Option Required	CRTX
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement sets the pen color to be used in the output area and display line of the CRT.



Item	Description/Default	Range Restrictions
pen value	numeric expression	—

Example Statements

```
PRINT PEN Pen_value
PRINT PEN 143
IF Color_blue THEN PRINT 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, PRINT 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.

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

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

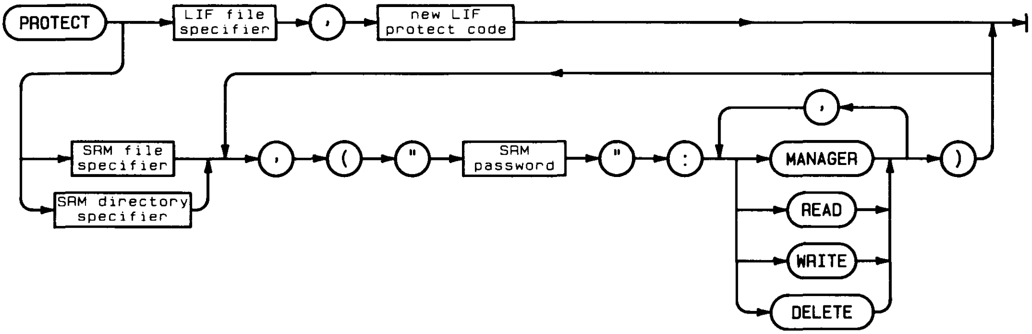
PRIORITY

See the SYSTEM PRIORITY statement.

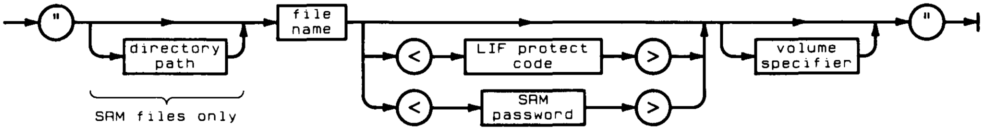
PROTECT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

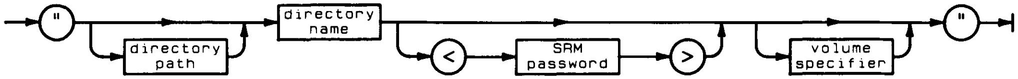
This statement specifies the protect code used on PROG, BDAT, and BIN files on LIF volumes. It also specifies passwords used on all types of files and directories on SRM volumes. (See PERMIT for access permissions of files and directories on HFS volumes.)



literal form of file specifier:



literal form of directory specifier:



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

Example Statements

```
PROTECT Name$,Lif_pc$
PROTECT "George<xy>:INTERNAL","NEW"

PROTECT "dir:REMOTE",("mgr":MANAGER),("rw":READ,WRITE)
PROTECT "File<rw>",("rw":DELETE)
```

Semantics

LIF Files

The protect code is necessary only for an operation which would write to the file or PURGE the file. The file can always be read without using the protect code (by LOAD, COPY, CAT "file name", etc.) The protect code is required for ASSIGN (and therefore ENTER) since ASSIGN opens a file for both read and write.

Protect codes are "trimmed" before they are used. Therefore, leading and trailing blanks are insignificant. Removing a protect code from a file is accomplished by assigning a protect code that is the null string or contains all blanks.

SRM Files (Requires SRM and DCOMM)

PROTECT allows you to control access to SRM files and directories by protecting access capabilities with password(s). Access capabilities are either public (available to all SRM users) or password-protected (available only to users supplying the correct password with the file or directory specifier).

The three access capabilities—MANAGER, READ and WRITE—are public until the PROTECT statement associates a password with one or more of those capabilities.

Once the capability on a given file or directory is password-protected, the capability can be exercised on the file or directory *only* if the correct password is included in the file or directory specifier. For instance, if a file's READ capabilities are protected, any user wishing to execute a command or statement that reads the file must supply a password protecting the file's READ capability.

MANAGER Access Capability (SRM)

Public MANAGER capability allows any SRM user to PROTECT, PURGE or RENAME a file or directory. Password-protected MANAGER capability provides READ and WRITE, as well as MANAGER, access capabilities to users who know the password.

You must have MANAGER capabilities on a file or directory to PROTECT the access capabilities on that file or directory. This includes adding, deleting and changing passwords.

READ Access Capability (SRM)

READ capability on a file allows use of commands and statements that read the contents of a file (for example: ENTER, LOAD, GET). READ capability on a directory allows you to read the files in the directory (CAT), or to "pass through" a directory by including the directory name (and password, if assigned) in a directory path.

WRITE Access Capability (SRM)

WRITE capability on a file allows use of commands and statements that write to the file (for example: OUTPUT, RE-SAVE, RE-STORE). WRITE capability on a directory allows use of commands that add or delete file names in the directory (for example: SAVE, STORE, PURGE, CREATE, RENAME).

Use of PROTECT on SRM

Each PROTECT statement allows up to six password/capability combinations per statement. The number of PROTECT statements that can be executed for each file or directory is unlimited, however, as long as each password is unique.

Successive associations of capabilities with the same password are not cumulative. To retain previous capability assignments for a file or directory, you must include those assignments in subsequent PROTECT statements designating the same password for that file or directory.

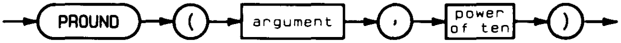
For example, say you protected the READ access capability on a file with the password `passme` then wanted to change that assignment so that `passme` would protect both the READ and WRITE access capabilities for that file. If you executed a second PROTECT statement associating `passme` with the WRITE capability only, *passme* would no longer protect the READ capability. Instead, you should specify the password and **both** the READ and WRITE capabilities in the second PROTECT statement.

To modify the access capabilities protected by a password, execute the PROTECT with the existing password and the new password/capability pair(s).

The secondary keyword DELETE is used to delete existing password assignments for a file or directory. To be effective, DELETE must be the only secondary keyword used with a password/capability pair in the PROTECT statement. Otherwise, DELETE is ignored. MANAGER capability is required to perform the DELETE. A DELETE executed without MANAGER capability results in a protect code violation error.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the value of the argument rounded to the specified power-of-ten.



Item	Description	Range
argument	numeric expression	
power of ten	numeric expression, rounded to an integer	

Example Statements

```
Money=PROUND(Result,-2)
PRINT PROUND(Quantity,Decimal_place)
```

Semantics

COMPLEX arguments are not allowed with this function.

PRT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This INTEGER function returns 701, the default (factory set) device selector for an external printer.

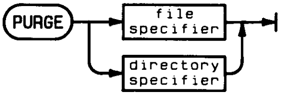
Example Statements

```
PRINTER IS PRT
OUTPUT PRT;A$
```

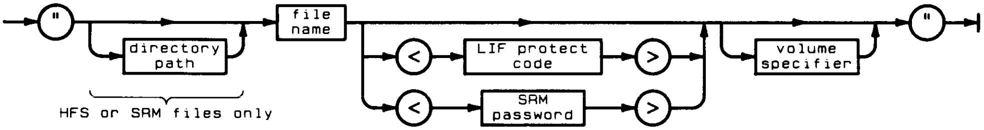
PURGE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

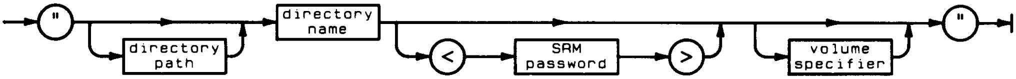
This statement deletes a file from a directory. On hierarchical-directory volumes (such as HFS and SRM), PURGE deletes an empty directory from its superior directory.



literal form of file specifier:



literal form of directory specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
directory 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)
directory name	literal	depends on volume's format (see Glossary)

Example Statements

```
PURGE File_name$
PURGE "File"
PURGE "George<PC>"
PURGE "Dir_a<SRM_RW_pass>/File<MGR_pass>"
PURGE "Dir1/Dir2/Dir3"
```

Semantics

Once a file is purged, you cannot access the information which was in the file. The records of a purged file are returned to "available space."

An open file must be closed before it can be purged. Any file except a PRINTER IS file, a PLOTTER IS file, or the current working directory can be closed by `ASSIGN TO *` (see ASSIGN). All files except those opened with the PRINTER IS statement are closed by RESET (SHIFT-PAUSE or Shift-Break). A PRINTER IS file can be closed by executing a PRINTER IS to another device or file. A PLOTTER IS file can be closed by GINIT or PLOTTER IS to another device or file. SCRATCH A also closes all files.

SRM and HFS Files and Directories

In order to PURGE an HFS or SRM directory or file, all of the following conditions must be met:

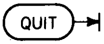
- It must be closed. The current working directory is closed by an MSI to a different directory. SCRATCH A closes all directories and files.
- It must be empty (directories only). That is, it must not contain any subordinate files or directories.
- You must have the appropriate access capabilities.
    - In order to PURGE a file or 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. Note that the ability to purge an HFS file is not determined by the file's permissions but rather by the permissions set on the parent directory.
    - In order to PURGE a file or directory on an SRM volume, you need to have M (manager) access capability on file or directory, as as well as R (read) and W (write) capabilities on the immediately superior directory and R capability on all superior directories.

Notes

QUIT

Supported On	UX
Option Required	n/a
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement cause BASIC to be exited.



Example Statements

```
QUIT
IF A$="DONE" THEN QUIT
```

Semantics

When used within a program, this statement stops the program, and then BASIC/UX exits.

When used as a keyboard command while a program is running, an error is given. You must first stop (or pause) the program before using the QUIT command.

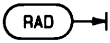
If a program is not running, then BASIC/UX is exited immediately.

Notes

RAD

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

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



Semantics

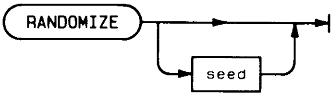
All functions which return an angle will return an angle in radians. All operations with parameters representing angles will interpret the angle in radians. If no angle mode is specified in a program, the default is radians (also see DEG).

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

RANDOMIZE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement selects a seed for the RND function.



Item	Description	Range
seed	numeric expression, rounded to an integer; Default = pseudo-random	1 thru $2^{31}-2$

Example Statements

```
RANDOMIZE
RANDOMIZE Old_seed*PI
```

Semantics

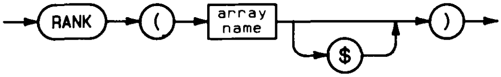
The seed actually used by the random number generator depends on the absolute value of the seed specified in the RANDOMIZE statement.

Absolute Value of Seed	Value Used
less than 1	1
1 thru $2^{31}-2$	INT(ABS(seed))
greater than $2^{31}-2$	$2^{31}-2$

The seed is reset to 37 480 660 by power-up, SCRATCH A, SCRATCH, and program prerun.

Supported On	WS,UX
Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the number of dimensions in an array. The value returned is an INTEGER.



Item	Description	Range
array name	name of an array	any valid name

Example Statements

```
•IF RANK(A)=2 THEN PRINT "A is a matrix"
R=RANK(Array)
```

RATIO

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

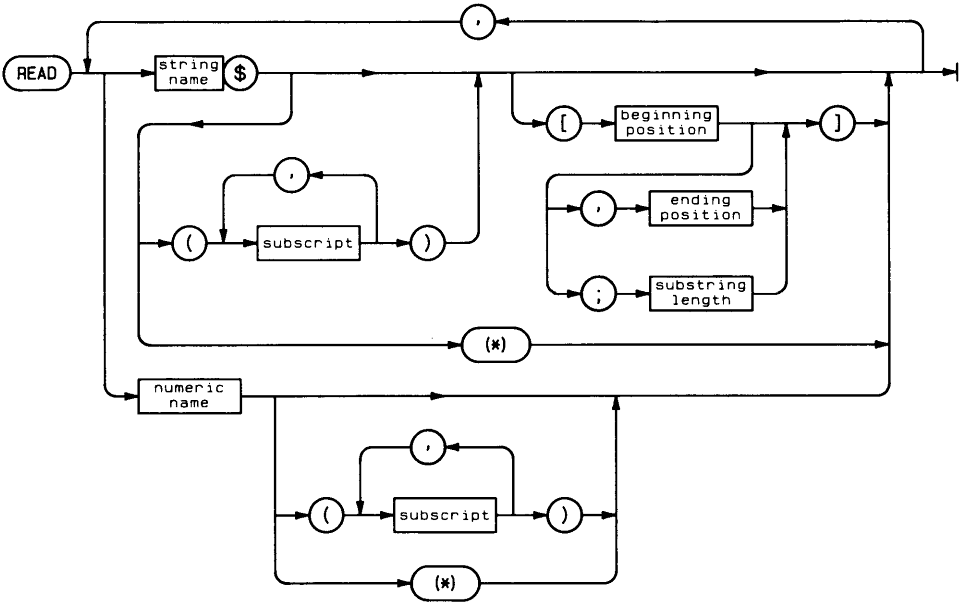
This function returns the ratio of the X hard clip limits to the Y hard clip limits for the current PLOTTER IS device.

Example Statements

```
WINDOW 0,10*RATIO,-10,10
Turn=1/RATIO
```

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement reads values from DATA statements and assigns them to variables.



Item	Description	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

```
READ Number,String$
READ Array(*)
READ Item(1,1),Item(2,1),Item(3,1)
```

Semantics

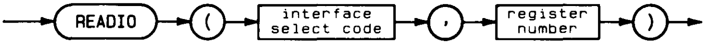
The numeric items stored in DATA statements are considered strings by the computer, and are processed with a VAL function to read into numeric variables in a READ statement. If they are not of the correct form, error 32 may result. Real DATA items will be rounded into an INTEGER variable if they are within the INTEGER range (−32 768 through 32 767). When a READ statement contains a COMPLEX variable, that variable is satisfied with two REAL values. A string variable may read numeric items, as long as it is dimensioned large enough to contain the characters.

The first READ statement in a context accesses the first item in the first DATA statement in the context unless RESTORE has been used to specify a different DATA statement as the starting point. Successive READ operations access following items, progressing through DATA statements as necessary. Trying to READ past the end of the last DATA statement results in error 36. The order of accessing DATA statements may be altered by using the RESTORE statement.

An entire array can be specified by replacing the subscript list with an asterisk. The array entries are made in row major order (right most subscript varies most rapidly).

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function reads the contents of the specified hardware register on the specified interface, or reads the specified byte or word of memory.



Item	Description	Range
select code	numeric expression, rounded to an integer	1 thru 31 and -31 thru -1; ±9826; 9827
register number or memory address	numeric expression, rounded to an integer	hardware-dependent

Note

Unexpected results may occur with select codes 9826 and 9827.

Example Statements

```
Upper_byte=READIO(Gpio,4)
PRINT "Register";I;"=";READIO(7,I)
Peek_byte=READIO(9826,Mem_addr)
Var_addr=READIO(9827,Integer_array)
```

Semantics

Positive select codes do a byte read (appropriate for most device registers); negative select codes do a word read.

Reading Memory ("Peek")

Select code 9826 is used to read a byte of memory, while −9826 is used to read a word (16 bits) of memory. The second parameter specified in the READIO function is the memory address of the byte to be read. This parameter is interpreted as a decimal address; for instance, an address of 100 000 is 10^5, not 2^20.

Determining the Location of Numeric Variables

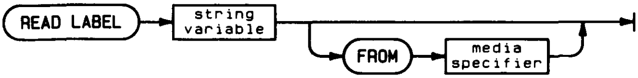
Select code 9827 is used to determine the memory address of a BASIC variable. You can use this address, for instance, with WRITEIO to perform a JSR ("Jump to SubRoutine") instruction in machine language, execute the instructions contained in the array, and then return to BASIC. (See WRITEIO for details.)

BASIC/UX Specifics

You are restricted to memory access within your own process space.

Supported On	WS,UX
Option Required	MS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement reads a volume label into a string variable.



Item	Description	Range
string variable	string variable which returns the volume name	—
volume specifier	string expression; Default = the default mass storage unit	(see MASS STORAGE IS)

Example Statements

```
READ LABEL Volume_name$ FROM ":INTERNAL,4,1"
IF Inserted$="Yes" THEN READ LABEL Vol_label$ FROM Vol_specifier$
```

Semantics

A LIF or HFS volume label consists of a maximum of 6 characters. SRM volumes can have labels up to 16 characters.

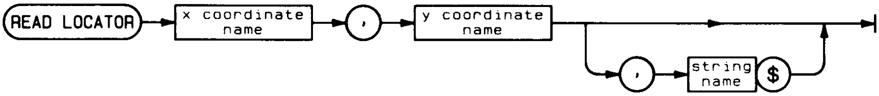
BASIC/UX Specifics

READ LABEL does not work for HFS in BASIC/UX.

READ LOCATOR

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement samples the locator device, without waiting for a digitizing operation.



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

Example Statements

```
READ LOCATOR X_pos,Y_pos
READ LOCATOR X,Y,Status$
```

Semantics

Executing this statement issues a request to the current locator device to return a set of coordinates. The coordinates are sampled immediately, without waiting for a digitizing action on the part of the user. GRAPHICS INPUT IS is used to establish the current locator device. The returned coordinates are in the unit-of-measure currently defined for the PLOTTER IS and GRAPHICS INPUT IS devices. The unit-of-measure may be default units or those defined by either the WINDOW or SHOW statement. If an INTEGER numeric variable is specified, and the value returned is out of range, Error 20 is reported.

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

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

Byte 1: Button status; This value represents the status of the digitizing button on the locator. A "0" means the button is not depressed, and a "1" means the button is depressed. This is an unprocessed value, and a "1" does not necessarily represent successful digitization. If the numeric value represented by this byte is used as the pen control value for a PLOT statement, continuous digitizing will be copied to the display device.

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

Byte 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.

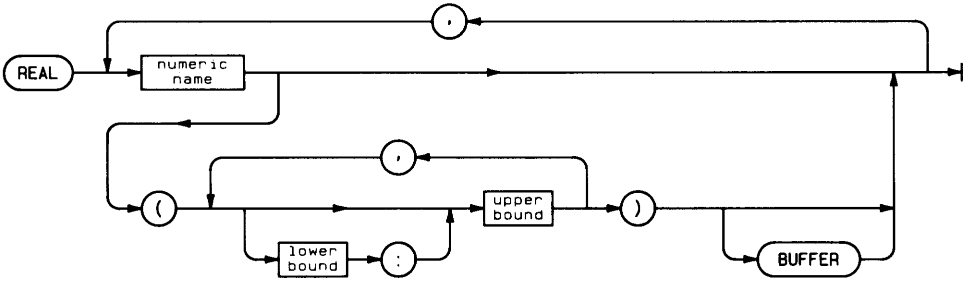
Bytes 7 and 8: The number of the buttons which are currently down. To interpret the ASCII number returned, change the number to its binary form and look at each bit. If the bit is "1", the corresponding button is down. If the bit is "0", the corresponding button is not down.

If the locator device (e.g., stylus or puck) goes out of proximity, a "button 7" is indicated in the "button number" bytes. The number will be exactly "64", regardless of whether any actual buttons are being held down at the time. The HP 9111A always returns "00" in bytes 7 and 8.

REAL

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	No

This statement reserves storage for floating-point variables and arrays. (For information about the REAL function, see the next entry in the keyword dictionary; for information about using REAL 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

```
REAL X,Y,Z
REAL Array(-128:127,15)
REAL A(512) BUFFER
```

Semantics

Each REAL variable or array element requires eight bytes of number storage. The maximum number of subscripts in an array is six, and no dimension may have more than 32 767 elements.

The total number of REAL 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).

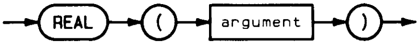
Declaring Buffers

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

REAL (function)

Supported On	WS,UX
Option Required	COMPLEX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the real part of a COMPLEX number.



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

Example Statements

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

Semantics

An INTEGER or REAL argument is returned unchanged.

RECORDS

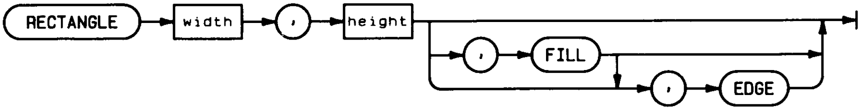
See the TRANSFER statement.

RECOVER

See the ON... statements.

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement draws a rectangle. It can be filled, edged, or both.



Item	Description	Range
width	numeric expression	—
height	numeric expression	—

Example Statements

```
RECTANGLE 4,6
RECTANGLE 3,-2,FILL,EDGE
```

Semantics

The rectangle is drawn with dimensions specified as displacements from the current pen position. Thus, both the width and the height may be negative.

Which corner of the rectangle is at the pen position at the end of the statement depends upon the signs of the parameters:

Sign of X	Sign of Y	Corner of Rectangle at Pen Position
+	+	Lower left
+	−	Upper left
−	+	Lower right
−	−	Upper right

Shape of Rectangle

A rectangle's shape is affected by the current viewing transformation. If isotropic units are in effect, the rectangle will be the expected shape, but if ansiotropic units are in effect, the rectangle will be distorted: stretched or compressed along the axes.

RECTANGLE is affected by the PIVOT and PDIR transformations. If a rotation transformation *and* anisotropic units are in effect, the rectangle is rotated first, then stretched or compressed along the unrotated axes.

FILL and EDGE

FILL causes the rectangle to be filled with the current fill color, and EDGE causes the perimeter to be drawn with the current pen color and line type. If both FILL and EDGE are specified, the interior will be filled, then the edge will be drawn. If neither FILL nor EDGE is specified, EDGE is assumed.

Rectangles sent to an HPGL plotter are edged but not filled regardless of any FILL or EDGE directives on the statement.

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

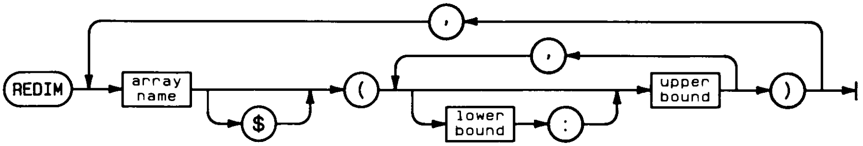
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

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

Note 4: RPLOT and IPLOT are affected by PDIR.

Supported On	WS,UX
Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement changes the subscript range of previously dimensioned arrays.



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

Example Statements

```
REDIM Array(5)
REDIM B(3:5,6,-2:2)
REDIM Constants$(X,Y,Z)
```

Semantics

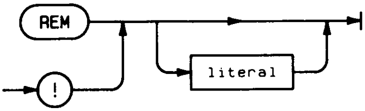
The following rules must be followed when redimensioning an array:

- The array to be redimensioned must have a currently dimensioned size known to the context (i.e., it must have been implicitly or explicitly dimensioned, or be currently allocated, or it must have been passed into the context.)
- You must retain the same number of dimensions as specified in the original dimension statement.
- The redimensioned array cannot have more elements than the array was originally dimensioned to hold.
- You cannot change the maximum string length of string arrays.

REDIM does not change any values in the array, although their locations will probably be different. The REDIM is performed left-to-right and if an error occurs, arrays to the left of the array the error occurs in will be redimensioned while those to the right will not be. If an array appears more than once in the REDIM, the right-most dimensions will be in effect after the REDIM.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	No

This statement allows comments in a program.



Item	Description	Range
literal	string constant composed of characters from the keyboard, including those generated with the ANY CHAR key	—

Example Program Lines

```
100  REM   Program Title
190  !
200  IF BIT(Info,2) THEN Branch  ! Test overrange bit
```

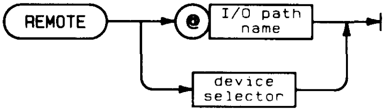
Semantics

REM must be the first keyword on a program line. If you want to add comments to a statement, an exclamation point must be used to mark the beginning of the comment. If the first character in a program line is an exclamation point, the line is treated like a REM statement and is not checked for syntax.

REMOTE

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement places HP-IB devices having remote/local capabilities into the remote state.



Item	Description	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

```
REMOTE 712
REMOTE @Hpib
```

Semantics

If individual devices are not specified, the remote state for all devices on the bus having remote/local capabilities is enabled. The bus configuration is unchanged, and the devices switch to remote if and when they are addressed to listen. If primary addressing is used, only the specified devices are put into the remote state.

When the computer is the system controller and is switched on, reset, or ABORT is executed, bus devices are automatically enabled for the remote state and switch to remote when they are addressed to listen.

The computer must be the system controller to execute this statement, and it must be the active controller to place individual devices in the remote state.

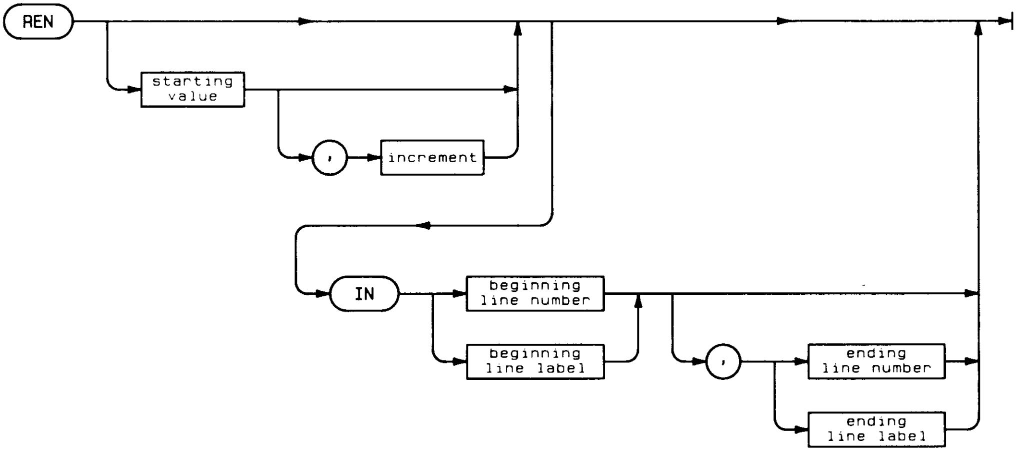
Summary of Bus Actions

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

REN

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	No
In an `IF...THEN`	No

This command allows you to renumber all or a portion of the program currently in memory.



Item	Description	Range
starting value	integer constant identifying a program line; Default = 10	1 thru 32 766
increment	integer constant; Default = 10	1 thru 32 767
beginning line number	integer constant identifying program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
REN
REN 1000,5
REN 270,1 IN 260,Label1
```

Semantics

The program segment to be renumbered is delimited by the beginning line number or label (or the first line in the program) and the ending line number or label (or the last line in the program). The first line in the renumbered segment is given the specified starting value, and subsequent line numbers are separated by the increment. If a renumbered line is referenced by a statement (such as GOTO or GOSUB), those references will be updated to reflect the new line numbers. Renumbering a paused program causes it to move to the stopped state.

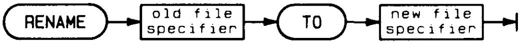
REN cannot be used to move lines. If renumbering would cause lines to overlap preceding or following lines, an error occurs and no renumbering takes place.

If the highest line number resulting from the REN command exceeds 32 766, an error message is displayed and no renumbering takes place. An error occurs if the beginning line is after the ending line, or if one of line labels specified doesn't exist.

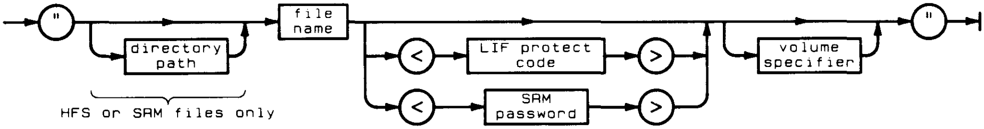
RENAME

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

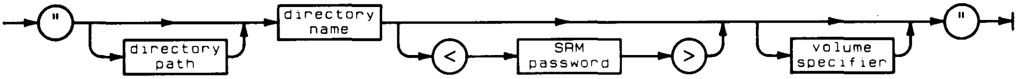
This statement changes a file's or directory's name.



literal form of file specifier:



literal form of directory specifier:



Item	Description	Range
old file specifier	string expression	(see "file specifier" drawing)
new file specifier	string expression	(see "file specifier" drawing)
old directory specifier	string expression	(see "directory specifier" drawing)
new directory specifier	string expression	(see "directory specifier" drawing)
directory path	literal	(see MASS STORAGE IS)

file name	literal	depends on volume's format:
		10 characters for LIF;
		14 characters for HFS (short file name);
		255 characters for HFS (long file name);
		16 characters for SRM;
		(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)

directory name	literal	depends on volume's format:
		10 characters for LIF;
		14 characters for HFS (short file name);
		255 characters for HFS (long file name);
		16 characters for SRM;
		(see Glossary)

Example Statements

```
RENAME "Old_name" TO "New_name"
RENAME File_name$&Vol$ TO Temp$
RENAME "TEMP<pc>" TO "FINAL"

RENAME Dir$&File$&Volume$
RENAME "/WORKSTATIONS/AUTOST" TO "old_autost"
RENAME "Dir1<SRM_RW_pass>/F1<MGR_pass>" TO "Dir2<RW_pass>/F1"
RENAME "Dir1/Dir2/MoveFile:REMOTE" TO "./Dir3/ToOtherDir"
```

Semantics

The new file or directory name must not duplicate the name of any other file in the directory.

SRM files and directories must be closed before being renamed.

- Files are closed by ASSIGN...TO * (explicitly closes an I/O path). All files except those opened with the PRINTER IS statement are also closed by RESET (SHIFT-PAUSE or Shift-Break). A PRINTER IS file can be closed by executing a PRINTER IS to another device or file. A PLOTTER IS file can also be closed by GINIT or PLOTTER IS to another device or file.
- The current working directory is closed by an MSI to a different directory.

SCRATCH A also closes all files and directories.

Because you cannot move a file from one mass storage volume to another with RENAME, an error will be given if a volume specifier is included which is not the current location of the file. (However, RENAME can perform limited file-move operations with SRM and HFS files. See details below.)

LIF Protect Codes

A protected file retains its old protect code, which must be included in the old file specifier.

HFS Permissions

In order to RENAME a file or 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.

SRM Passwords

In order to RENAME an SRM file or directory, you need to have M (manager) access capability on the file or directory, R (read) and W (write) capabilities on the immediately superior directory, and R capabilities on all other superior directories.

Including an SRM password in the file or directory specifier does not protect it. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but passwords in the "new file/directory name" portion of the RENAME statement are ignored. However, any existing SRM password is retained by the renamed file or directory.

SRM File and Directory Specifier Length

A maximum of nine names (files or directories) are allowed in *both* file or directory specifiers in the RENAME statement. (The number of names in the old file/directory specifier plus the number of names in the new file/directory specifier must not exceed nine.) No more than six names are allowed in either file specifier individually.

Limited File Moves with SRM and HFS

With SRM and HFS, RENAME can be used to move files within the directory structure. Directories cannot be moved with RENAME. Moving of files must occur within a single volume. If you move a file with RENAME, the original file ("old file specifier") is purged.

BASIC/UX Specifics

RENAMEing across volumes is allowed.

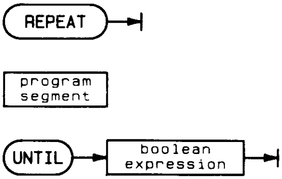
REORDER

See the MAT REORDER statement.

REPEAT...UNTIL

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	No

This construct defines a loop which is repeated until the boolean expression in the UNTIL statement evaluates to be logically true (evaluates to non-zero).



Item	Description	Range
boolean expression	numeric expression; evaluated as true if non-zero and false if zero	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested constructs(s).	—

Example Program Segments

```
530  REPEAT
540    PRINT Factor
550    Factor=Factor*1.1
560  UNTIL Factor>10

680  REPEAT
690    INPUT "Enter a positive number",Number
700  UNTIL Number>=0
```

Semantics

The REPEAT...UNTIL construct allows program execution dependent on the outcome of a relational test performed at the **end** of the loop. Execution starts with the first program line following the REPEAT statement, and continues to the UNTIL statement where a relational test is performed. If the test is false a branch is made to the first program line following the REPEAT statement.

When the relational test is true, program execution continues with the first program line following the UNTIL statement.

Branching into a REPEAT...UNTIL construct (via a GOTO) results in normal execution up to the UNTIL statement, where the test is made. Execution will continue as if the construct had been entered normally.

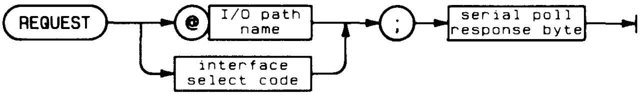
Nesting Constructs Property

REPEAT...UNTIL constructs may be nested within other constructs provided the inner construct begins and ends before the outer construct can end.

REQUEST

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used by a non-active controller to send a Service Request (SRQ) on an HP-IB interface.



Item	Description	Range
I/O path name	name assigned to an HP-IB interface	any valid name
interface select code	numeric expression, rounded to an integer	7 thru 31
serial poll response byte	numeric expression, rounded to an integer	0 thru 255

Example Statements

```
REQUEST @Hp_ib;Bit_6+Bit_0
REQUEST Isc;Response
```

Semantics

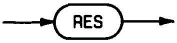
To request service, the value of the serial poll response must have bit 6 set; this bit asserts the SRQ line. SRQ will remain set until either the Active Controller performs a Serial Poll or until the computer executes another REQUEST with bit 6 clear.

Only the interface select code may be specified to receive the Request; if a device selector that contains address information, or an I/O path assigned to a device selector with address information is specified, an error results. An error will also results if the computer is currently the Active Controller.

RES

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the result of the last numeric computation which was executed from the keyboard.



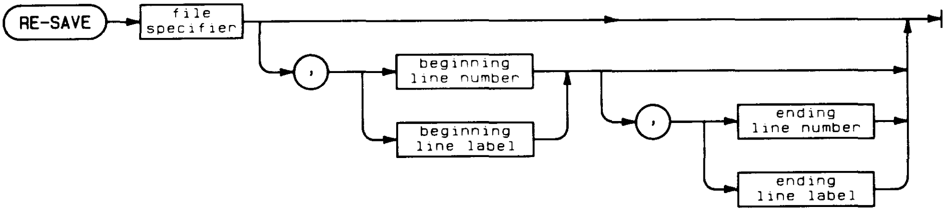
Example Statements

```
RES
3.5*RES+A
```

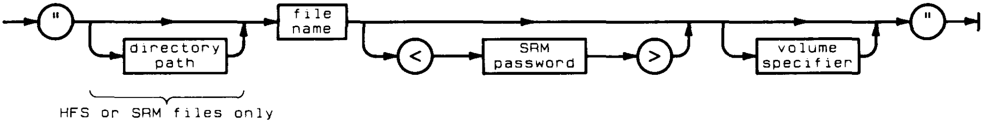
RE-SAVE

Supported On	WS,UX
Option Required	Edit
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement creates a specified ASCII file if it does not exist; otherwise, it re-writes a specified ASCII or HP-UX file by copying program lines as strings into that file.



literal form of file specifier:



Item	Description	Range
file specifier	string expression	(see drawing)
beginning line number	integer constant identifying program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name
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

```
RE-SAVE "NailFile"
RE-SAVE Name$,1,Sort
RE-SAVE "Dir<SRM_RW_pass>/File<SRM_RW_pass>"
```

Semantics

An entire program can be saved, or the portion delimited by beginning and (if needed) ending line labels or line numbers. If the file name already exists, the old file entry is removed from the directory after the new file is successfully saved on the mass storage media. Attempting to RE-SAVE any existing file that is not an ASCII or HP-UX text file results in an error. (Note that if you RE-SAVE an existing HP-UX text file, a new HP-UX file will be created; otherwise, an ASCII file will be created.)

If the file does not already exist, RE-SAVE performs the same action as SAVE.

Pressing RESET during a RE-SAVE operation results in the old file being retained.

If a specified line label does not exist, error 3 occurs. If a specified line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs.

Note that both hard and symbolic links to a file are broken by RE-SAVE (see LINK).

HFS Permissions

In order to RE-SAVE a file on an HFS volume, you need to have W (write) permission on the file (if one already exists), W (write) and X (search) permission of the immediately superior directory, as well as X permission on all other superior directories. If a file already exists, its permission bits will be preserved.

SRM Access Capabilities

In order to RE-SAVE an SRM file, you need to have R (read) and W (write) access capabilities on the file (if one already exists), R and W capabilities on the immediately superior directory, and R capabilities on all other superior directories.

If the file exists and is read/write protected, you must specify the correct password with RE-SAVE. If you specify the wrong password on a protected file, the system returns an error. Any existing SRM password is retained by the re-saved file.

If the file does not exist, including an SRM password with the file name 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 file name portion of the RE-SAVE statement will be ignored.

RE-SAVE on SRM Files

RE-SAVE opens the remote file in exclusive mode (denoted as `LOCK` in a CAT listing) and enforces that status on the file until the RE-SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-SAVE.

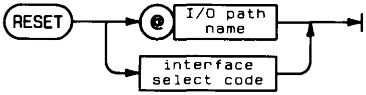
Use of RE-SAVE on SRM and HFS may leave temporary files on the mass storage media if [CLR I/O] ([Break]) or [RESET] ([SHIFT]-[PAUSE] or [Shift]-[Break]) is pressed or a TIMEOUT occurs during the RE-SAVE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

BASIC/UX Specifics

The temporary file name begins with `rmbt` followed by a letter and the BASIC/UX process id.

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement resets an interface or the pointers of either a mass storage file or buffer. (For information about RESET as a secondary keyword, see the SUSPEND INTERACTIVE statement.)



Item	Description	Range
I/O path name	name assigned to an interface, mass storage file, or buffer	any valid name
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
RESET Hpib
RESET 20
RESET @Buffer_x
```

Semantics

A RESET directed to an interface initiates an interface-dependent action; see the "Interface Registers" section for further details.

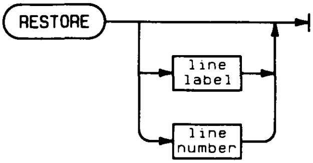
A RESET directed to a mass storage file resets the file pointer to the beginning of the file.

A RESET directed to a buffer resets all registers to their initial values: the empty and fill pointers are set to 1, and the current-number-of-bytes and all other registers are reset to zero.

If a TRANSFER is currently being made to or from the specified resource, the computer waits until the TRANSFER is complete before executing the RESET. If the TRANSFER is not to be completed, an ABORTIO may be executed to halt the TRANSFER before executing the RESET. If a busy buffer is specified in a RESET statement, error 612 results.

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	Yes

RESTORE specifies which DATA statement will be used by the next READ operation.



Item	Description	Range
line label	name of a program line	any valid name
line number	integer constant identifying a program line; Default = first DATA statement in context	1 thru 32 766

Example Statements

```
RESTORE
RESTORE Third_array
```

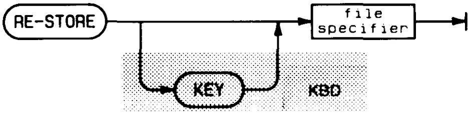
Semantics

If a line is specified which does not contain a DATA statement, the computer uses the first DATA statement after the specified line. RESTORE can only refer to lines within the current context. An error results if the specified line does not exist.

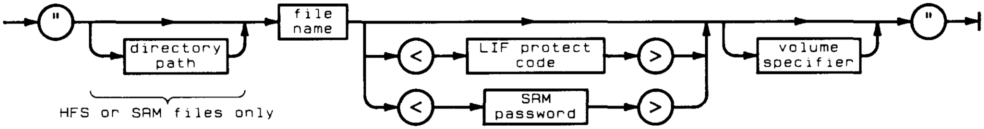
RE-STORE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement creates a file and stores the program or typing-aid softkey definitions in it.



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)

Example Statements

```
RE-STORE Filename$&Volume$
RE-STORE "Prog_a"
RE-STORE "Dir<SRM_RW_pass>/Prog_z<SRM_RW_pass>"

RE-STORE KEY "Typing_aids"
RE-STORE KEY "KEYS:REMOTE"
```

Semantics

If the specified file already exists, the old file is removed from the directory after the new file is successfully stored in the current mass storage device. If an old file does not exist, a new one is created as if this were the STORE statement.

Pressing Reset during a RE-STORE operation causes the old file to be retained. (See note below for effects on an SRM system.)

LIF Protect Codes

If the old file had a protect code, the same protect code must be used in the RE-STORE operation. Attempting to RE-STORE a file which is the wrong type results in an error. (RE-STORE creates a PROG file, and RE-STORE KEY creates a BDAT file.)

HFS Permissions

In order to RE-STORE a file on an HFS volume, you need to have W (write) permission on the file (if one already exists), W (write) and X (search) permission of the immediately superior directory, as well as X permission on all other superior directories. If the file already exists, its permission bits will be preserved.

SRM Access Capabilities

In order to RE-STORE an SRM file, you need to have R (read) and W (write) access capability on the file (if one already exists), R (read) and W (write) capabilities on the immediately superior directory, and R capability on all other superior directories.

If the file exists and is read/write protected, you must specify the correct password with RE-STORE. If you specify the wrong password on a protected file, the system returns an error. Any existing SRM password is retained by the re-saved file.

If the file does not exist, including an SRM password with the file name 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 file name portion of the RE-STORE statement will be ignored.

RE-STORE with SRM Volumes

RE-STORE opens an SRM file in exclusive mode (denoted as `LOCK` in a CAT listing) and enforces that status on the file until the RE-STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-STORE.

Use of RE-STORE on SRM or HFS may leave temporary files on the mass storage media if CLR I/O (Break) or RESET is pressed or a TIMEOUT occurs during the RE-STORE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

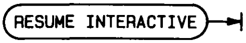
BASIC/UX Specifics

The temporary file name begins with `rmbt` followed by a letter and the BASIC/UX process id.

RESUME INTERACTIVE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes[1]
Programmable	Yes
In an IF...THEN	Yes

This statement enables the EXECUTE, ENTER, Return, PAUSE, STOP, STEP, CLR I/O, Break and RESET keys after a SUSPEND INTERACTIVE statement.



Example Statements

```
RESUME INTERACTIVE
IF Kbd_flag THEN RESUME INTERACTIVE
```

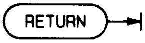
[1] This statement is executable from the keyboard, but only while SUSPEND INTERACTIVE is **not** in effect.

RETURN

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

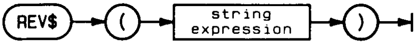
This statement returns program execution to the line following the invoking GOSUB. The keyword RETURN is also used in user-defined functions (see DEF FN).

See also ERROR RETURN.



Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns a string formed by reversing the sequence of characters in the specified string.



Example Statements

```
Reverse$=REV$("palindrome")
Last_blank=LEN(Sentence$)-POS(REV$(Sentence$)," ")
```

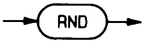
Semantics

The REV$ function is useful when searching for the last occurrence of an item within a string.

RND

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns a pseudo-random number greater than 0 and less than 1.



Example Statements

```
Percent=RND*100
IF RND<.5 THEN Case1
```

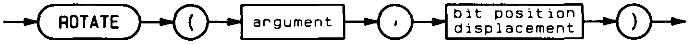
Semantics

The random number returned is based on a seed set to 37 480 660 at power-on, SCRATCH, SCRATCH A, or program prerun. Each succeeding use of RND returns a random number which uses the previous random number as a seed. The seed can be modified with the RANDOMIZE statement.

ROTATE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument by the number of bit positions specified. The shift is performed with wrap-around.



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

Example Statements

```
New_word=ROTATE(Old_word,2)
Q=ROTATE(Q,Places)
```

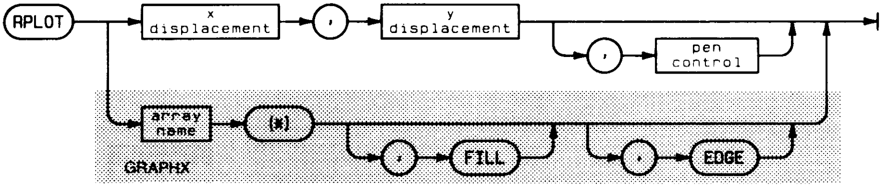
Semantics

The argument is converted into a 16-bit, two's-complement form. If the bit position displacement is positive, the rotation is towards the least-significant bit. If the bit position displacement is negative, the rotation is towards the most-significant bit. The rotation is performed without changing the value of any variable in the argument.

RPLOT

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement moves the pen from the current pen position to the point specified by adding the x and y displacements to the local origin. It can be used to move with or without drawing a line depending on the pen control parameter.



Item	Description	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	−32 768 thru +32 767
array name	name of two-dimensional, two-column or three-column numeric array. Requires GRAPHX	any valid name

Example Statements

```
RPLOT Rel_x,Rel_y,Pen_action
RPLOT 5,12
RPLOT Shape(*),FILL,EDGE
```

Semantics

This statement moves the pen to the specified X and Y coordinates relative to the local coordinate origin. Both moves and draws may be generated, depending on the pen control parameter. Lines are drawn using the curren pen color and line type.

The local coordinate origin is the logical pen position at the completion of one of the following statements. The local coordinate origin is **not** changed by the RPLOT statement.

AXES DRAW FRAME GINIT GRID IDRAW IMOVE
IPLOT LABEL MOVE PLOT POLYGON POLYLINE RECTANGLE
SYMBOL

The line is clipped at the current clipping boundary. RPLOT is affected by the PIVOT and PDIR transformations. If none of the line is inside the current clip limits, the pen is not moved, but the logical pen position is updated.

Non-Array Parameters

The specified X and Y displacements information is interpreted according to the current unit-of-measure. Lines are drawn using the current pen color and line type.

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

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

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

Note 4: RPLOT and IPLOT are affected by PDIR.

The optional pen control parameter specifies the following plotting actions; the default value is +1 (down after move).

Pen Control Parameter

Pen Control	Resultant Action
−Even	Pen up before move
−Odd	Pen down before move
+Even	Pen up after move
+Odd	Pen down after move

The above table is summed up by: even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Array Parameters

When using the RPLOT statement with an array, either a two-column or a three-column array may be used. If a two-column array is used, the third parameter is assumed to be +1; pen down after move.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the range 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the RPLOT statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the RPLOT statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the RPLOT statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

When using a RPLOT 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.

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 RPLOT statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens

An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types

An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color

Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation selector 14 is used in conjunction with operation selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessarily immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word; that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities (which range from 0 thru 1) for red, green, and blue in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored, and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME

Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the RPLOT statement, so one probably would not have more than one operation selector 12 in an array to RPLOT, since the last FRAME will overwrite all the previous ones.

Premature Termination

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

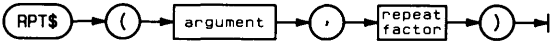
Ignoring Selected Rows in the Array

Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater that fifteen is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

RPT$

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the string repeated a given number of times.



Item	Description	Range
argument	string expression	—
repeat factor	numeric expression, rounded to an integer	0 thru 32 767

Example Statements

```
PRINT RPT$("*",80)
Center$=RPT$(" ",(Right-Left-Length)/2)
```

Semantics

The value of the numeric expression is rounded to an integer. If the numeric expression evaluates to a zero, a null string is returned.

An error will result if the numeric expression evaluates to a negative number or if the string created by RPT$ contains more than 32 767 characters.

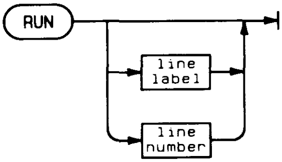
RSUM

See the MAT statement.

RUN

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	No
In an IF...THEN	No

This command starts program execution at a specified line.



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

Example Statements

```
RUN 10
RUN Part2
```

Semantics

Pressing the RUN key is the same as executing RUN with no label or line number. RUN is executed in two phases: prerun initialization and program execution.

The prerun phase consists of:

- Reserving memory space for variables specified in COM statements (both labeled and blank). See COM for a description of when COM areas are initialized.
- Reserving memory space for variables specified by DIM, REAL, COMPLEX, INTEGER, or implied in the main program segment. This does not include variables used with ALLOCATE, which is done at run-time. Numeric variables are initialized to 0; string variables are initialized to the null string.
- Checking for syntax errors which require more than one program line to detect. Included in this are errors such as incorrect array references, and mismatched parameter or COM lists.

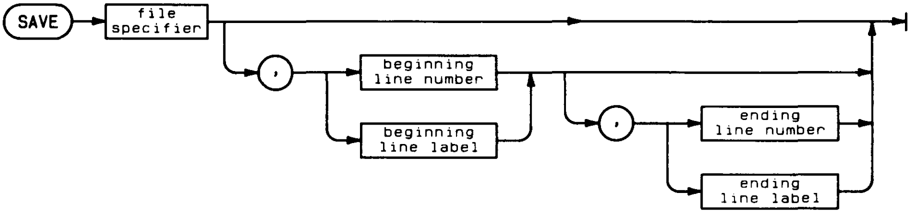
If an error is detected during prerun phase, prerun halts and an error message is displayed on the CRT.

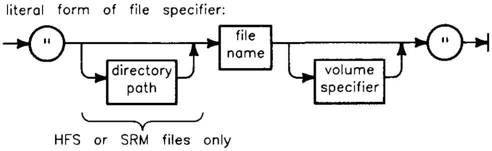
After successful completion of prerun initialization, program execution begins with either the lowest numbered program line or the line specified in the RUN command. If the line number specified does not exist in the main program, execution begins at the next higher-numbered line. An error results if there is no higher-numbered line available within the main program, or if the specified line label cannot be found in the main program.

Notes

Supported On	WS,UX
Option Required	EDIT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement creates an ASCII file and copies program lines as strings into that file.





Item	Description	Range
file specifier	string expression	(see drawing)
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name
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)

Example Statements

```
SAVE "WHALES"
SAVE "TEMP",1,Sort
SAVE "Dir<SRM_RW_pass>/File"
SAVE "Ascii_file:REMOTE"
```

Semantics

An entire program can be saved, or any portion delimited by the beginning and (if needed) ending line numbers or labels. This statement is for creating new files. Attempting to SAVE a file name that already exists causes error 54. If you need to replace an old file, see RE-SAVE.

If a specified line label does not exist, error 3 occurs. If a specified line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs. If no program lines are in the specified range, error 46 occurs.

HFS Permissions and File Headers

In order to SAVE 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 permission on all other superior directories.

When a file is saved on an HFS volume, access permission bits are set to RW-RW-RW-. You can modify the access permission bits with PERMIT if desired. For BASIC/UX, these permissions are subject to alteration by the user's `umask` value, if set. See the *HP-UX Reference*, umask(1).

On HFS volumes, SAVE creates an ASCII file that contains a 512-byte header (at the beginning of the file's contents). This header allows the BASIC system to recognize the file as being an ASCII file. (The header is handled automatically by the BASIC system, so you do not have to take any special actions.)

SRM Passwords and Exclusive Mode

In order to SAVE an SRM file, you need to have R (read) and W (write) capabilities on the immediately superior directory, and R capabilities on all other superior directories.

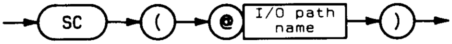
Including an SRM password with the file name does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but a password in the file name portion of the SAVE statement will be ignored.

SAVE opens an SRM file in exclusive mode (denoted as `LOCK` in a CAT listing) and enforces that status on the file until the SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the SAVE.

SC

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the interface select code associated with an I/O path name.



Item	Description	Range
I/O path name	name of a currently assigned I/O path	any valid name

Example Statements

```
Isc=SC(@Device)
Drive_isc=SC(@File)
```

Semantics

If the I/O path name is assigned to a device selector (or selectors) with primary and/or secondary addressing, only the interface select code is returned. If the specified I/O path name is assigned to a mass storage file, the interface select code of the drive is returned. If the specified I/O path name is assigned to a buffer, a zero is returned.

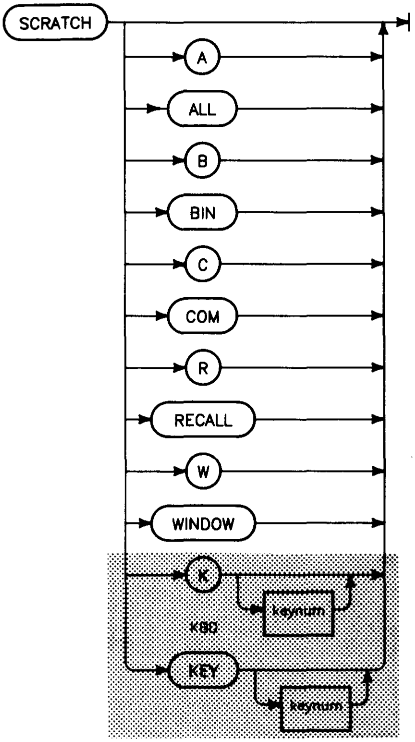
If the I/O path name is not currently assigned to a resource, an error is reported.

BASIC/UX Specifics

If the I/O path name refers to a file on an HFS disk, SC returns the constant value 701.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	No
In an `IF...THEN`	No

This command erases all or selected portions of memory.



Item	Description	Range
key number	integer constant	0 thru 23

Example Statements

```
SCRATCH
SCRATCH A
SCRATCH ALL       (BASIC/UX only)
SCRATCH KEY
SCRATCH KEY 21
SCRATCH WINDOW    (BASIC/UX under X Windows only)
```

Semantics

The BASIC Workstation (BASIC/WS) does not support the following secondary keywords with the keyword SCRATCH:

```
ALL        RECALL
B          W
COM        WINDOW
```

Both full names and single character abbreviations for actions are accepted (BASIC/UX only).

SCRATCH clears the BASIC program and all variables not in COM. Key definitions are left intact.

SCRATCH C clears all variables, including those in COM. The program and keys are left intact.

SCRATCH R clears the RECALL key buffer.

To scratch a key, type **SCRATCH KEY**, followed by the key number, and press EXECUTE, ENTER, or Return. Also, pressing a softkey after typing **SCRATCH** will cause **SCRATCH KEY**, followed by the key number, to be displayed. When a key is specified, the definition for that key only is cleared. When an individual key is not specified, all key definitions are cleared. In either case, the program and all variables are left intact.

SCRATCH A clears the BASIC program memory, all the key definitions, and all variables (including those in COM). Most internal parameters in the computer are reset by this command. The clock is not reset and the recall buffer is not cleared. See the Master Reset Table in the "Useful Tables" section in the back of this manual for details.

SCRATCH BIN

SCRATCH BIN causes an extended SCRATCH A. It resets the computer to its power up state. All programs, variables, and BINs are deleted from memory. The BIN which contains the CRT driver for the current CRT is not deleted. Note that SCRATCH BIN will not remove any binaries that reside in ROM.

SCRATCH BIN and SCRATCH B are not supported on BASIC/UX.

SCRATCH A Effects on SRM and HFS Volumes

With SRM volumes, SCRATCH A releases the system resources allocated to the workstation executing the SCRATCH A, making those resources available to other SRM workstations. More specifically, SCRATCH A closes all files and directories, and resets the workstation's working directory to the root directory of the default volume (the mass storage volume from which the workstation booted). SCRATCH A also closes files and directories with HFS volumes.

If the workstation has Boot ROM version 3.0 or A or later, and booted from the SRM, SCRATCH A resets the working directory to the root of the default system volume. If the workstation has an earlier version Boot ROM, SCRATCH A resets the working directory to the device from which the workstation booted (for example, :INTERNAL if the workstation booted from a built-in drive).

SCRATCH W or SCRATCH WINDOW (BASIC/UX only)

In a windowing environment, this command causes all created windows to be destroyed. Note that this does not destroy the root BASIC window.

This command is only valid when running within a window system. When not in a window system, this command causes an error.

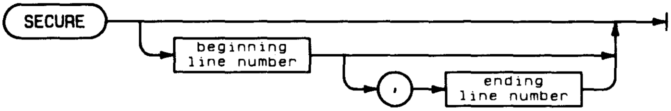
SEC

See the SEND statement

SECURE

Supported On	WS,UX
Option Required	PDEV
Keyboard Executable	Yes
Programmable	No
In an IF...THEN	No

This command protects program lines so that they cannot be listed.



Item	Description	Range
beginning line number	integer constant; Default = first line in program	—
ending line number	integer constant; Default = beginning line number if specified, or last line in program	—

Example Statements

```
SECURE
SECURE 45
SECURE 1,100
```

Semantics

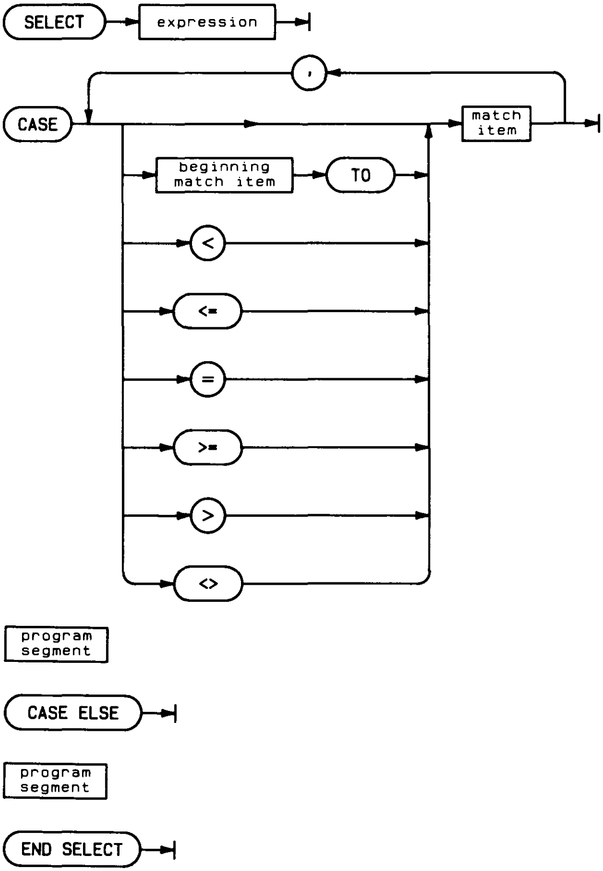
If no lines are specified, the entire program is secured. If one line number is specified, only that line is secured. If two lines are specified, all lines between and including those lines are secured.

Program lines which are secure are listed as an *. Only the line number is listed.

SELECT...CASE

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	No

This construct provides conditional execution of one of several program segments.



Item	Description	Range
expression	a numeric or string expression	—
match item	a numeric or string expression; must be same type as the SELECT expression	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
650  SELECT Expression
660    CASE <0
670      PRINT "Negative number"
680    CASE ELSE
690      PRINT "Non-negative number"
700  END SELECT
```

```
750  SELECT Expression$
760    CASE "A" TO "Z"
770      PRINT "Uppercase alphabetic"
780    CASE ":",";",",","."
790      PRINT "Punctuation"
800  END SELECT
```

Semantics

SELECT...END SELECT is similar to the IF...THEN...ELSE...END IF construct, but allows **several** conditional program segments to be defined; however, **only one segment will be executed** each time the construct is entered. Each segment starts after a CASE or CASE ELSE statement and ends when the next program line is a CASE, CASE ELSE, or END SELECT statement.

The SELECT statement specifies an expression, whose value is compared to the list of values found in each CASE statement. When a match is found, the corresponding program segment is executed. The remaining segments are skipped and execution continues with the first program line following the END SELECT statement.

All CASE expressions must be of the same type, (either string or numeric) and must agree in type with the corresponding SELECT statement expression.

The optional CASE ELSE statement defines a program segment to be executed when the selected expression's value fails to match any CASE statement's list.

Branching into a SELECT...END SELECT construct (via GOTO) results in normal execution until a CASE or CASE ELSE statement is encountered. Execution then branches to the first program line following the END SELECT statement.

Errors encountered in evaluating CASE statements will be reported as having occurred in the corresponding SELECT statement.

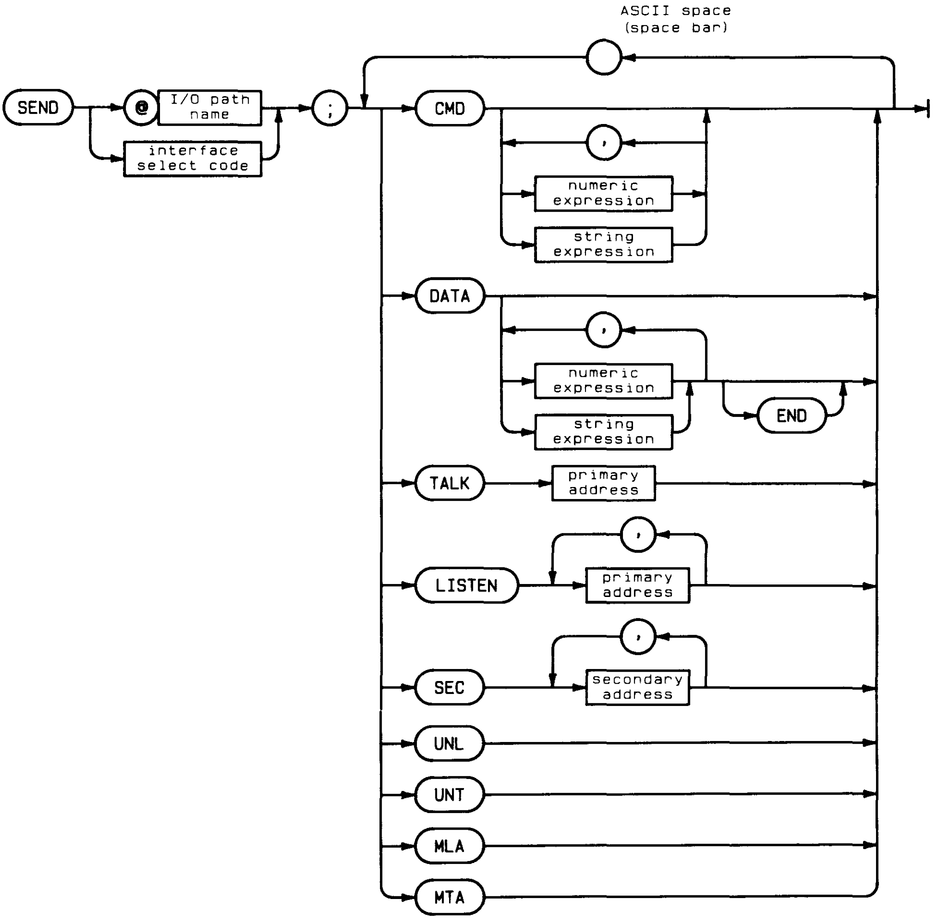
Nesting Constructs Properly

SELECT...END SELECT constructs may be nested, provided inner construct begins and ends before the outer construct can end.

SEND

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement sends messages to an HP-IB.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	7 thru 31
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
primary address	numeric expression, rounded to an integer	0 thru 31
secondary address	numeric expression, rounded to an integer	0 thru 31

Example Statements

```
SEND 7;UNL MTA LISTEN 1 DATA "HELLO" END
SEND @Hpib;UNL MLA TALK Device CMD 24+128
```

Semantics

CMD

The expressions following a CMD are sent with ATN true. The ASCII characters representing the evaluated string expression are sent to the HP-IB. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. CMD with no items sets ATN true.

DATA

The expressions following DATA are sent with ATN false. The ASCII characters representing the evaluated string expression are sent. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. If END is added to the data list, EOI is set true before sending the last byte. DATA with no items sets ATN false without waiting to be addressed as a talker.

If the computer is active controller, and addressed as a talker, the data is sent immediately. If the computer is not active controller, it waits until it is addressed to talk before sending the data.

TALK

TALK sets ATN true and sends the specified talk address. Only one primary address is allowed for a single talker. An extended talker may be addressed by using SEC secondary address after TALK. A TALK address of 31 is equivalent to UNT (untalk).

UNT

UNT sets ATN true and sends the untalk command. (There is no automatic untalk.) A TALK address of 31 is equivalent to UNT.

LISTEN

LISTEN sets ATN true, sends one or more primary addresses, and addresses those devices to listen. A LISTEN address of 31 is equivalent to UNL (unlisten).

UNL

UNL set ATN true and sends the unlisten command. (There is no automatic unlisten.) A LISTEN address of 31 is equivalent to UNL.

SEC

SEC sets ATN true and sends one or more secondary addresses (commands).

MTA

MTA sets ATN true and sends the interface's talk address. It is equivalent to performing a status sequence on the interface and then using the returned talk address with a SEND..TALK sequence.

MLA

MLA sets ATN true and sends the interface's listen address. It is equivalent to performing a status sequence on the interface and then using the returned listen address with a SEND..LISTEN sequence.

Summary

The computer must be the active controller to execute SEND with CMD, TALK, UNT, LISTEN, UNL, SEC, MTA and MLA.

The computer does not have to be the active controller to send DATA. DATA is sent when the computer is addressed to talk.

The following table lists the HP-IB message mnemonics, descriptions of the messages, and the secondary keywords required to send the messages. Any numeric values are decimal.

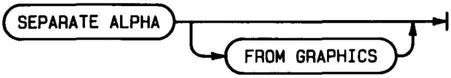
Mnemonic	Description	Secondary Keyword and Value
DAB	Data Byte	DATA 0 thru DATA 255
DCL	Device Clear	CMD 20 or CMD 148
EOI	End or Identify	DATA (data) END (sends EOI with ATN false, which is the END message; EOI with ATN true is the Identify message, sent automatically with the PPOLL function)
GET	Group Execute Trigger	CMD 8 or CMD 136
GTL	Go To Local	CMD 1 or CMD 129
IFC	Interface Clear	Not possible with SEND. An ABORT statement must be used.
LAG	Listen Address Group	LISTEN 0 thru LISTEN 31; or CMD 32 thru CMD 63
LLO	Local Lockout	CMD 17
MLA	My Listen Address	MLA
MTA	My Talk Address	MTA
PPC	Parallel Poll Configure	CMD 5 or CMD 133
PPD	Parallel Poll Disable	PPC (CMD 5 or CMD 133), followed by CMD 112; or CMD 240; or SEC 16.
PPE	Parallel Poll Enable	PPC (CMD 5 or CMD 133), followed by CMD 96 thru CMD 111; or CMD 224 thru CMD 239; or SEC 0 thru SEC 15 (SEC 0 allows a mask to be specified by a numeric value)
PPU	Parallel Poll Unconfigure	CMD 21 or CMD 149
PPOLL	Parallel Poll	Not possible with SEND. PPOLL function must be used.
REN	Remote Enable	Not possible with SEND. REMOTE statement must be used.
SDC	Selected Device Clear	CMD 4 or CMD 132

Mnemonic	Description	Secondary Keyword and Value
SPD	Serial Poll Disable	CMD 25 or CMD 153
SPE	Serial Poll Enable	CMD 24 or CMD 152
TAD	Talk Address	TALK 0 thru TALK 31, or CMD 64 thru CMD 95, or CMD 192 thru CMD 223.
TCT	Take Control	CMD 9 or CMD 137
UNL	Unlisten	UNL, or LISTEN 31, or CMD 63, or CMD 191.
UNT	Untalk	UNT, or TALK 31, or CMD 95, or CMD 223.

SEPARATE ALPHA FROM GRAPHICS

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used to simulate the separate alpha and graphics rasters of Series 200 displays (not valid in a windowing environment, such as X Windows).



Example Statements

```
SEPARATE ALPHA

IF (S_300 AND Multi_plane) THEN SEPARATE ALPHA FROM GRAPHICS
```

Semantics

This statement is used to set up the planes on multi-plane bit-mapped alpha displays for *independent* use as separate alpha and graphics rasters. (This is the way that Series 200 displays work.) If the display is a monochrome, 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* bit-mapped alpha with more than one plane (not monochrome), then the following actions are taken:
   a. The screen is cleared.
   b. The alpha mask is set (see table below for details).
   c. The alpha pen is set (see table below for details).
   d. All appropriate color map entries are initialized (see table below for details).
   e. The graphics mask is set so that it does not overlap with the alpha mask (the complement of the alpha mask).
   f. The alpha display is re-written in the new alpha color.

Display-Specific Parameters

Here are the values of parameters for the different types of Series 300 bit-mapped alpha displays:

Number of Planes	Alpha Mask	Color Map	Graphics Mask
4	Plane 4 (1000 base 2) Alpha pen is 8.	Pens 0 thru 7 have normal default values; pens 8 thru 15 are green.	Planes 1 thru 3 (0111 base 2) Graphics pens are 0 thru 7.

6	Planes 5 & 6
	(110000 base 2)
	Alpha pens are 16, 32, and 48.
		Pens 0 thru 15 have normal default values;
		pens 16 thru 31 are green;
		pens 32 thru 47 are brown;
		pens 48 thru 63 are cyan.
			Planes 1 thru 4
			(001111 base 2)
			Graphics pens are 0 thru 15.

8	Planes 7 & 8
	(11000000 base 2)
	Alpha pens are 64, 128, and 192.
		Pens 0 thru 63 have normal default values;
		pens 64 thru 127 are green;
		pens 128 thru 191 are brown;
		pens 192 thru 255 are cyan.
			Planes 1 thru 6
			(00111111 base 2)
			Graphics pens are 0 thru 63.

Color map entries below the lowest alpha pen value have their default colors set by `PLOTTER IS CRT,"INTERNAL"`. Using a value in this range as an alpha pen will produce transparent text (i.e., is equivalent to using pen 0). Setting up the color map as given in the table causes the alpha text to be dominant over graphics images. If the COLOR MAP option is used with PLOTTER IS, the SET PEN statement can still be used to set all color map entries, not just those dedicated to graphics pens.

Here is a BASIC program that performs similar configuration of the planes of a 4-plane display:

```
100   PLOTTER IS CRT, "INTERNAL";COLOR MAP ! Select Series 300 display.
110   FOR I=8 TO 15
120     SET PEN I INTENSITY 0,1,0          ! Set alpha colors (green).
130   NEXT I
140   ALPHA PEN 0                          ! Set alpha pen to black (temp.).
150   ALPHA MASK 15                        ! Enable all planes (temp.).
160   CLEAR SCREEN
170   ALPHA MASK 8                         ! Enable plane 4 for alpha.
180   ALPHA PEN 8                          ! Set alpha pen.
190   INTEGER Gm(0)                        ! Declare array for GESCAPE.
200   Gm(0)=7                              ! Set bits 2,1,0, which select
210   GESCAPE CRT,7,Gm(*)                  !  graphics planes 3,2,1.
220   ALPHA ON                             ! Display alpha plane.
230   GRAPHICS ON                          ! Display graphics planes.
240   PLOTTER IS CRT,"INTERNAL"            ! Return to non-color-map
250   END                                  !  mode.
```

Note that when using this operation with AREA COLOR and AREA INTENSITY, there may be unexpected results. The algorithm that AREA COLOR and AREA INTENSITY use to select graphics pens does not account for the graphics write-enable or display-enable masks. If the pens selected by these statements have bits outside of the write-enable mask, then the planes corresponding to these bits will not be affected. The result is that the area fill colors will not be what is expected.

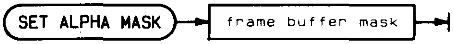
BASIC/UX Specifics

Does not work in a windowed environment.

SET ALPHA MASK

Supported On	WS,UX
Option Required	CRTX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used to specify which plane(s) can be modified by alpha display operations.



Item	Description/Default	Range Restrictions
frame buffer mask	numeric expression, rounded to an integer	1 thru 2^n − 1 where n equals the number of display planes

Example Statements

```
SET ALPHA MASK Frame_mask
SET ALPHA MASK 3
SET ALPHA MASK IVAL("1100",2)
IF Total_frames = 5 THEN SET ALPHA MASK 8
```

Semantics

This statement does not affect the operation of monochrome displays or the display of the Model 236C.

Setting bit 0 of the frame buffer mask (i.e. SET ALPHA MASK 1) enables alpha write permission to plane 1; setting bits 2 and 3 of the frame buffer mask (i.e. SET ALPHA MASK 12) enables write permission to planes 3 and 4. The masks you can use to enable write permissions range from 1 thru $2\hat{}n - 1$ where n is the number of display planes (e.g. the range of frame buffer masks for 4-planes would be 1 thru 15).

This statement affects any alpha display operation using the CRT (e.g. `PRINT`, `DISP`, `CAT`, error messages, etc.).

The difference between this statement and SET DISPLAY MASK is SET ALPHA MASK specifies which plane(s) can be *modified* by alpha operations (regardless of whether or not it/they are *displayed*). SET DISPLAY MASK specifies the plane(s) that are to be *displayed* (regardless of whether or not anything has been or can be *written* to it/them).

For further information on the alpha write-enable mask, read the section entitled "The Alpha Mask" in the chapter "Using Graphics Effectively" found in the *BASIC Graphics Techniques* manual.

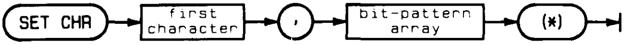
Note that the functionality of this statement can be achieved through CRT CONTROL register 18.

For more information related to this statement, see SEPARATE ALPHA and MERGE ALPHA which are found in this reference.

SET CHR

Supported On	WS,UX
Option Required	CRTX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement re-defines the bit-pattern used for character(s) in the current font (on bit-mapped alpha/graphics displays only).



Item	Description	Range
first character	numeric expression, rounded to an integer, which specifies the numeric code of the first character to be re-defined	0 thru 256
bit-pattern array	name of an INTEGER array	any valid name

Example Statements

```
ALLOCATE INTEGER Char_cell(1:CHRY,1:CHRX)
SET CHR Char_code,Char_cell(*)

ALLOCATE INTEGER Entire_font(1:Num_chars,1:CHRY,1:CHRX)
SET CHR 0,Entire_font(*)
```

Semantics

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 attempting to execute this statement results in error 880.

The "first character" parameter specifies the code of the first character whose bit-pattern is to be re-defined.

The "bit-pattern array" contains the actual pixels that are to comprise the new character. If the display is monochrome (single-plane), then only the low-order bit of each INTEGER element is used. If the display is color (multi-plane), then as many bits are used as there are planes in the display.

If the bit-pattern array parameter has only two dimensions, then only one character is re-defined. The first dimension must have a range of exactly the value of CHRY for this display; the second must have a range of CHRX. (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 bit-pattern array parameter has three dimensions, then multiple characters are re-defined beginning at the character specified by the "first character" parameter, and continuing until the array is exhausted (or character code 256 is reached, whichever occurs first). The first dimension of this array corresponds to the character's code, the second to the character-cell row, and the third to the character-cell column.

Note that character code 256 is the pattern which is exclusive OR'd with a character to produce underlined characters on the display. [Underlining is enabled by writing a CHR$(132) on the display, such as with PRINT, OUTPUT, or DISP.]

Restoring the Power-Up Default Font

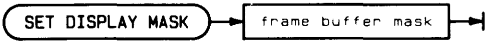
If you want to return to using the default font, then execute this statement:

```
CONTROL CRT,21;1
```

SET DISPLAY MASK

Supported On	WS,UX
Option Required	CRTX
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement is used to specify which plane(s) can be seen on the alpha display.



Item	Description/Default	Range Restrictions
frame buffer mask	numeric expression, rounded to an integer	0 thru 2^n − 1 where n equals the number of display planes

Example Statements

```
SET DISPLAY MASK Frame_mask
SET DISPLAY MASK 3
SET DISPLAY MASK IVAL("1100",2)
IF Disp_frames = 5 THEN SET DISPLAY MASK 8
```

Semantics

This statement does not affect the operation of monochrome displays or the display of the Model 236C.

Setting bit 0 of the frame buffer mask (i.e. SET DISPLAY MASK 1) enables the displaying of alpha plane 1; setting bits 2 and 3 of the frame buffer mask (i.e. SET DISPLAY MASK 12) enables displaying of alpha planes 3 and 4. The masks you can use to enable display range from 0 thru $2\hat{}n - 1$ where n is the number of display planes (e.g. the range of frame buffer masks for 4-planes would be 0 thru 15).

This statement affects any display operation using the CRT (e.g. **PRINT**, **DISP**, **CAT**, error messages, graphics, etc.).

The difference between this statement and SET ALPHA MASK is SET DISPLAY MASK specifies the plane(s) that are to be *displayed* (regardless of whether or not anything has been or can be *written* to it/them). SET ALPHA MASK specifies which plane(s) can be *modified* by alpha operations (regardless of whether or not it/they are *displayed*).

For further information on the display-enable mask, read the section entitled "The Alpha Mask" in the chapter "Using Graphics Effectively" found in the *BASIC Graphics Techniques* manual.

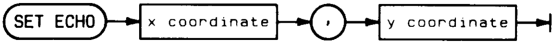
Note that the functionality of this statement can be achieved through CRT CONTROL register 20.

For more information related to this statement, see ALPHA ON/OFF, GRAPHICS ON/OFF, and GESCAPE found in this reference.

SET ECHO

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement sets an echo to the specified location on the current PLOTTER IS device.



Item	Description	Range
x coordinate	numeric expression in current units	—
y coordinate	numeric expression in current units	—

Example Statements

```
SET ECHO Xin,Yin
SET ECHO 1000,10000
```

Semantics

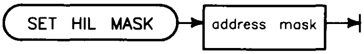
If the current PLOTTER IS device is a CRT, a 9-by-9-dot cross-hair is displayed at the specified coordinates if they are within the hard clip limits; the soft clip limits are ignored. No echo is displayed if the coordinates are outside the hard clip limits.

If the current PLOTTER IS device is an HPGL plotter, the pen is raised and moved to the specified coordinates if they are within the current clip limits. If the pen is inside the clip limits and the new echo position is not, it moves towards the new echo position but stops at the clip boundary. If the pen is outside the clip limits and the new echo position is outside the clip limits, the pen moves along the nearest clip boundary.

SET ECHO is frequently used with the READ LOCATOR statement.

Supported On	UX
Option Required	n/a
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement enables the specified HIL devices for use by the BASIC system.



Item	Description	Range
address mask	the sum of 2 raised to the power of each of the addresses of the desired devices	any even number from 0 to 254

Example Statements

```
SET HIL MASK 16
SET HIL MASK 2^Mouse+2^Knobbox1+2^Buttonbox2
```

Semantics

The address mask provides the capability of specifying the HIL devices to be used by the BASIC system. The most recent SET HIL MASK statement specifies the HIL devices which are used in subsequent ON KNOB, ON CDIAL, ON HIL EXT, and GRAPHICS INPUT IS statements. In addition, it specifies the devices which generate arrow keystrokes during live keyboard and editing when the devices are not being used by any of the above statements.

The value of the mask is obtained by raising 2 to the power of each of the addresses of the desired device, and adding these values. Suppose you want to create a mask which would only allow interrupts from HP-HIL devices at addresses 1 and 3. You would take 2 and raise if to the first power and add this result to 2 raised to the third power; the final result is a mask value of 10.

At start-up time, the BASIC system attempts to use all available devices on the HP-HIL link. You may then use this statement to select only those devices which you require and relinquish the other devices for use by different HP-UX processes (e.g. other BASIC/UX processes). You should never specify the address of the HIL keyboard with this statement since this interferes with the operation of BASIC and block all keyboard input.

Any HIL device which has been specified with this statement or which is not owned by other processes can be identified using the HIL SEND statement as in:

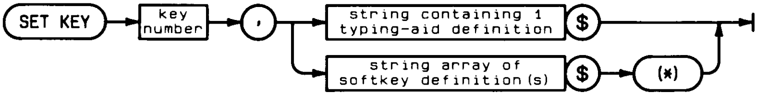
```
HIL SEND 4; IDD
```

You should note that the X Windows environment monopolizes all HIL devices unless explicitly specified not to do so. When a device is thus owned by X Windows, it is not available for use by any BASIC processes running under the environment.

SET KEY

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement programmatically re-defines typing-aid softkey(s).


SET KEY
key number
,
string containing 1 typing-aid definition
$
string array of softkey definition(s)
$
(*)

Item	Description	Range
key number	numeric expression, rounded to an integer	0 thru 23
string containing 1 softkey definition	string expression	any valid string expression
string array of softkey definition(s)	name of a string array	any valid name

Example Statements

```
SET KEY 1,OneKey$
SET KEY First_key,Several_keys$(*)
```

Semantics

Typing-aid softkeys are used when typing text at the keyboard. They are active whenever there is *not* a running program that has defined interrupt service routines for the keys (with ON KEY).

The "first key" parameter indicates the first key to be re-defined.

The second parameter (the string expression or array) determines the number of keys to be re-defined:

- If the parameter is a *string expression* (which includes a simple string variable), then *only one* typing-aid softkey is re-defined.
- If the parameter is a *string array*, then *several* typing-aid softkeys may be re-defined. Softkeys are re-defined in ascending order, one for each array element, until one of the following conditions is true:
    - the end of the array is reached
    - the last softkey is re-defined
    - typing-aid softkey memory overflows

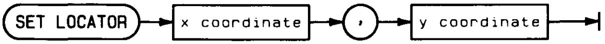
  For instance, if this parameter has a value of 5, and the string array has 3 elements, then softkeys f5, f6, and f7 are re-defined, respectively.

In order to minimize the chances of typing-aid memory overflows, keys in the range to be re-defined are first cleared and then the corresponding string values are placed into typing-aid memory. For instance, if the "first key" parameter is 3 and the array contains 4 elements, then softkeys 3 through 6 are cleared, after which the string array elements are placed into the corresponding softkeys. If typing-aid memory does overflow, the remaining keys in the range remain undefined. For instance, in this example if a memory overflow occurred while defining key 5, then keys 3 and 4 would have new definitions while keys 5 and 6 would remain undefined.

If the string, or string array element, contains a null (0 length), string, the corresponding typing-aid becomes undefined. Use EDIT KEY or LOAD KEY to define null string typing-aids.

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement specifies a new position for the locator of the current graphics input device.



Item	Description	Range
x coordinate	numeric expression specifying the x coordinate of the locator's new position in current units	range of REAL
y coordinate	numeric expression specifying the y coordinate of the locator's new position in current units	range of REAL

Example Statements

```
SET LOCATOR 12,95
SET LOCATOR X_cor,Y_cor
```

Semantics

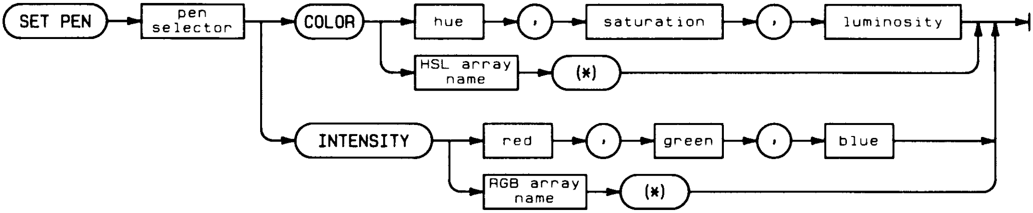
If any of the coordinates are outside the device's limits, they are truncated to the nearest boundary.

In order to change the X and Y coordinates of the locator, the graphics input device must have a programmable locator position, (e.g. graphics input is from the keyboard and other relative locators).

SET PEN

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement defines the color for one or more entries in the color map.



Item	Description	Range
pen selector	numeric expression, rounded to an integer	0 thru 32 767
hue	numeric expression	0 thru 1
saturation	numeric expression	0 thru 1
luminosity	numeric expression	0 thru 1
HSL array name	name of a two-dimensional, three-column REAL array	any valid name
red	numeric expression	0 thru 1
green	numeric expression	0 thru 1
blue	numeric expression	0 thru 1
RGB array name	name of a two-dimensional, three-column REAL array	any valid name

Example Statements

```
SET PEN 3 COLOR Hue,Saturation,Luminosity
SET PEN Pen_number INTENSITY Color_map_array(*)
SET PEN 0 INTENSITY 4/15,4/15,4/15
```

Semantics

This statement defines the color for one or more entries in the color map. Either the HSL (hue/saturation/luminosity) color model or the RGB (red/green/blue) color model may be used. This statement is ignored for non-color mapped devices and color mapped devices in non-color map mode.

For both SET PEN COLOR and SET PEN INTENSITY, the pen selector specifies the first color map entry to be defined. If individual RGB or HSL values are given, that entry in the color map is the only one defined. If an array is specified, the color map is redefined, starting at the specified pen, and continuing until either the highest-numbered entry in the color map is redefined or the source array is exhausted.

Specifying color with the SET PEN and AREA PEN statements (resulting in non-dithered color) results in a much more accurate representation of the desired color than specifying the color with an AREA statement. Compare the five color plates shown in this entry with the corresponding plates in the AREA statement.

Note

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

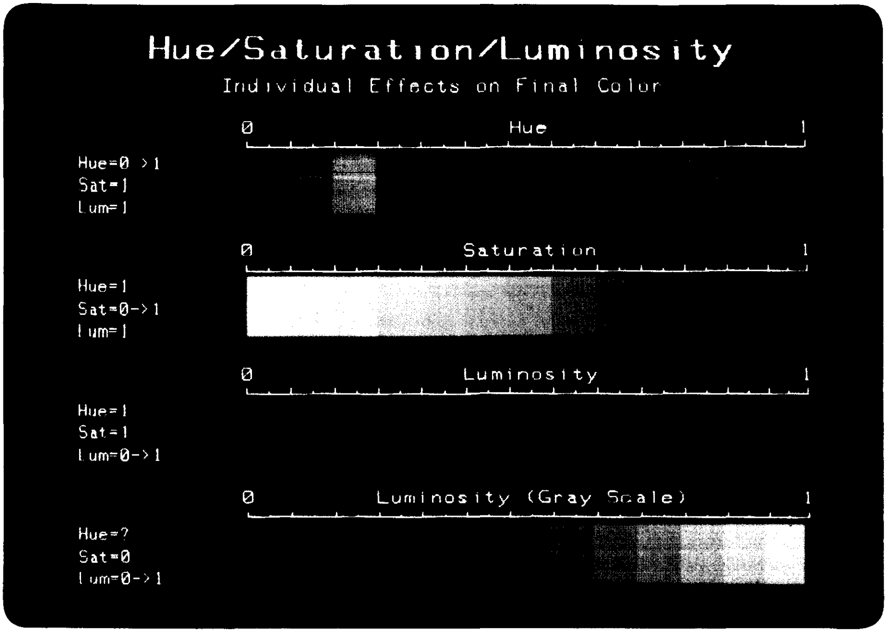
The hue value specifies the color. The hue ranges from zero to one, in a circular manner, with a value of zero resulting in the same hue as a value of one. The hue, as it goes from zero to one, proceeds through red, orange, yellow, green, cyan, blue, magenta, and back to red.

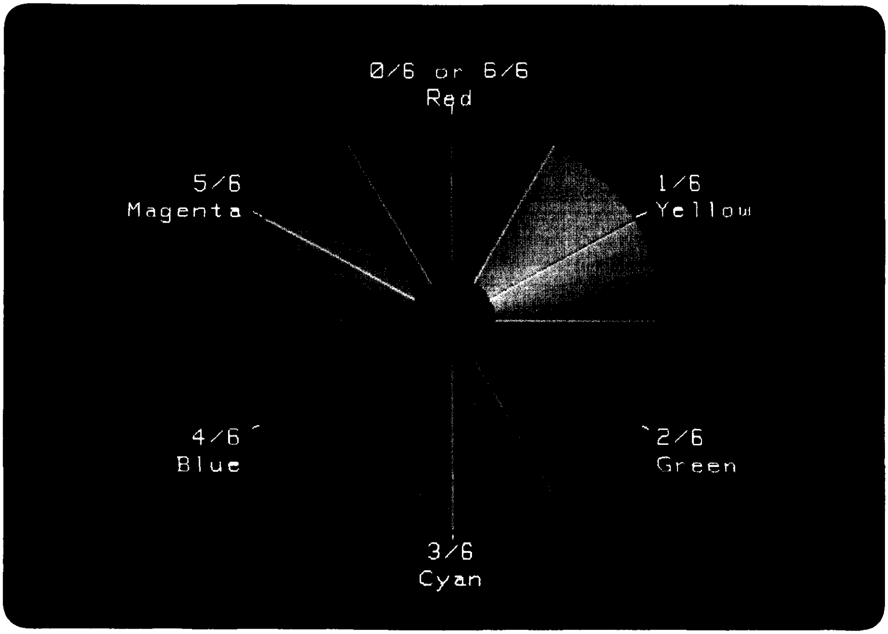
The saturation value, classically defined, is the inverse of the amount of white added to a hue. What this means is that saturation specifies the amount of hue to be mixed with white. As saturation goes from zero to one, there is 0% to 100% of pure hue added to white. Thus, a saturation of zero results in a gray, dependent only upon the luminosity; hue makes no difference.

The luminosity value specifies the brightness per unit area of the color. A luminosity of zero results in black, regardless of hue or saturation; if there is no color, it makes no difference which color it is that is not there.

The first color plate on the following page shows the changes brought about by varying one HSL parameter 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.

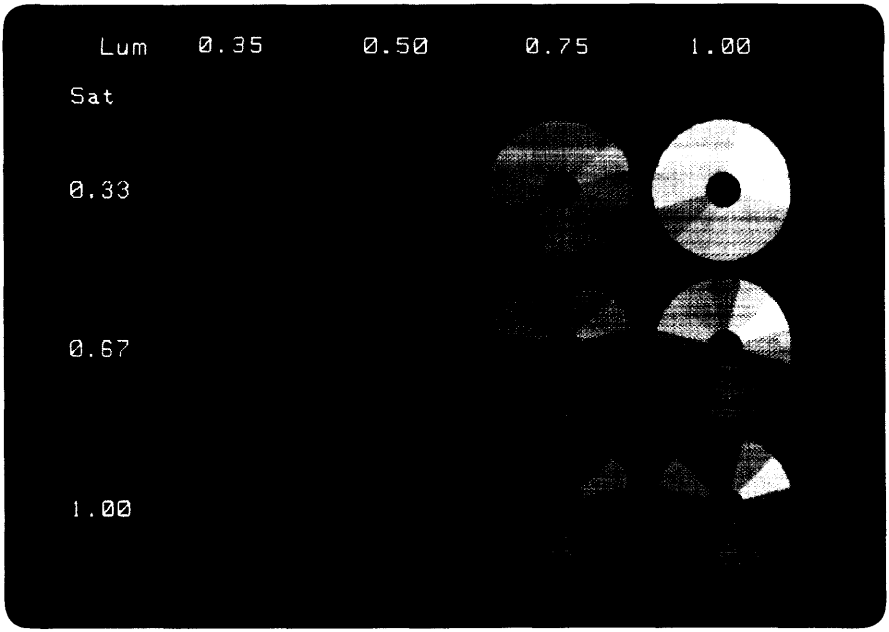
The second color wheel on the following page represents the fully saturated, fully luminous colors selected as the hue value goes from 0 through 1. Any value between zero and one, inclusive, can be chosen to select color, but the resolution (the amount the value can change before the color on the screen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





HSL Color Wheel

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



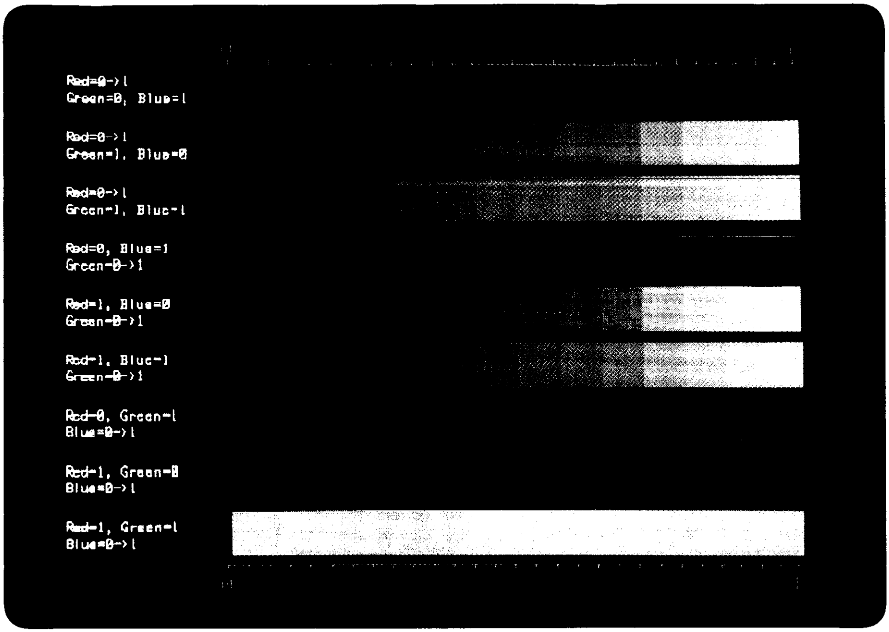
Effects of Saturation and Luminosity on Color

SET PEN INTENSITY

The red, green, and blue values specify the intensities of the red, green, and blue colors displayed on the screen.

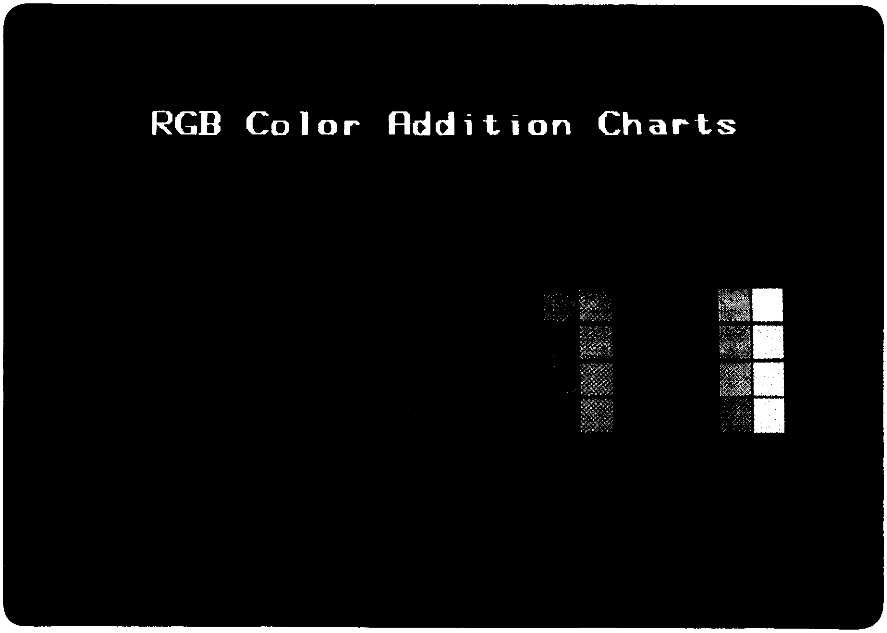
The first color plate demonstrates the effect of varying the intensity of one color component while the other two remain constant.

The second plate on the following page shows combinations of red, green and blue. The values are represented in fifteenths: 0 fifteenths, 5 fifteenths, 10 fifteenths, and 15 fifteenths—every fifth value. Fifteenths are the units. Thus, zero fifteenths through fifteen fifteenths made a total of sixteen levels. the values for each color component are represented in that color.



```
Red=0->1
Green=0, Blue=1

Red=0->1
Green=1, Blue=0

Red=0->1
Green=1, Blue=1

Red=0, Blue=1
Green=0->1

Red=1, Blue=0
Green=0->1

Red=1, Blue=1
Green=0->1

Red=0, Green=1
Blue=0->1

Red=1, Green=0
Blue=0->1

Red=1, Green=1
Blue=0->1
```

RGB Addition: One Color at a Time



RGB Color Addition Charts

BASIC/UX Specifics

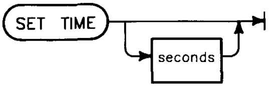
Dithering on the HP 2397 terminal assumes that the hardware color map contains power-on color assignments. However, these do not correspond to the standard BASIC color map. To make dithering results accurate on the HP 2397, the color map must be set to the following with SET PEN:

Pen	R	G	B
0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	1.0	0.0
3	1.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	1.0
6	0.0	1.0	1.0
7	1.0	1.0	1.0

SET TIME

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement resets the time-of-day given by the real-time clock.



Item	Description	Range
seconds	numeric expression, rounded to the nearest hundredth	0 thru 86 399.99

Example Statements

```
SET TIME 0
SET TIME Hours*3600+Minutes*60
SET TIME TIME("8:37:30")
SET TIME                (BASIC/UX only)
```

Semantics

SET TIME changes only the time within the current day, not the date. The new clock setting is equivalent to (TIMEDATE DIV 86 400)×86 400 plus the specified setting.

BASIC/UX Specifics

This statement does **not** reset the HP-UX clock, even if the user is super-user. Instead it resets the clock which BASIC/UX keeps for itself.

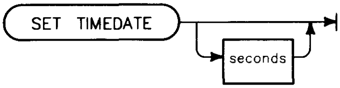
SET TIME without a parameter resynchronizes the time with the HP-UX clock. This does not affect the date nor the timezone. If the timezone is subsequently resynchronized with HP-UX (via TIMEZONE IS), then the time will change accordingly. The proper way to resynchronize both the time and timezone is to do the timezone first as in:

```
TIMEZONE IS
SET TIME
```

SET TIMEDATE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement resets the absolute seconds (time and day) given by the real-time clock.



Item	Description	Range
seconds	numeric expression, rounded to the nearest hundredth	2.086 629 12 E+11 thru 2.143 252 223 999 9 E+11

Example Statements

```
SET TIMEDATE TIMEDATE+3600
SET TIMEDATE Strange_number
SET TIMEDATE DATE("1 Jan 1989") + TIME("13:57:20")
SET TIMEDATE          (BASIC/UX only)
```

Semantics

The volatile clock is set to 2.086 629 12 E+11 (midnight March 1, 1900) at power-on (BASIC Workstation semantics). If there is a battery-backed (non-volatile) clock, then the volatile clock is synchronized with it at power-up. If the computer is on an SRM system (and has no battery-backed clock), then the volatile clock is synchronized with the SRM clock when the SRM and DCOMM binaries are loaded. The clock values represent Julian time, expressed in seconds.

BASIC/UX Specifics

The volatile clock is set to the current HP-UX time at power-on. The clock values represent Julian time, expressed in seconds.

Note that this statement does NOT reset the HP-UX clock, even if the user is super-user. Instead it resets the clock which BASIC keeps for itself.

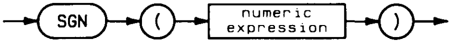
SET TIMEDATE without a parameter resynchronizes the time and date with the HP-UX clock. This does not affect the timezone. If the timezone is subsequently resynchronized with HP-UX (via TIMEZONE IS), then the time and date will change accordingly. The proper way to resynchronize the time, date, and timezone is to do the timezone first as in:

```
TIMEZONE IS
SET TIMEDATE
```

SGN

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns 1 if the argument is positive, 0 if it equals zero, and −1 if it is negative.



Example Statements

```
Root=SGN(X)*SQR(ABS(X))
Z=2*PI*SGN(Y)
```

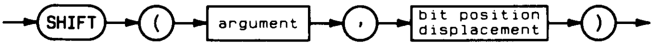
Semantics

COMPLEX arguments are not allowed with this function.

SHIFT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument by the number of bit positions specified, without wrap-around.



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

Example Statements

```
New_word=SHIFT(Old_word,-2)
Mask=SHIFT(1,Position)
```

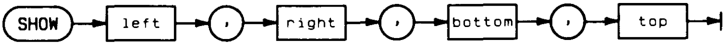
Semantics

If the bit position displacement is positive, the shift is towards the least-significant bit. If the bit position displacement is negative, the shift is towards the most-significant bit. Bits shifted out are lost. Bits shifted in are zeros. The SHIFT operation is performed without changing the value of any variable in the argument.

SHOW

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement is used to define an isotropic current unit-of-measure for graphics operations.



Item	Description	Range
left	numeric expression	—
right	numeric expression	≠ left
bottom	numeric expression	—
top	numeric expression	≠ bottom

Example Statements

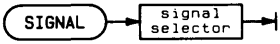
```
SHOW -5,5,0,100
SHOW Left,Right,Bottom,Top
```

Semantics

SHOW defines the values which must be displayed within the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. SHOW creates isotropic units (units the same in X and Y). The direction of an axis may be reversed by specifying the left greater than the right or the bottom greater than the top. (Also see WINDOW.)

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement generates a software interrupt.



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

Example Statements

```
SIGNAL 3
SIGNAL Bailout
```

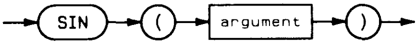
Semantics

If an ON SIGNAL statement for the specified signal selector exists, and all the other conditions for an event-initiated branch are fulfilled, the branch defined in the ON SIGNAL statement is taken. If no ON SIGNAL exists for the specified signal selector, the SIGNAL statement causes no action.

SIN

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the sine of the angle represented by the argument.



Item	Description	Range Restrictions
argument	numeric expression in current units of angle when arguments are INTEGER or REAL numeric expression in radians when argument is COMPLEX	absolute values less than: 1.708 312 781 2 E+10 deg. or 2.981 568 26 E+8 rad.; see "Range Restriction Specifics" for COMPLEX arguments

Examples Statements

```
Sine=SIN(Angle)
PRINT "Sine of ";Theta;"=";SIN(Theta)
```

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 SIN of a COMPLEX value, the COMPLEX binary must be loaded.

Range Restriction Specifics

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

```
CMPLX(SIN(Real_part)*COSH(Imag_part),COS(Real_part)*SINH(Imag_part))
```

where `Real_part` is the real part 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,

```
SIN(CMPLX(0,MAXREAL))
```

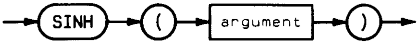
will cause error 22 due to the `COSH(Imag_part)` calculation.

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`).

SINH

Supported On	WS,UX
Option Required	COMPLEX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the hyperbolic sine 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=SINH(-8.2475)
PRINT "Hyperbolic Sine = ";SINH(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 SINH is as follows:

```
(EXP(Argument) - EXP(-Argument))/2
```

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

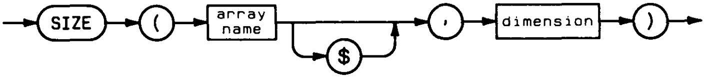
```
SINH(MAXREAL)
```

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

SIZE

Supported On	WS,UX
Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the size (number of elements) of a dimension of an array. This INTEGER value represents the difference between the upper bound and the lower bound, plus 1.



Item	Description	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

```
Upperbound(2)=BASE(A,2)+SIZE(A,2)-1
Number_words=SIZE(Words$,1)
```

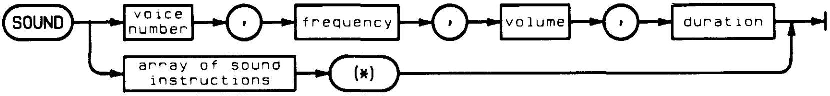
SORT

See the MAT SORT statement.

SOUND

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement produces a single tone or multiple tones on the sound generator of an HP-HIL interface.



Item	Description	Range
voice number	numeric expression, rounded to an integer	1 thru 3
frequency	numeric expression, rounded to an integer	83 thru 83 333 Hz (see following table)
volume	numeric expression, rounded to an integer	0 thru 15
duration	numeric expression, rounded to an integer	0, 0.01 thru 2.55
array of sound instructions	INTEGER array	must contain the proper number of non-zero values (see Semantics)

Example Statements

```
SOUND Voice_num,Freq,Volume,Duration
SOUND 1,440,12,0.50
SOUND Instructions(*)
```

Semantics

If the **multiple-parameter syntax** is used, then the SOUND statement generates one tone on the specified voice number; the frequency, volume, and duration of the tone are as specified by the last three parameters of the statement. Note that the BASIC system does not wait for the tone to finish before executing the following program line or statement (if any). If you want to generate a sequence of tones, you must either generate a delay between SOUND statements (such as with WAIT), or use the SOUND syntax described below.

If the **single-parameter syntax** is used (that is, a numeric array is specified), then the elements of the array are read sequentially and interpreted according to the following rules:

Instruction	Sound Chip Effect Produced
0	Exit the SOUND statement (and stop reading array elements)
1 to 3	The specified voice is to be used; also says to read the *next three* array elements, and interpret them as follows, respectively: • tone number—used to set the frequency (frequency = 83 333 / tone number). • volume—0 = off; 1 thru 15 are lowest to highest volume. • duration—values 0 thru 255 are interpreted as follows: 0 is interpreted as "sound indefinitely". 1 thru 255 are interpreted as 10's of milliseconds (i.e., 1/100 second);

4	Specifies that the **noise voice** is to be used; also says to read the next *three* array elements and interpret them as above (the same as with voice numbers 1 to 3), *except* that the *tone number* parameter is interpreted as follows:
	0 => *periodic* noise; *fast* shift register clock;
	1 => *periodic* noise; *medium* shift register clock;
	2 => *periodic* noise; *slow* shift register clock;
	3 => *periodic* noise; clock shift register *with voice 3*;
	4 => *white* noise; *fast* shift register clock;
	5 => *white* noise; *medium* shift register clock;
	6 => *white* noise; *slow* shift register clock;
	7 => *white* noise; clock shift register *with voice 3.*

5 to 8	Wait for voice 1 to 4, respectively, to finish sounding before executing the next sound instruction (if any).
9	Read the following array element, and wait the specified interval (100 microseconds × that element's value) before executing the next instruction (if any).

If the end of the array is reached on one of these boundaries, then the SOUND statement terminates normally; however, if the last element of the array has been reached and the BASIC system expects to read more values, then error 17 will be reported (subscript out of range).

Producing Notes in the Equal-Tempered Scale

Here is a list of the notes in the equal-tempered[1] musical scale. The table shows that the frequencies available with the SOUND statement are *close* to the even-tempered notes, but are *not exact.*

Note	Frequency	Tone[2] Number	Closest Frequency	Error (Hz)	Error[3] (%)
E	82.41	1011	82.43	+0.02	+0.02
F	87.31	954	87.35	+0.04	+0.05
F#	92.50	901	92.49	-0.01	-0.01
G	98.00	850	98.04	+0.04	+0.04
G#	103.83	803	103.78	-0.05	-0.05
A	110.00	758	109.94	-0.06	-0.06
A#	116.54	715	116.55	+0.01	+0.01
B	123.47	675	123.46	-0.01	-0.01
C	130.81	637	130.82	+0.01	+0.01
C#	138.59	601	138.66	+0.07	+0.05
D	146.83	568	146.71	-0.12	-0.08
D#	155.56	536	155.47	-0.09	-0.06
E	164.81	506	164.69	-0.12	-0.08
F	174.61	477	174.70	+0.09	+0.05
F#	185.00	450	185.18	+0.19	+0.10
G	196.00	425	196.08	+0.08	+0.04
G#	207.65	401	207.81	+0.16	+0.08

[1] The *equal-tempered* scale is derived from the following relationship:

frequency of note = 2^(1/12) × frequency of preceding note

[2] The *tone number* is the value of the parameter required when using the SOUND statement with an array of instructions.

[3] This error does not account for errors in the clock used to generate these frequencies.

Note	Frequency	Tone[1] Number	Closest Frequency	Error (Hz)	Error[2] (%)
A	220.00	379	219.88	-0.12	-0.06
A#	233.08	358	232.77	-0.31	-0.13
B	246.94	337	247.28	+0.34	+0.14
C	261.63	319	261.23	-0.39	-0.15
C#	277.18	301	276.85	-0.33	-0.12
D	293.66	284	293.43	-0.24	-0.08
D#	311.13	268	310.94	-0.18	-0.06
E	329.63	253	329.38	-0.25	-0.08
F	349.23	239	348.67	-0.55	-0.16
F#	369.99	225	370.37	+0.37	+0.10
G	392.00	213	391.23	-0.76	-0.19
G#	415.30	201	414.59	-0.71	-0.17
A	440.00	189	440.92	+0.92	+0.21
A#	466.16	179	465.55	-0.62	-0.13
B	493.88	169	493.09	-0.79	-0.16
C	523.25	159	524.11	+0.86	+0.16
C#	554.37	150	555.55	+1.19	+0.21
D	587.33	142	586.85	-0.48	-0.08
D#	622.25	134	621.89	-0.37	-0.06
E	659.26	126	661.37	+2.12	+0.32
F	698.46	119	700.28	+1.82	+0.26
F#	739.99	113	737.46	-2.53	-0.34
G	783.99	106	786.16	+2.17	+0.28
G#	830.61	100	833.33	+2.72	+0.33

[1] The *tone number* is the value of the parameter required when using the SOUND statement with an array of instructions.

[2] This error does not account for errors in the clock used to generate these frequencies.

Note	Frequency	Tone[1] Number	Closest Frequency	Error (Hz)	Error[2] (%)
A	880.00	95	877.19	-2.81	-0.32
A#	932.33	89	936.33	+4.00	+0.43
B	987.77	84	992.06	+4.29	+0.43
C	1046.50	80	1041.66	-4.84	-0.46
C#	1108.73	75	1111.11	+2.38	+0.21
D	1174.66	71	1173.70	-0.95	-0.08
D#	1244.51	67	1243.78	-0.73	-0.06
E	1318.51	63	1322.75	+4.24	+0.32
F	1396.91	60	1388.88	-8.03	-0.57
F#	1479.98	56	1488.09	+8.11	+0.55
G	1567.98	53	1572.32	+4.34	+0.28
G#	1661.22	50	1666.66	+5.44	+0.33
A	1760.00	47	1773.04	+13.04	+0.74
A#	1864.66	45	1851.84	-12.81	-0.69
B	1975.53	42	1984.12	+8.59	+0.43
C	2093.00	40	2083.33	-9.68	-0.46
C#	2217.46	38	2192.97	-24.49	-1.10
D	2349.32	35	2380.94	+31.62	+1.35
D#	2489.02	33	2525.24	+36.23	+1.46
E	2637.02	32	2604.16	-32.86	-1.25
F	2793.83	30	2777.77	-16.06	-0.57
F#	2959.96	28	2976.18	+16.22	+0.55
G	3135.96	27	3086.41	-49.56	-1.58
G#	3322.44	25	3333.32	+10.88	+0.33

[1] The *tone number* is the value of the parameter required when using the SOUND statement with an array of instructions.

[2] This error does not account for errors in the clock used to generate these frequencies.

Note	Frequency	Tone[1] Number	Closest Frequency	Error (Hz)	Error[2] (%)
A	3520.00	24	3472.21	-47.79	-1.36
A#	3729.31	22	3787.86	+58.55	+1.57
B	3951.07	21	3968.24	+17.17	+0.43
C	4186.01	20	4166.65	-19.36	-0.46
C#	4434.92	19	4385.95	-48.97	-1.10
D	4698.64	18	4629.61	-69.03	-1.47
D#	4978.03	17	4901.94	-76.09	-1.53
E	5274.04	16	5208.31	-65.73	-1.25
F	5587.65	15	5555.53	-32.12	-0.57
F#	5919.91	14	5952.36	+32.45	+0.55
G	6271.93	13	6410.23	+138.30	+2.21
G#	6644.88	13	6410.23	-0234.64	-3.53
A	7040.00	12	6944.42	-95.58	-1.36
A#	7458.62	11	7575.73	+117.11	+1.57
B	7902.13	11	7575.73	-326.41	-4.13
C	8372.02	10	8333.30	-38.72	-0.46
C#	8869.84	9	9259.22	+389.38	+4.39
D	9397.27	9	9259.22	-138.05	-1.47
D#	9956.06	8	10416.63	+460.56	+4.63
E	10548.08	8	10416.63	-131.46	-1.25
F	11175.30	7	11904.71	+729.41	+6.53
F#	11839.82	7	11904.71	+64.89	+0.55
G	12543.85	7	11904.71	-639.14	-5.10
G#	13289.75	6	13888.83[3]	+599.08	+4.51

[1] The *tone number* is the value of the parameter required when using the SOUND statement with an array of instructions.

[2] This error does not account for errors in the clock used to generate these frequencies.

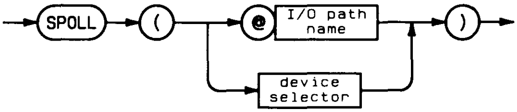
[3] While it is possible to specify frequencies above 13888 Hz, they may be inaudible (due to the high-frequency roll-off of the speaker amplifier circuit.

SPANISH

See the LEXICAL ORDER IS statement.

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns an integer containing the serial poll response from the addressed device.



Item	Description	Range
I/O path	name name assigned to a device	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	must include a primary address (see Glossary)

Example Statements

```
Stat=SPOLL(707)
IF SPOLL(@Device) THEN Respond
```

Semantics

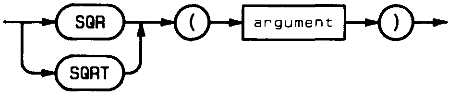
The computer must be the active controller to execute this function. Multiple listeners are not allowed. One secondary address may be specified to get status from an extended talker. Refer to the documentation provided with the device being polled for information concerning the device's status byte.

Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT
Not Active Controller	Error			

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the square root of the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	any valid INTEGER or REAL value for INTEGER and REAL expressions; for COMPLEX arguments, the range restriction for ABS applies here.

Examples Statements

```
Amps=SQRT(Watts/Ohms)
PRINT "Square root of ";X;"=";SQR(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 SQR or SQRT of a COMPLEX value, the COMPLEX binary must be loaded.

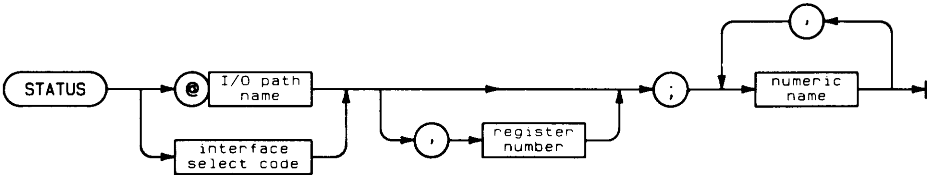
STANDARD

See the LEXICAL ORDER IS statement.

STATUS

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement returns the contents of interface or I/O path name status registers.



Item	Description	Range
I/O path name	name assigned to a device, devices, mass storage file, buffer, or pipe	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	1 thru 40
register number	numeric expression, rounded to an integer; Default = 0	interface dependent
numeric name	name of a numeric variable	any valid name

Example Statements

```
STATUS 1;Xpos,Ypos
STATUS @File,5;Record
```

Semantics

The value of the beginning register number is copied into the first variable, the next register value into the second variable, and so on. The information is read until the variables in the list are exhausted; there is no wrap-around to the first register. An attempt to read a nonexistent register generates an error.

The register meanings depend on the specified interface or on the resource to which the I/O path name is currently assigned. Register 0 of I/O path names can be interrogated with STATUS even if the I/O path name is currently invalid (i.e., unassigned to a resource). Note that the Status registers of an I/O path are different from the Status registers of an interface. All Status and Control registers are summarized in the "Interface Registers" section at the back of the book.

STEP

See the FOR...NEXT construct.

STOP

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement terminates execution of the program.

Semantics

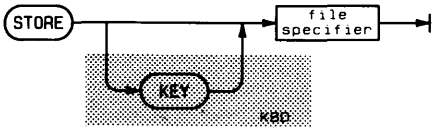
Once a program is stopped, it cannot be resumed by CONTINUE. RUN must be executed to restart the program. PAUSE should be used if you intend to continue execution of the program.

A program can have multiple STOP statements. Encountering an END statement or pressing the STOP (Shift-Stop on the ITF keyboards) key has the same effect as executing STOP. After a STOP, variables that existed in the main context are available from the keyboard.

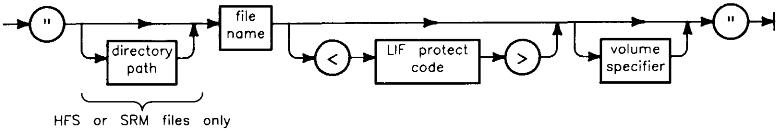
STORE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement creates a file and stores the program or typing-aid key definitions into it.



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)

Example Statements

```
STORE Filename$&Vol$
STORE "Dir<SRM_RW_pass>/Program"

STORE KEY "Typing_aids"
STORE KEY "KEYS:REMOTE"
STORE KEY "/USERS/MARK/TYPING"
```

Semantics

In all STORE statements, an error will occur if the storage media cannot be found, the media or directory is full, or the file specified already exists. Also, if a LIF protect code is specified, it will be applied to the new LIF file. To update a file which already exists, see RE-STORE.

STORE

The STORE statement creates a PROG file and stores an internal form of the program into that file.

STORE KEY

STORE KEY creates a file of type BDAT, and stores the current *typing-aid* softkey definitions (**not** ON KEY softkey definitions) into it. These definitions may subsequently be reloaded with the LOAD KEY statement.

For each defined typing-aid softkey, an integer and a string are sent to the file. The integer is the key number, and the string is the key definition. The data is written with FORMAT OFF (see the OUTPUT statement). Keys with no definition are not written to the file.

HFS Permissions and File Headers

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

When a file is stored on an HFS volume, access permission bits are set to RW-RW-RW-. You can modify the access permission bits with PERMIT, if desired.

On HFS volumes, STORE creates a PROG file that contains a 512-byte header (at the beginning of the file's contents). This header allows the BASIC system to recognize the file as being a PROG file. (The header is handled automatically by the BASIC system, so you do not have to take any special actions.)

SRM Passwords and Exclusive Mode

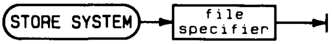
In order to STORE an SRM file, you need to have R (read) and W (write) capabilities on the immediately superior directory, and R capabilities on all other superior directories.

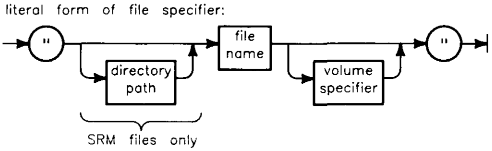
Including an SRM password with the file name does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but a password in the file name portion of the STORE statement will be ignored.

STORE opens the remote file in exclusive mode (denoted as `LOCK` in a CAT listing) and enforces that status on the file until the STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the STORE.

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

The command stores the entire BASIC operating system currently in memory including any BINs that are loaded (only use on BASIC Workstation).





Item	Description	Range
file specifier	string expression	(see drawing)
directory path	literal	(see MASS STORAGE IS)
file name	literal	(see Semantics)
volume specifier	literal	(see MASS STORAGE IS)

Example Statements

```
STORE SYSTEM "SYSTEM_BA5:,700"
STORE SYSTEM "BACKUP1"
STORE SYSTEM "SYSTEM_B1:REMOTE"
STORE SYSTEM "/SYSTEMS/SYSTEM_NEW"
STORE SYSTEM "/SYS_HFS"
```

Semantics

If the file name already exists, an error is reported.

On LIF volumes, SYSTM file names can be up to 10 characters long; on SRM volumes, they may be up to 16 characters long.

On HFS volumes, SYSTM file names may only be up to 9 characters long, since the HFS loader assumes that strings will be terminated with the null character, CHR$(0). In addition, SYSTM file names on HFS volumes less than 9 characters long will be padded with null characters to a length of 10 characters. This may cause unexpected results, since null characters act as "wild cards" on HFS volumes. For instance, suppose that there are two SYSTM files on the same HFS volume named SYSTEM_BA and SYSTEM_B, and that they are listed as 1B and 2B, respectively, by the Boot ROM. Typing 1B will boot SYSTEM_BA, as expected. However, typing 2B will also boot SYSTEM_BA because of the null (wild card) character in the 9th position in the SYSTM file named SYSTEM_B.

The BASIC system and any BINs in memory are stored in the SYSTM file. If the file name begins with SYSTEM_, the Boot ROM can find it and load it at power up or SYSBOOT. (Note that Boot ROM 3.0 and A, and later versions, can find and load files beginning with SYS.) On SRM, the system must be located in /SYSTEMS for the Boot ROM to find it. On HFS, the system must be stored in the root ("/") for the Boot ROM to find it.

Note that if you did a SCRATCH BIN to remove the CRT driver you did not need, and then stored the system, when you reboot, the CRT driver for the other display is not available. If the CRT needs the other driver, you cannot use the display. Execute a LOAD BIN command to load the needed driver.

STORE SYSTEM cannot be used with ROM BASIC systems.

HFS Permissions and File Headers

In order to use STORE SYSTEM on an HFS volume, you need to have W (write) and X (search) permission on the root directory. ON HFS, you can STORE SYSTEM only to the root directory.

Do not RENAME a file stored into the root directory of an HFS volume by STORE SYSTEM.

A SYSTM file (or an HP-UX file stored by STORE SYSTEM) which is placed in the root directory of an HFS volume by COPY or LINK will not be found by the Boot ROM.

The R (read) access capability on the system file created with STORE SYSTEM must be public to allow use of the file for booting.

On HFS volumes, STORE SYSTEM creates an HP-UX file that contains a special header (at the beginning of the file's contents) to make the file conform to the HP-UX "a.out" file format. (The header is handled automatically by the BASIC system, so you do not have to take any special actions.)

SRM Access Capabilities

In order to use STORE SYSTEM on an SRM volume, you need to have R (read) and W (write) capabilities on the immediately superior directory, and R capabilities on all other superior directories.

The R (read) access capability on the system file created with STORE SYSTEM must be public to allow use of the file for booting.

Including an SRM password with the file name does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but a password in the file name will be ignored.

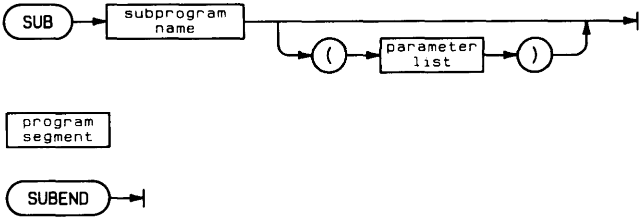
BASIC/UX Specifics

STORE SYSTEM is not necessary nor supported on BASIC/UX as BASIC/UX is a unified system.

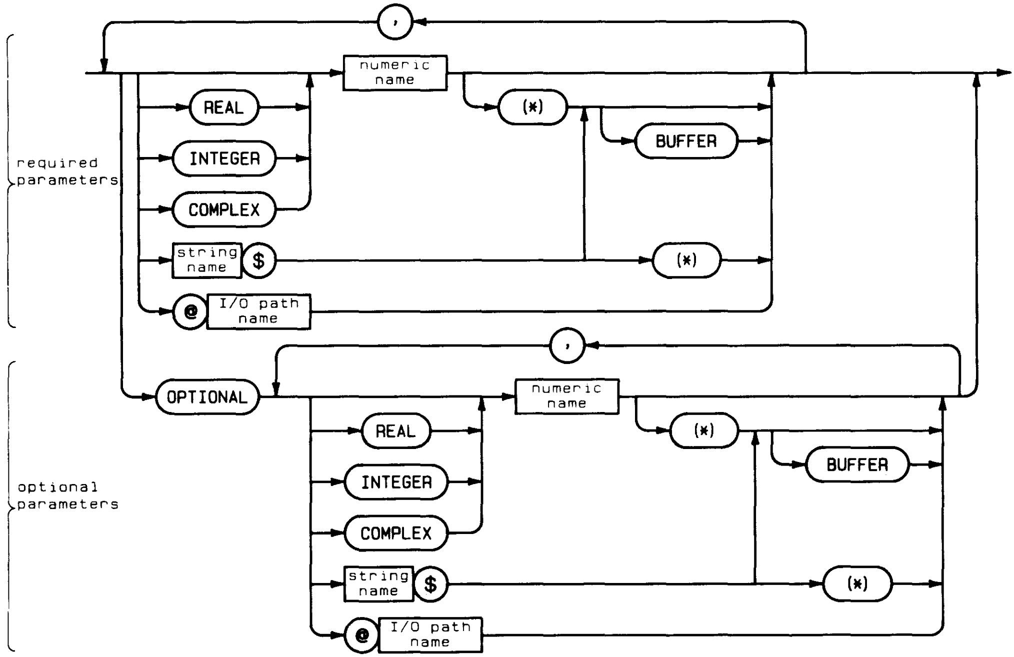
SUB

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	No

This is the first statement in a SUB subprogram and can specify the subprogram's formal parameters.



parameter list:



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

Example Statements

```
SUB Parse(String$)
SUB Transform(@Printer,INTEGER Array(*),OPTIONAL Text$)
SUB Complex_sub(COMPLEX Real_imag)
```

Semantics

SUB subprograms must appear after the main program. The first line of the subprogram must be a SUB statement. The last line must be a SUBEND statement. Comments after the SUBEND are considered to be part of the subprogram.

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

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

SUBEXIT may be used to leave the subprogram at some point other than the SUBEND. Multiple SUBEXITs are allowed, and SUBEXIT may appear in an IF...THEN statement. SUBEND is prohibited in IF...THEN statements, and may only occur once in a subprogram. ERROR SUBEXIT may be used in place of SUBEXIT.

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

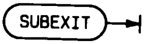
SUBEND

See the SUB statement.

SUBEXIT

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an `IF...THEN`	Yes

This statement may be used to return from a SUB subprogram at some point other than the SUBEND statement. It allows multiple exits from a subprogram.

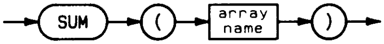


See also ERROR SUBEXIT.

SUM

Supported On	WS,UX
Option Required	MAT
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the sum of all elements of a numeric array. The value returned is of the same type as the array.



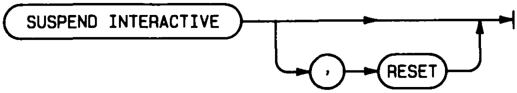
Item	Description	Range
array name	name of a numeric array	any valid name

Example Statements

```
Array_sum=SUM(A)
Sum_squares=SUM(Squares)
```

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement disables the [EXECUTE], [ENTER], [Return], [PAUSE], [STOP], [CLR I/O], [Break], and (optionally) [RESET] key functions during a running program.



Example Statements

```
SUSPEND INTERACTIVE,RESET
IF NOT Kbd_flag THEN SUSPEND INTERACTIVE
```

Semantics

Execution of a PAUSE statement, a TRACE PAUSE statement, or a fatal execution error temporarily restores the suspended key functions. CONTINUE after a PAUSE will again disable the keys.

SUSPEND INTERACTIVE is cancelled by RESUME INTERACTIVE, STOP, END, RUN, SCRATCH, GET, LOAD, or [RESET]. Although LOAD cancels SUSPEND INTERACTIVE, LOADSUB does not. SUSPEND INTERACTIVE has no effect unless a program is running.

Note

Suspending the [RESET] key will prevent you from stopping a program before it ends.

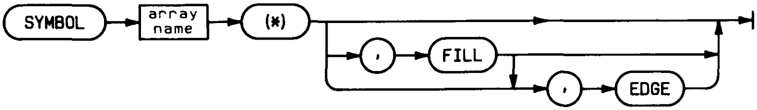
[EXECUTE], [ENTER], and [Return] can still be used to respond to an ENTER or INPUT statement, but cannot be used for live keyboard execution.

SWEDISH

See the LEXICAL ORDER IS statement.

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement allows labelling with user-defined symbols.



Item	Description	Range
array name	name of a two-dimensional, two-column or three-column REAL array	any valid name

Example Statements

```
SYMBOL My_char(*)
SYMBOL Logo(*),FILL,EDGE
```

Semantics

The user-defined symbol is created with moves and draws defined in a *symbol coordinate system.* The symbol coordinate system is a rectangular area nine units wide and fifteen units high, that is, a character cell. A symbol can extend outside the limits of the 9×15 symbol coordinate system rectangle. A symbol defined in the symbol coordinate system is affected by the label transformations CSIZE, LDIR, and LORG. The symbol is drawn using the current pen and line type, and it will be clipped at the current clip boundary.

When defining a symbol in the symbol coordinate system, coordinates may be ouside the 9×15 character cell; thus, characters can be made which are several character cells wide and several character cells high. For this reason, the current pen position is not updated to the next character's reference point, but it remains at the last X,Y coordinate specified in the array. A move is made to the first point regardless of the value in the third column of that row in the array.

The symbol may have polygons defined in its data, and the polygons may be filled and/or edged. The fill color and pen number/line type used are those defined at the time the polygon is closed.

FILL and EDGE

When FILL or EDGE is specified, each sequence of two or more lines forms a polygon. The polygon begins at the first point on the sequence, includes each successive point, and the final point is connected or closed back to the first point. A polygon is closed when the end of the array is reached, or when the value in the third column is an even number less than three, or in the range 5 to 8 or 10 to 15.

If FILL and/or EDGE are specified on the SYMBOL statement itself, it causes the polygons defined within it to be filled with the current fill color and/or edged with the current pen color. If polygon mode is entered from within the array, and the FILL/EDGE directive for that series of polygons differs from the FILL/EDGE directive on the SYMBOL statement itself, the directive in the array replaces the directive on the statement. In other words, if a "start polygon mode" operation selector (a 6, 10, or 11) is encountered, any current FILL/EDGE directive (whether specified by a keyword or an operation selector) is replaced by the new FILL/EDGE directive.

If FILL and EDGE are both declared on the SYMBOL statement, FILL occurs first. If neither one is specified, simple line drawing mode is assumed; that is, polygon closure does not take place.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

Applicable Graphics Transformations

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

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

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

Note 4: RPLOT and IPLOT are affected by PDIR.

When using a SYMBOL statement, the following table of *operation selectors* applies. An operation selector is the value in the third column of a row of the array to be plotted. The array must be a two-dimensional, two-column or three-column array. If the third column exists, it will contain operation selectors which instruct the computer to carry out certain operations. Polygons may be defined, edged (using the current pen), filled (using the current fill color), pen and line type may be selected, and so forth. See the list below.

Column 1	Column 2	Operation Selector	Meaning
X	Y	−2	Pen up before moving
X	Y	−1	Pen down before moving
X	Y	0	Pen up after moving (Same as +2)
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	ignored	3	Select pen
line type	repeat value	4	Select line type
color	ignored	5	Color value
ignored	ignored	6	Start polygon mode with FILL
ignored	ignored	7	End polygon mode
ignored	ignored	8	End of data for array
ignored	ignored	9	NOP (no operation)
ignored	ignored	10	Start polygon mode with EDGE
ignored	ignored	11	Start polygon mode with FILL and EDGE
ignored	ignored	12	Draw a FRAME
pen number	ignored	13	Area pen value
red value	green value	14	Color 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 SYMBOL statement. Even is up, odd is down, positive is after pen motion, negative is before pen motion. Zero is considered positive.

Selecting Pens

An operation selector of 3 selects a pen. The value in column one is the pen number desired. The value in column two is ignored.

Selecting Line Types

An operation selector of 4 selects a line type. The line type (column one) selects the pattern, and the repeat value (column two) is the length in GDUs that the line extends before a single occurrence of the pattern is finished and it starts over. On the CRT, the repeat value is evaluated and rounded *down* to the next multiple of 5, with 5 as the minimum.

Selecting a Fill Color

Operation selector 13 selects a pen from the color map with which to do area fills. This works identically to the AREA PEN statement. Column one contains the pen number.

Defining a Fill Color

Operation selector 14 is used in conjunction with operation selector 15. Red and green are specified in columns one and two, respectively, and column three has the value 14. Following this row in the array (not necessarily immediately), is a row whose operation selector in column three has the value of 15. The first column in that row contains the blue value. These numbers range from 0 to 32 767, where 0 is no color and 32 767 is full intensity. Operation selectors 14 and 15 together comprise the equivalent of an AREA INTENSITY statement, which means it can be used on both a monochromatic and a color CRT.

Operation selector 15 actually puts the area intensity into effect, but only if an operation selector 14 has already been received.

Operation selector 5 is another way to select a fill color. The color selection is through a Red-Green-Blue (RGB) color model. The first column is encoded in the following manner. There are three groups of five bits right-justified in the word; that is, the most significant bit in the word is ignored. Each group of five bits contains a number which determines the intensity of the corresponding color component, which ranges from zero to sixteen. The value in each field will be sixteen minus the intensity of the color component. For example, if the value in the first column of the array is zero, all three five-bit values would thus be zero. Sixteen minus zero in all three cases would turn on all three color components to full intensity, and the resultant color would be a bright white.

Assuming you have the desired intensities (which range from 0 thru 1) for red, green, and blue in the variables R, G, and B, respectively, the value for the first column in the array could be defined thus:

```
Array(Row,1)=SHIFT(16*(1-B),-10)+SHIFT(16*(1-G),-5)+16*(1-R)
```

If there is a pen color in the color map similar to that which you request here, that non-dithered color will be used. If there is not a similar color, you will get a dithered pattern.

Polygons

A six, ten, or eleven in the third column of the array begins a "polygon mode". If the operation selector is 6, the polygon will be filled with the current fill color. If the operation selector is 10, the polygon will be edged with the current pen number and line type. If the operation selector is 11, the polygon will be both filled and edged. Many individual polygons can be filled without terminating the mode with an operation selector 7. This can be done by specifying several series of draws separated by moves. The first and second columns are ignored and should not contain the X and Y values of the first point of a polygon.

Operation selector 7 in the third column of a plotted array terminates definition of a polygon to be edged and/or filled and also terminates the polygon mode (entered by operation selectors 6, 10, or 11). The values in the first and second columns are ignored, and the X and Y values of the last data point should not be in them. Edging and/or filling of the most recent polygon will begin immediately upon encountering this operation selector.

Doing a FRAME

Operation selector 12 does a FRAME around the current soft-clip limits. Soft clip limits cannot be changed from within the SYMBOL statement, so one probably would not have more than one operation selector 12 in an array to SYMBOL, since the last FRAME will overwrite all the previous ones.

Premature Termination

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

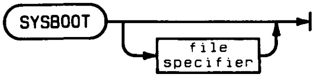
Ignoring Selected Rows in the Array

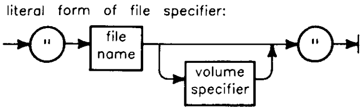
Operation selector 9 causes the row of the array it is in to be ignored. Any operation selector greater than 15 is also ignored, but operation selector 9 is retained for compatibility reasons. *Operation selectors less than −2 are not ignored.* If the value in the third column is less than zero, only evenness/oddness is considered.

If you attempt to fill a figure on an HPGL plotter, the figure will not be filled, but will be edged, regardless of the directives on the statement.

Supported On	WS
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This command returns control to the Boot ROM to restart the system configuration and selection process. Only use on BASIC Workstation.





Item	Description	Range
file specifier	string expression, specifying a SYSTM file to be booted	(see drawing)
file name	literal	(see Semantics)
volume specifier	literal	(see MASS STORAGE IS)

Example Statements

```
SYSBOOT
SYSBOOT Sys_file$&Volume$
SYSBOOT "SYSTEM_BA5:,700"
```

Semantics

If no file specifier is included, the normal Boot ROM power-up search sequence is initiated. (See "Loading BASIC" in the *Installing, Using, and Maintaining the BASIC System* manual for a sequence of mass storage devices searched.)

If a file specifier is included, it must a valid LIF file specifier (10 characters or less). The Boot ROM restricts the file name, if included, to 10 characters. System names on SRM can be up to 16 characters. To boot a system whose name is more than 10 characters, do not specify the file name and use the Boot ROM to select the correct file.

If no volume specifier is included in the file specifier, the current default volume is assumed.

To boot a system from the SRM, public read access is required and the system must be located in /SYSTEMS. The directory path, /SYSTEMS must be omitted from the file specifier. The Boot ROM looks for the file in /SYSTEMS.

To boot from HFS the system must be located in the root directory (/). System names on HFS must be 9 characters or less.

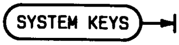
BASIC/UX Specifics

Not supported on BASIC/UX. It generates an error.

SYSTEM KEYS

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement changes the softkey definitions on an ITF keyboard to the System menu.



Example Statements

```
SYSTEM KEYS
IF Change_keys THEN SYSTEM KEYS
```

Semantics

This statement only affects the normal mode of the ITF Keyboard (i.e. it does nothing on an HP 98203A/B/C Keyboard and causes no visible change on an ITF Keyboard when the Keyboard Compatibility Mode, KBD CMODE, is on).

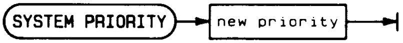
Note that the functionality of this statement can be achieved through KBD CONTROL register 2.

For information on the softkey definitions, read the *Installing, Using, and Maintaining the BASIC System* manual.

SYSTEM PRIORITY

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement sets system priority to a specified value.



Item	Description	Range
new priority	numeric expression, rounded to an integer	0 thru 15

Example Statements

```
SYSTEM PRIORITY Old
IF Critical_code THEN SYSTEM PRIORITY 15
```

Semantics

Zero is the lowest user-specifiable priority and 15 is the highest. The END, ERROR, and TIMEOUT events have an effective priority higher than the highest user-specifiable priority. If no SYSTEM PRIORITY has been executed, minimum system priority is 0.

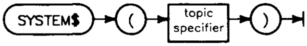
This statement establishes the minimum for system priority. Once the minimum system priority is raised with this statement, any events of equal or lower priority will be logged but not serviced. In order to allow service of lower-priority events, minimum system priority must be explicitly lowered.

If SYSTEM PRIORITY is used to change the minimum system priority in a subprogram context, the former value is restored when the context is exited.

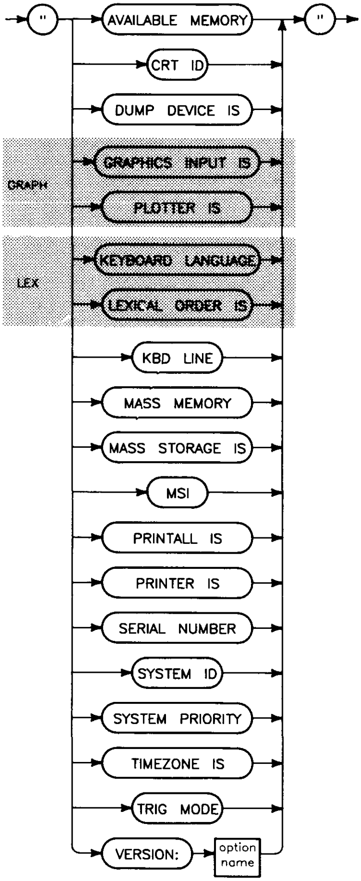
Error 427 results if SYSTEM PRIORITY is executed in a service routine for an ON ERROR GOSUB or ON ERROR CALL statement.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns a string containing system status and configuration information.



literal form of topic specifier:



Item	Description	Range
type of information	string expression	—
option name	literal specifying an option or BIN	BASIC, BCD, BUBBLE, CLOCK, COMPLEX, CRTA, CRTB, CRTX, CS80, DCOMM, DISC, EDIT, EPROM, ERR, FHPIB, GPIO, GRAPH, GRAPHX, HFS, HP9885, HPIB, IO, KBD, LEX, MAT, MS, PDEV, SERIAL, SRM, TRANS, XREF, etc.

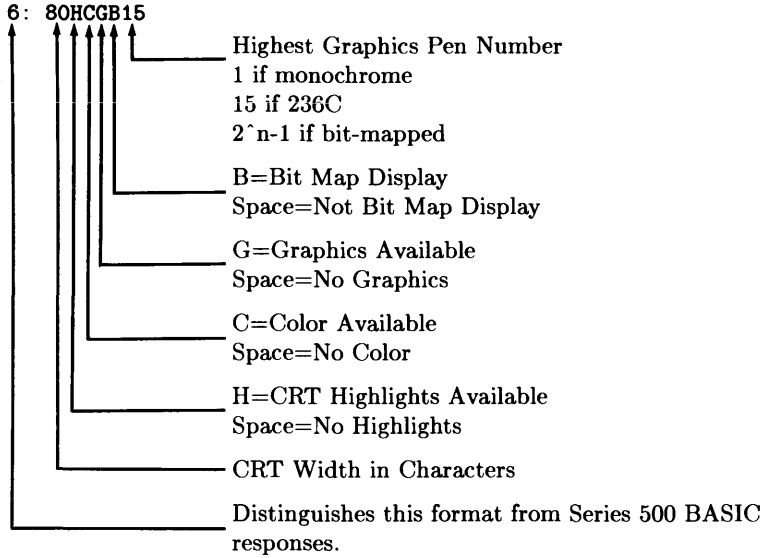
Example Statements

```
IF SYSTEM$("TRIG MODE")="RAD" THEN CALL Change_mode
System_prior=VAL(SYSTEM$("SYSTEM PRIORITY"))
SYSTEM$("VERSION:OS")        (BASIC/UX only)
```

Semantics

The topic specifier is used to specify what system configuration information SYSTEM$ will return. The following table lists the valid topic specifiers and the information returned for each one.

Topic Specifier	Information Returned
AVAILABLE MEMORY	Bytes of available memory

CRT ID	
	6: 80HCGB15
	Highest Graphics Pen Number
	1 if monochrome
	15 if 236C
	2^n-1 if bit-mapped
	B=Bit Map Display
	Space=Not Bit Map Display
	G=Graphics Available
	Space=No Graphics
	C=Color Available
	Space=No Color
	H=CRT Highlights Available
	Space=No Highlights
	CRT Width in Characters
	Distinguishes this format from Series 500 BASIC responses.

DUMP DEVICE IS	A string containing numerals which specify the device selector for the currently assigned DUMP DEVICE IS device.
GRAPHICS INPUT IS	A string containing numerals which specify the device selector for the currently assigned GRAPHICS INPUT IS device. Zero is returned if no device is currently selected. (Requires GRAPH)
KBD LINE	A string containing the current contents of the keyboard input line(s). Note that this operation does not change the contents of the line(s).
KEYBOARD LANGUAGE	ASCII, BELGIAN, CANADIAN ENGLISH, CANADIAN FRENCH, DANISH, DUTCH, FINNISH, FRENCH, GERMAN, ITALIAN, KATAKANA, LATIN, NORWEGIAN, SPANISH, SWEDISH, SWISS FRENCH, SWISS GERMAN, SWISS FRENCH*, SWISS GERMAN*, or UNITED KINGDOM (Requires LEX)
LEXICAL ORDER IS	ASCII, GERMAN, FRENCH, SPANISH, SWEDISH or USER DEFINED (Requires LEX)
MASS MEMORY	X000YZ0000000000 X=Number of internal disc drives Y=Number of initialized EPROM cards Z=Number of bubble memory cards If Y or Z exceed 9, an asterisk appears. BASIC/UX: value is always a string of 0's as internal disk drives, EPROM or bubble memory cards are not supported.

Topic Specifier	Information Returned
MASS STORAGE IS MSI	The mass storage unit specifier of the current MASS STORAGE IS device, as it appears in a CAT heading.
PLOTTER IS	A string containing numerals which specify the device selector of the current PLOTTER IS device or the path name of the current PLOTTER IS file. (Requires GRAPH)
PRINTALL IS	A string containing numerals which specify the device selector of the current PRINTALL IS device.
PRINTER IS	A string containing numerals which specify the device selector of the current PRINTER IS device or the path name of the current PRINTER IS file.
SERIAL NUMBER	If an ID PROM is present, this string contains bytes 4-14 of that PROM. If an ID MODULE is present (ID MODULE requires KBD), the string contains encoded information. (See the "Software Security" section in the "Editing Programs" chapter of *Using the BASIC System.*) Otherwise, a null string is returned. (Requires KBD.)
SYSTEM ID	`S300:30` on Series 300 computers with an MC68020 processor; or `S300:20` on Series 300 computers with an MC68010 processor; or `S300:10` on Series 300 computers with an MC68010 processor; or bytes 15 thru 21 of the ID PROM in a Series 200 computer (if present); or `9816, 9826A, or 9836A` padded with trailing spaces to make a seven character string.
SYSTEM PRIORITY	A string containing numerals which specify the current system priority.
TIMEZONE IS	A string specifying the seconds from Greenwich Mean Time that represent the current timezone value.
TRIG MODE	DEG or RAD
VERSION: *binary name*	A string containing numerals which specify the revision number of the specified binary (also displayed after LOAD BIN or LIST BIN).

SYSTEM$ with SRM and HFS Systems

When SYSTEM$ of MASS STORAGE IS (MSI), PLOTTER IS, or PRINTER IS is executed on a system using SRM or HFS volumes, the information returned includes the full file specifier describing the file or directory about which the information is requested. (SRM passwords are not included in the specifier.)

The system remembers a maximum of 160 characters for any one specifier. If a specifier contains more than 160 characters, the excess characters are removed from the *beginning* of the specifier and are not retained. An asterisk (*) as the left-most character in the specifier indicates that leading characters were truncated for the function.

TAB

See the PRINT and DISP statements.

TABXY

See the PRINT statement.

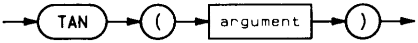
TALK

See the SEND statement.

TAN

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the tangent of the angle represented by the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression in the current units of angle when arguments are INTEGER or REAL. numeric expression in radians when the argument is COMPLEX.	absolute values less than: 8.541 563 906 E+9 deg. or 1.490 784 13 E+8 rad. for INTEGER and REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments

Examples Statements

```
Tangent=TAN(Angle)
PRINT "Tangent of ";Z;"=";TAN(Z)
```

Semantics

Error 31 is reported for INTEGER and REAL arguments when trying to compute the TAN of an odd multiple of 90 degrees.

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 TAN of a COMPLEX value, the COMPLEX binary must be loaded.

Range Restriction Specifics

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

$$CMPLX\left(\frac{SIN(2*Real\_part)}{COS(2*Real\_part)+COSH(2*Imag\_part)},\frac{SINH(2*Imag\_part)}{COS(2*Real\_part)+COSH(2*Imag\_part)}\right)$$

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

```
TAN(CMPLX(0,710))
```

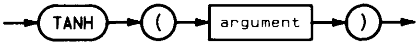
will cause error 22 due to the `COSH(2*Imag_part)` calculation.

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`).

TANH

Supported On	WS,UX
Option Required	COMPLEX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the hyperbolic tangent of a numeric expression.



Item	Description	Range Restrictions
argument	numeric expression	any value for INTEGER or REAL arguments; see "Range Restriction Specifics" for COMPLEX arguments.

Example Statements

```
Result=TANH(-5.7723)
PRINT "Hyperbolic Tangent = ";TANH(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

For COMPLEX arguments, the formula for computing TANH is:

$$CMPLX\left(\frac{SINH(2*Real\_part)}{COSH(2*Real\_part)+COS(2*Imag\_part)},\frac{SIN(2*Imag\_part)}{COSH(2*Real\_part)+COS(2*Imag\_part)}\right)$$

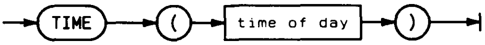
where `Real_part` is the real part of the COMPLEX argument and `Imag_part` is the imaginary part. Some values of the argument may cause errors in this computation. For example:

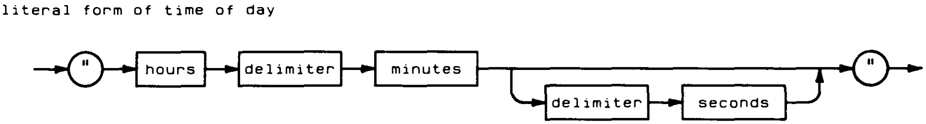
```
TANH(CMPLX(710,3))
```

will cause error 22 REAL overflow due to the SINH(2*Real_part) calculation.

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function converts the formatted time of day (HH:MM:SS), into the number of seconds past midnight. (For information on using TIME as a secondary keyword, see the OFF TIME, ON TIME, and SET TIME statements.)





Item	Description	Range
time of day	string expression representing the time in 24-hour format	(see drawing)
hours	literal	0 thru 23
minutes	literal	0 thru 59
seconds	literal; default = 0	0 thru 59.99
delimiter	literal; single character	(see text)

Example Statements

```
Seconds=TIME(T$)
SET TIME TIME("8:37:30")
ON TIME TIME("12:12") GOSUB Food_food
```

Semantics

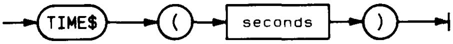
TIME returns a REAL whole number, in the range 0 thru 86 399, equivalent to the number of seconds past midnight.

While any number of non-numeric characters may be used as a delimiter, a single colon is recommended. Leading blanks and non-numeric characters are ignored.

TIME$

Supported On	WS,UX
Option Required	CLOCK
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function converts the number of seconds past midnight into a string representing the time of day (HH:MM:SS).



Item	Description	Range
seconds	numeric expression, truncated to the nearest second; seconds past midnight	0 thru 86 399

Example Statements

```
DISP "The time is:  ";TIME$(TIMEDATE)
PRINT TIME$(45296)
```

Semantics

TIME$ takes time (in seconds) and returns the time of day in the form HH:MM:SS, where HH represents hours, MM represents minutes, and SS represents seconds. A modulo 86 400 is performed on the parameter before it is formatted as a time of day.

TIMEDATE

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This function returns the current value of the real-time clock. (Also see the SET TIMEDATE statement.)

Example Statements

```
Elapsed=TIMEDATE-T0
DISP TIMEDATE MOD 86400
```

Semantics

The value returned by TIMEDATE represents the sum of the last time setting and the number of seconds that have elapsed since that setting was made. The volatile clock value set at power-on is 2.086 629 12 E+11, which represents midnight March 1, 1900 (for BASIC/UX, the power-on value was the HP-UX time). If there is a battery-backed (non-volatile) clock, then the volatile clock is synchronized with it at power-up. If the computer is on an SRM system (and has no battery-backed clock), then the volatile clock is synchronized with the SRM clock when the SRM and DCOMM binaries are loaded. The clock values represent Julian time, expressed in seconds. The time value accumulates from that setting unless it is changed by SET TIME or SET TIMEDATE.

The resolution of the TIMEDATE function is .01 seconds. If the clock is properly set, `TIMEDATE MOD 86400` gives the number of seconds since midnight.

See also TIMEZONE IS.

BASIC/UX Specifics

Resolution is limited to 20 milliseconds.

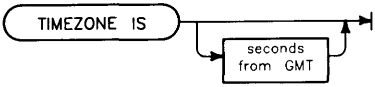
TIMEOUT

See the OFF TIMEOUT and ON TIMEOUT statements.

TIMEZONE IS

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement specifies the offset from Greenwich Mean Time.



Item	Description	Range
seconds from GMT	numeric expression rounded to the nearest hundredth	0 thru ±86 399.99 (=24*60*60−0.01)

Example Statements

```
TIMEZONE IS Hours_from_GMT*60*60
TIMEZONE IS -7*3600  (Mountain Standard Time)
TIMEZONE IS          (BASIC/UX only)
```

Semantics

Workstation BASIC Specifics

TIMEZONE specifies the number of seconds that will be *added* to the clock to calculate the "local" time (when it is set to Greenwich Mean Time, or GMT). Therefore TIMEZONE IS parameter's value for the GMT timezone is 0. The TIMEZONE IS parameter's value for the Mountain Standard timezone is −7×60×60, because it is 7 hours *behind* GMT. For each one-hour timezone to the east, add 3600 seconds to the parameter's value; for each timezone to the west, subtract 3600 seconds.

You can determine the current value of the TIMEZONE IS parameter by executing SYSTEM$("TIMEZONE IS"). See SYSTEM$ for details.

Note

If you have a battery-backed (non-volatile) clock, then you may need to first use SET TIMEDATE before using TIMEZONE IS and SET TIMEDATE as described above. Otherwise, the clock may initially be set to 1 March 1900, and SET TIMEDATE could generate a "parameter out of range" error (when it *subtracts* the TIMEZONE's "offset from GMT" parameter from the specified clock value while calculating the GMT value to put into the clock register.)

You can use STATUS register 4 of select code 32 to determine whether or not you have a battery-backed clock.

HP-UX Compatibility

This statement provides compatibility with HP-UX time stamps on files when switching back and forth between the BASIC and HP-UX operating systems. *(If you will not be doing that, you do* **not** *need to use the TIMEZONE statement.)*

TIMEZONE is required for HP-UX compatibility when:

- The non-volatile clock is set to Greenwich Mean Time for HP-UX.
- The real-time clock is set to "local" time for BASIC.

An HP-UX environment variable called TZ is used to calculate "local time", which is an offset from GMT. Thus, when a time stamp (in GMT) is put on a file by HP-UX, the time value (printed in a directory listing) is derived with this formula:

local time = HP-UX clock value (GMT) + TZ

When using TIMEZONE for HP-UX compatibility, you can set the non-volatile (battery-backed) clock to GMT by the following sequence of commands:

1. Specify the "local" offset to GMT with TIMEZONE IS. For example:

   ```
   TIMEZONE IS -7*3600
   ```

2. Set the "local" time with SET TIMEDATE. For example:

   ```
   SET TIMEDATE DATE("5 Dec 1986")+TIME("09:00:00")
   ```

   (The actual value written into the battery-backed clock is the specified time *minus* the TIMEZONE IS value.)

Note also that LIF volumes have "local time" stamps, while HFS volumes have GMT time stamps.

BASIC/UX Specifics

The TIMEZONE is set to the current HP-UX timezone in effect at the start of BASIC. Daylight savings time is automatically included. Any changes in timezone that occur after BASIC has started must be accounted for by the user with the TIMEZONE IS statement.

Note that this statement does NOT reset the HP-UX timezone, even if the user is super-user. Instead it resets the timezone which BASIC keeps for itself

TIMEZONE IS without a parameter resynchronizes the timezone with the current HP-UX timezone in effect (this does take into account Daylight Savings Time changes). This command will affect any previous SET TIME or SET TIMEDATE statements. The proper way to resynchronize the time, date, and timezone is to do the timezone first as in:

```
TIMEZONE IS
SET TIMEDATE
```

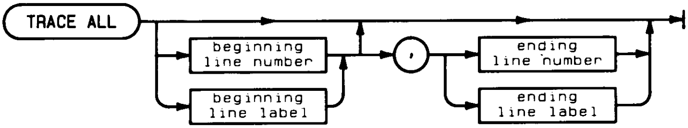
You can determine the current value of the TIMEZONE IS parameter by executing SYSTEM$("TIMEZONE IS"). See SYSTEM$ for details.

Workstation Compatibility

This statement provides backward compatibility to the BASIC workstation. It is intended to provide compatibility with HP-UX time stamps on files when switching back and forth between the BASIC and HP-UX operating systems. Since the BASIC Workstation default timezone is synchronized with HP-UX at start-up time, this statement is generally NOT needed when working with BASIC Workstation.

Supported On	WS,UX
Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement allows tracing program flow and variable assignments during program execution.



Item	Description	Range
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
TRACE ALL Sort
TRACE ALL 1500,2450
```

Semantics

The entire program, or any part delimited by beginning and (if needed) ending line numbers or labels, may be traced.

Tracing starts when the beginning line is first executed and continues until the ending line is executed.

The ending line is not included in the trace output. The trace output stops immediately before the ending line is executed. When a line is traced, the line number and any variable which receives a new value is output to the system message line of the CRT. Any type of variable (string, numeric or array) can be displayed. For simple string and numeric variables, the name and the new value are displayed. For arrays, a message is displayed stating that the array has a new value rather than outputting the entire array contents.

TRACE ALL output can also be printed on the PRINTALL printer, if PRINTALL is ON. TRACE ALL is disabled by TRACE OFF. The line numbers specified for TRACE ALL are not affected by REN.

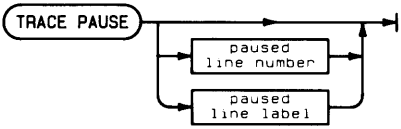
Supported On	WS,UX
Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement turns off all tracing activity.

TRACE PAUSE

Supported On	WS,UX
Option Required	PDEV
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement causes program execution to pause before executing the specified line, and displays the next line to be executed on the CRT.



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

Example Statements

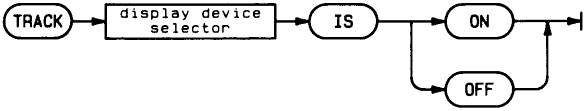
```
TRACE PAUSE
TRACE PAUSE Loop_end
```

Semantics

Not specifying a line for TRACE PAUSE results in the pause occurring before the next line is executed. Only one TRACE PAUSE can be active at a time. TRACE PAUSE is cancelled by TRACE OFF.

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement enables and disables tracking of the current locator position on the current display device.



Item	Description	Range
display device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
TRACK 709 IS ON
TRACK Plot IS OFF
```

Semantics

The current locator is defined by a GRAPHICS INPUT IS statement, and the current display device is defined by a PLOTTER IS statement. If TRACK...IS ON is executed, an echo on the current display device tracks the locator position during DIGITIZE statements. On a CRT, the echo is a 9-by-9-dot crosshair. On a plotter, the pen position tracks the locator. When a point is digitized, the echo is left at the location of the digitized point and tracking ceases.

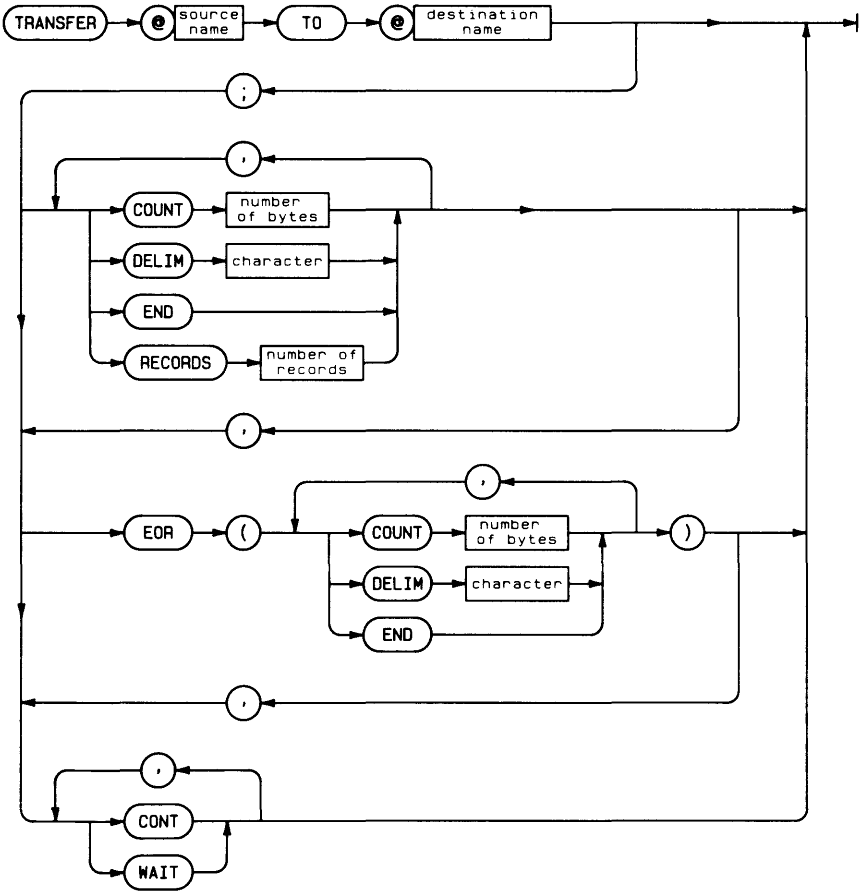
The display device selector must match that used in the most recently executed PLOTTER IS statement, or error 708 results.

Executing TRACK...IS OFF disables tracking.

TRANSFER

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement initiates unformatted I/O transfers.



Item	Description	Range
source name	I/O path name assigned to a device, a group of devices, a mass storage file, pipe, or a buffer	any valid name
destination name	I/O path name assigned to a device, a group of devices, a mass storage file, pipe, or a buffer	any valid name
number of bytes	numeric expression, rounded to an integer	1 thru $2^{31}-1$
character	string expression with a length of zero or one	—
number of records	numeric expression, rounded to an integer	1 thru $2^{31}-1$

Example Statements

```
TRANSFER @Device TO @Buff
TRANSFER @Buff TO @File;CONT
TRANSFER @Path TO @Destination;COUNT 256
TRANSFER @Source TO @Buffer;DELIM "/"
TRANSFER @Path TO @Buffer;RECORDS 12,EOR(COUNT 8)
```

Semantics

The TRANSFER statement allows unformatted data transfers between the computer and devices (mass storage drives are considered devices for this operation). Whenever possible, a TRANSFER takes place concurrently with continued program execution. Since no formatting is performed and the TRANSFER statement executes concurrently (overlapped) with regular program execution, the highest possible data transfer rate is achieved.

Before a data transfer can take place, a buffer must be declared. Every TRANSFER will need a buffer as either its source or its destination. An outbound TRANSFER empties the buffer (source) while an inbound TRANSFER fills the buffer (destination). Device to device transfers and buffer to buffer transfers are not allowed.

Two types of buffers are available; named and unnamed. A named buffer is a REAL array, INTEGER array, COMPLEX array, or a string scalar declared with the keyword BUFFER. See ASSIGN, COM, DIM, INTEGER, COMPLEX, and REAL. Unnamed buffers are created in the ASSIGN statement by specifying the keyword BUFFER and the number of bytes to be reserved for the buffer. See ASSIGN.

Every buffer has two pointers associated with it. The fill pointer indicates the next available location in the buffer for data. The empty pointer indicates the next item to be removed from the buffer. This allows an inbound TRANSFER and an outbound TRANSFER to access the same buffer simultaneously.

BDAT and HP-UX files are the only file types allowed in a TRANSFER. An end-of-file error will prematurely terminate a TRANSFER, thus triggering an end-of-transfer condition. If an end-of-record condition was satisfied when the end-of-file was reached, the EOR event will also be true.

I/O path names should be used to access the contents of the buffer. This ensures the automatic updating of the fill and empty pointers during a transfer. For named buffers, the contents of the buffer can also be accessed by the buffer's variable name. However, accessing the contents of the buffer by the variable name does not update the fill and empty pointers and is likely to corrupt the data in the buffer.

TRANSFER with HFS and SRM Files

With files on HFS and SRM volumes, the TRANSFER statement runs in overlapped mode until the BASIC system encounters a statement that accesses the same volume (such as CAT or ASSIGN); at such times, the BASIC system performs an implicit WAIT FOR EOT.

SRM is not supported for TRANSFER in BASIC/UX.

Transfer Parameters

When no parameters are specified for a TRANSFER, an inbound TRANSFER will fill the buffer with data and then terminate. An outbound transfer will empty the buffer and then terminate. Both inbound and outbound transfers execute in overlapped mode when possible.

The CONT parameter specifies that the TRANSFER is to continue indefinitely. Instead of terminating on buffer full or buffer empty conditions, the TRANSFER will be temporarily suspended until there is space available in the buffer (for inbound transfers) or until there is data available in the buffer (for outbound transfers).

The WAIT parameter specifies that the TRANSFER is to take place serially (non-overlapped). Program execution will not leave the TRANSFER statement until the data transfer is completed.

A TRANSFER can be specified to terminate when a device dependent signal is received (END), after a specified number of bytes has been transferred (COUNT), or after a specific character is detected (DELIM). The DELIM parameter can only be used with inbound transfers.

If `END` is included on a TRANSFER to a file, the end-of-file pointer is updated when the TRANSFER terminates; including `EOR(END)` causes the end-of-file pointer to be updated at the end of each record.

When the RECORD parameter is specified, the end-of-record parameter must also be specified (EOR). The end-of-record condition can be either COUNT, DELIM, END or any combination of conditions.

Overlapped execution of the TRANSFER statement can be deferred until a record has been transferred or until the entire TRANSFER has completed. See WAIT FOR EOR and WAIT FOR EOT.

Supported Devices

The TRANSFER statement supports data transfers to and from the following devices.

HP-IB	(HP 98624)
GPIO	(HP 98622)
Serial	(HP 98626)
Datacomm	(HP 98628)
MUX (BASIC/UX only)	(HP 98642)

TRANSFER can also be used with BDAT and HP-UX files on any of the mass storage devices or pipes supported by BASIC.

Transfer Method (BASIC Workstation only)

The transfer method is device dependent and chosen by the computer. The three possible transfer modes are:

INT	interrupt mode
FHS	fast handshake
DMA	direct memory access

The DMA mode will be used whenever possible. If the DMA mode cannot be used (DMA card is not installed, both channels are busy, DELIM is specified, or the interface does not support DMA) then the INT mode will be used. FHS is used with the HP-IB or GPIO interfaces only when DMA cannot be used and the WAIT parameter is specified.

Interactions

When the computer tries to move into the stopped state, it will wait for any transfer to complete. Therefore, operations which would cause a stopped state will make the computer unresponsive (or "hung") if a TRANSFER is in progress. Operations in this category include a programmed GET, modifying a paused program, and STOP. Also, the computer will not exit a context until any TRANSFER in that context is complete. This will cause the program to wait at a SUBEXIT, ERROR SUBEXIT, SUBEND, or RETURN <expression> statement while a TRANSFER is in progress. If the program is paused, but a TRANSFER is still active, the run-light will be an "Io" character and the system status Indicator (if present) will say "Transfer."

To terminate a transfer before it has finished (and free the computer), execute an ABORT IO (or, as a last resort, press [RESET]).

See also: ASSIGN, WAIT FOR EOT, WAIT FOR EOR, ABORTIO, RESET and the "Advanced Transfer Techniques" chapter of the *BASIC Interfacing Techniques* manual.

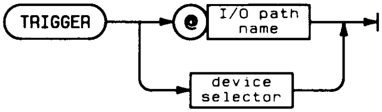
BASIC/UX Specifics

Either `io_burst` is used (if specified with `CONTROL` *isc*`,255;3`) or else DMA allocation is managed by the HP-UX kernel and may be used for some or all of the TRANSFER segments.

TRIGGER

Supported On	WS,UX
Option Required	IO
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement sends a trigger message to a selected device, or all devices addressed to listen, on the HP-IB.



Item	Description	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

```
TRIGGER 712
TRIGGER @Hpib
```

Semantics

The computer must be the active controller to execute this statement.

If only the interface select code is specified, all devices on that interface which are addressed to listen are triggered. If a primary address is given, the bus is reconfigured and only the addressed device is triggered.

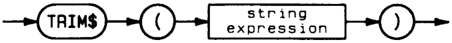
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN UNL LAG GET	ATN GET	ATN UNL LAG GET
Not Active Controller	Error			

TRIM$

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns the string stripped of all leading and trailing ASCII spaces.



Example Statements

```
Unjustify$=TRIM$("   center   ")
Clean$=TRIM$(Input$)
```

Semantics

Only leading and trailing ASCII spaces are removed. Embedded spaces are not affected.

TRN

See the MAT statement.

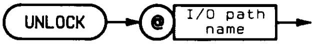
UNL

See the SEND statement.

UNLOCK

Supported On	WS,UX
Option Required	SRM,DCOMM, OR HFS
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement is used to remove exclusive access (placed by the LOCK statement) on an SRM or HFS file associated with an I/O path name (see ASSIGN).



Item	Description	Range
I/O path name	name identifying an I/O path to an SRM file	any valid name (see Glossary)

Example Statements

```
UNLOCK @File
IF Done THEN UNLOCK @File
```

Semantics

This statement unlocks a file previously locked with the LOCK statement. While a file is locked, other SRM workstations or HP-UX processes cannot access the file. After UNLOCK, other users may access the file provided they possess the proper access capability (or capabilities).

If multiple LOCKs were executed on the file, the same number of UNLOCKs must be executed to unlock the file.

UNLOCK is performed automatically by SCRATCH A, SCRATCH BIN, RESET and ASSIGN...TO * (explicit closing of an I/O path).

BASIC/UX Specifics

Since LOCK is not available for LIF on BASIC/UX, UNLOCK is not supported for LIF on BASIC/UX. However, no error is generated when a LOCK is attempted on a LIF file.

UNT

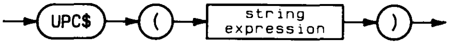
See the SEND statement.

UNTIL

See the REPEAT...UNTIL construct.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function replaces any lowercase characters with their corresponding uppercase characters.



Example Statements

```
Capital$=UPC$("lower")
IF UPC$(Name$)="TOM" THEN Equal_tom
```

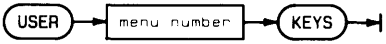
Semantics

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

USER KEYS

Supported On	WS,UX
Option Required	KBD
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement changes the softkey definitions on an ITF keyboard to one of three User softkey menus.



Item	Description	Range
menu number	numeric expression, rounded to an integer	1 thru 3

Example Statements

```
USER Menu_number KEYS
IF Change_keys THEN USER 1 KEYS
```

Semantics

This statement only affects the normal mode of the ITF Keyboard (i.e. it does nothing on an HP 98203A/B/C Keyboard and causes no visible change on an ITF Keyboard when the Keyboard Compatibility mode is on).

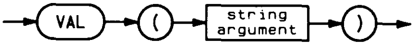
Note that the functionality of this statement can be achieved through KBD CONTROL register 2.

USING

See the DISP, ENTER, LABEL, OUTPUT, and PRINT statements.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function converts a string expression into a numeric value.



Item	Description	Range
string argument	string expression	numerals, decimal point, sign and exponent notation

Example Statements

```
Day=VAL(Date$)
IF VAL(Response$)<0 THEN Negative
```

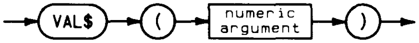
Semantics

The first non-blank character in the string must be a digit, a plus or minus sign, or a decimal point. The remaining characters may be digits, a decimal point, or an E, and must form a valid numeric constant. If an E is present, characters to the left of it must form a valid mantissa, and characters to the right must form a valid exponent. The string expression is evaluated when a non-numeric character is encountered or the characters are exhausted.

VAL$

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This function returns a string representation of the value of the argument. The returned string is in the default print format, except that the first character is not a blank for positive numbers. No trailing blanks are generated.



Item	Description	Range
numeric argument	numeric expression	—

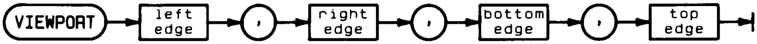
Example Statements

```
PRINT Esc$;VAL$(Cursor-1)
Special$=Text$&VAL$(Number)
```

VIEWPORT

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement defines an area onto which WINDOW and SHOW statements are mapped. It also sets the soft clip limits to the boundaries it defines.



Item	Description	Range
left edge	numeric expression	—
right edge	numeric expression	>left edge
bottom edge	numeric expression	—
top edge	numeric expression	>bottom edge

Example Statements

```
VIEWPORT 0,35,50,80
VIEWPORT Left,Right,Bottom,Top
```

Semantics

The parameters for VIEWPORT are in Graphic Display Units (GDUs). Graphic Display Units are 1/100 of the shorter axis of a plotting device. The units are isotropic (the same length in X and Y). The soft clip limits are set to the area specified, and the units defined by the last WINDOW or SHOW are mapped into the area.

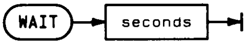
For the plotter specifier "INTERNAL" (the CRT), the shorter axis is Y. The longer axis is X, which is 100×RATIO GDUs long. For the plotter specifier "HPGL" (which deals with devices other than the CRT), the RATIO function may be used to determine the ratio of the length of the X axis to the length of the Y axis. If the ratio is greater than one, the Y axis is 100 GDUs long, and the length of the X axis is 100×RATIO. If the ratio is less than one, then the length of the X axis is 100 GDUs and the length of the Y axis is 100×RATIO.

A value of less than zero for the left edge or bottom is treated as zero. A value greater than the hard clip limit is treated as the hard clip limit for the right edge and the top. The left edge must be less than the right edge, and the bottom must be less than the top, or error 704 results.

WAIT

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement will cause the computer to wait approximately the number of seconds specified before executing the next statement. Numbers less than 0.001 do not generate a WAIT interval.



Item	Description	Range
seconds	numeric expression, rounded to the nearest thousandth	less than 2 147 483.648

Example Statements

```
WAIT 3
WAIT Old_time/2
```

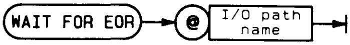
BASIC/UX Specifics

Resolution is limited to 20 milliseconds. Accuracy depends on system load and real time priority, but is generally 40 milliseconds.

WAIT FOR EOR

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement waits until an end-of-record event occurs in the TRANSFER on the specified I/O path.



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

Example Statements

```
WAIT FOR EOR @File
WAIT FOR EOR @Device
```

Semantics

The I/O path may be assigned either to a device, a group of devices, a pipe, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the WAIT FOR EOR statement is executed.

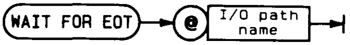
The WAIT FOR EOR statement prevents further program execution until an end-of-record event occurs in the TRANSFER whose I/O path name was specified. This allows ON EOR events, which might otherwise be missed, to be serviced. If the system priority prevents the servicing of an ON EOR event, the event will be logged.

The I/O path specified must be involved in an active TRANSFER for the statement to have any effect.

WAIT FOR EOT

Supported On	WS,UX
Option Required	TRANS
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement waits until the TRANSFER on the specified I/O path is completed.



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

Example Statements

```
WAIT FOR EOT @File
WAIT FOR EOT @Device
```

Semantics

The I/O path may be assigned either to a device, a group of devices, a pipe, or to a mass storage file. If the I/O path is assigned to a BUFFER, an error is reported when the WAIT FOR EOT statement is executed.

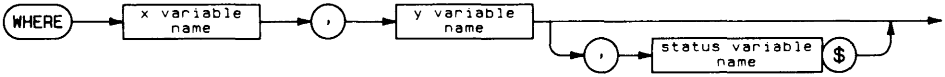
The WAIT FOR EOT statement prevents further program execution until the specified TRANSFER is completed. This allows ON EOT events, which might otherwise be missed, to be serviced. If the system priority prevents the servicing of an ON EOT event, the event will be logged.

The I/O path specified must be involved in an active TRANSFER for the statement to have any effect.

WHERE

Supported On	WS,UX
Option Required	GRAPHX
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement returns the current logical position of the pen and, optionally, pen status information.



Item	Description	Range
x variable name	name of a numeric variable	any valid name
y variable name	name of a numeric variable	any valid name
status variable name	name of a string variable whose dimensioned length is at least 3	any valid name

Example Statements

```
WHERE X,Y
WHERE X_position,Y_position,Status$
```

Semantics

The characters in the status string may be interpreted as follows:

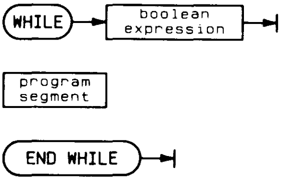
Byte 1	Byte 2	Byte 3
Pen Status	,	Point Significance

Byte	Value	Meaning
1	"0"	Pen is up
	"1"	Pen is down
2	comma	(delimiter)
3	"0"	Current position is outside hard clip limits.
	"1"	Current position is inside hard clip limits but outside viewport boundary.
	"2"	Current position is inside viewport boundary and hard clip limits.

WHILE

Supported On	WS,UX
Option Required	None
Keyboard Executable	No
Programmable	Yes
In an IF...THEN	No

This construct defines a loop which is executed as long as the boolean expression in the WHILE statement evaluates to true (evaluates to a non-zero value).



Item	Description	Range
boolean expression	numeric expression: evaluated as true if non-zero and false if zero.	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
840   WHILE Value<Min OR Value>Max
850     BEEP
860     INPUT "Out of range; RE-ENTER",Value
870   END WHILE
```

```
1220  WHILE P<=LEN(A$)
1230    IF NUM(A$[P])<32 THEN
1240      A$[P]=A$[P+1]  ! Remove control codes
1250    ELSE
1260      P=P+1          ! Go to next character
1270    END IF
1280  END WHILE
```

Semantics

The WHILE...END WHILE construct allows program execution dependent on the outcome of a relational test performed at the **start** of the loop. If the condition is true, the program segment between the WHILE and END WHILE statements is executed and a branch is made back to the WHILE statement. The program segment will be repeated until the test is false. When the relational test is false, the program segment is skipped and execution continues with the first program line after the END WHILE statement.

Branching into a WHILE...END WHILE construct (via a GOTO) results in normal execution up to the END WHILE statement, a branch back to the WHILE statement, and then execution as if the construct had been entered normally.

Nesting Constructs Properly

WHILE...END WHILE constructs may be nested within other constructs, provided the inner construct begins and ends before the outer construct can end.

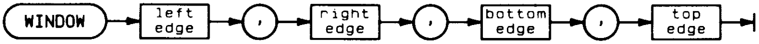
WIDTH

See the PRINTALL IS and PRINTER IS statements.

WINDOW

Supported On	WS,UX
Option Required	GRAPH
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN	Yes

This statement is used to define the current-unit-of-measure for graphics operations.



Item	Description	Range
left edge	numeric expression	—
right edge	numeric expression	≠ left edge
bottom edge	numeric expression	—
top edge	numeric expression	≠ bottom edge

Example Statements

```
WINDOW -5,5,0,100
WINDOW Left,Right,Bottom,Top
```

Semantics

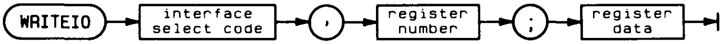
WINDOW defines the values represented at the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. WINDOW may be used to create non-isotropic (not equal in X and Y) units. The direction of an axis may be reversed by specifying the left edge greater than the right edge, or the bottom edge greater than the top edge. (Also see SHOW.)

WORD

See the ASSIGN statement.

Supported On	WS,UX
Option Required	None
Keyboard Executable	Yes
Programmable	Yes
In an `IF...THEN`	Yes

This statement writes an integer representation of the register-data parameter into the specified hardware register on the specified interface, or into memory. The actual action resulting from this operation depends on the interface and register (or memory address) selected.



Item	Description	Range

select code	numeric expression, rounded to an integer	1 thru 31;
		−31 thru −1;
		±9826; 9827

register number or memory address	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$ (hardware-dependent)
register or memory data	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$

Example Statements

```
WRITEIO 12,0;Set_pctl
WRITEIO Hpib,23;12
WRITEIO 9826,Mem_addr;Poke_byte
WRITEIO 9827,Jsr_address;D0_data
```

Semantics

A positive select code (appropriate for most interfaces), writes a byte of data to the register, and a negative select code writes a word of data to the register.

Writing Memory ("Poke")

Using a select code value of 9826 allows you to write directly into memory addresses.

`WRITEIO 9826,Mem_address; Data_byte` *writes a byte of data*

`WRITEIO -9826,Mem_address; Data_word` *writes a word of data*

The second parameter specified in the WRITEIO statement is the memory address of the byte or word to be written. This parameter is interpreted as a decimal address; for instance, an address of 100 000 is 10^5, not 2^20. The third parameter is also interpreted as a decimal number.

CAUTION

WRITING INTO CERTAIN MEMORY ADDRESSES WILL DAMAGE YOUR COMPUTER'S HARDWARE. IN ORDER TO AVOID THIS, YOU SHOULD ONLY WRITE INTO NUMERIC ARRAY VARIABLES WITH WRITEIO. HP CANNOT BE HELD LIABLE FOR ANY DAMAGES CAUSED BY IMPROPER USE OF THIS FEATURE.

For a description of the architecture of the computer, see the *Pascal System Designer's Guide.*

Calling Machine-Language Routines

Using a select code value of 9827 allows you to execute a machine-language JSR ("Jump to SubRoutine") instruction. One parameter must be specified in the WRITEIO statement (`D0_data` in the example below), which will be written into the processor's D0 register before the JSR instruction is executed. The following program provides a framework for placing a machine-language subroutine in an INTEGER array and then jumping to this subroutine.

```
10   DATA (INTEGER values of machine-language
20   DATA  instructions go here.)
      .
      .
100  INTEGER Int_array(1:100)
110  READ Int_array(*),D0_data          ! Read instructions
115  !                                    and D0 register data.
120  Jsr_addr=READIO(9827,Int_array(1)) ! Get JSR address.
130  WRITEIO 9827,Jsr_addr;D0_data      ! Put data in D0, then do JSR.
140  PRINT "Returned from subroutine."
      .
      .
```

BASIC first keeps a copy of processor registers A2 through A6 on the stack. Then the value represented by the expression `D0_data` is placed in the D0 register, and a machine-language JSR instruction is executed. The value of the expression `Jsr_addr` is the address of an INTEGER array that contains machine-language instructions. The value of `Jsr_addr` is forced to an even address before the JSR is executed.

The last instruction in the subroutine should return control to BASIC with a RTS ("ReTurn from Subroutine") instruction. BASIC will first restore the processor registers A2 through A6 (from the stack) to the state they were in before the JSR was performed (by the WRITEIO statement). Register A7 (the stack pointer) must have the same value at the final RTS as it had when BASIC executed the JSR. The other processor register can be used freely in the assembly routine. BASIC then resumes program execution at the line following the WRITEIO statement.

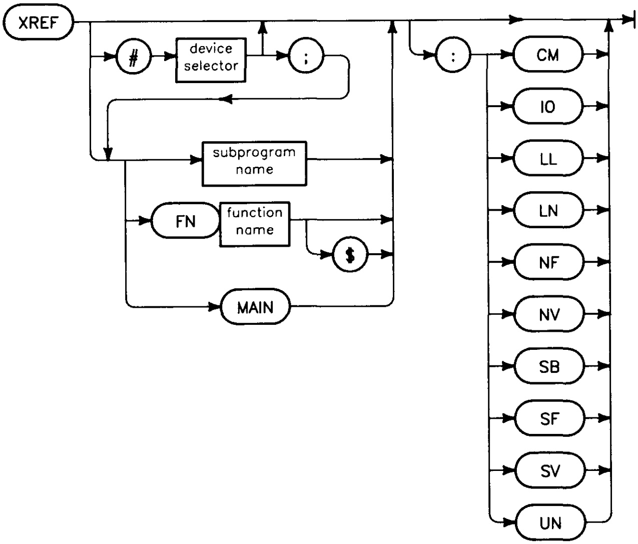
BASIC/UX Specifics

You can write only to your own process' data space.

Notes

Supported On	WS,UX
Option Required	XREF
Keyboard Executable	Yes
Programmable	No
In an `IF...THEN`	No

This command allows you to obtain a cross-reference listing of the identifiers in a program or subprogram.



Item	Description	Range
device selector	numeric expression; rounded to an integer Default = PRINTER IS device	(see Glossary)
subprogram name	name of a SUB subprogram currently in memory	any valid name
function name	name of a user-defined function currently in memory	any valid name

Example Statements

```
XREF
XREF #705;FNUser$
XREF Print
XREF :NV
```

Semantics

The cross-reference listing is printed one context at a time, in the order that they occur in the program. The main program is listed first, followed by the subprograms.

The cross-reference listing starts with this line:

```
    >>>> Cross Reference <<<<
```

Before each subsequent program segment, this line is printed:

```
    >>>> Subprogram <<<<
```

followed by the line number of the first line in that context and the name of the context. If the subprogram is a user-defined function, an `FN` will precede the name, and if it is a string function, a `$` will follow its name.

Within each context, identifiers are listed by type. They occur in the following order:

- NV—Numeric Variables
- SV—String Variables
- IO—I/O Path Names
- LL—Line Labels
- LN—Line Numbers
- NF—Numeric Functions
- SF—String Functions
- SB—SUB Subprograms
- CM—Common Block Names
- UN—Unused Entries

If a type is specified in the command, only that type is printed. If there are no identifiers of a particular type in the context being cross-referenced, that heading is not printed.

Within each group (which is composed of a header telling what kind of entity follows, then the list of those entities), names are alphabetized according to the ASCII collating sequence, and line numbers are in numerical order. If a reference is a formal parameter in a SUB or DEF FN statement, declared in a COM, DIM, REAL, or INTEGER statement, or is a line label, the characters `<-DEF` will be printed immediately to the right of the line number containing the defining declaration. Note that variables declared by ALLOCATE are not given this marker. If unlabelled (blank) COM is used, it will have no name associated with it.

At the end of each context, a line is printed that begins with:

```
Unused entries =
```

This is a count of the symbol table entries which have been marked by a prerun as "unused." Unreferenced symbol table locations which have not yet been marked "unused" by the prerun processing will show up in the lists of identifiers with empty reference lists. Note that a subprogram that is not directly recursive will show up in its own cross-reference listing with an empty reference list. (See the "Debugging Programs" chapter of *BASIC Programming Techniques* for further details of "Unused entries".)

If a subprogram name or MAIN is specified in the XREF command, the above rules are followed, but only the specified subprogram or the MAIN program is cross-referenced. If there are two or more subprograms of the same name in the computer, they will all be cross-referenced.

An XREF can be aborted by pressing [RESET], [CLR I/O] or [Break].

Notes

Language History A

This manual documents the BASIC 5.0/5.1 Language System used on HP 9000 Series 200/300 computers. There are several versions (other than 5.X) of this language in use today. The following table is provided for those users who have more than one BASIC version, or who are upgrading to BASIC 5.0/5.1. The first column is a list of BASIC statements. The remaining columns show versions of BASIC with entries in a column indicating the optional binary file (BIN) you must load to use the keyword. Note that "n.a." in any of the columns indicates that the binary was not availible for that release of BASIC. A column without an entry in it indicates that the minimal form of this statement does not require any binaries.

Since BASIC/UX has all binaries permanently loaded, it is unnecessary to include any BASIC/UX specific information.

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
ABORT		IO	IO
ABORTIO	AP2.0	TRANS	TRANS
ABS			
ACS			
ACSH	n.a.	n.a.	COMPLEX
ALLOCATE			
ALPHA		GRAPH	GRAPH
ALPHA HEIGHT	n.a.	n.a.	CRTX
ALPHA PEN	n.a.	n.a.	CRTX
AND			
AREA	GRAPH2.1	GRAPHX	GRAPHX
ARG	n.a.	n.a.	COMPLEX
ASN			
ASNH	n.a.	n.a.	COMPLEX
ASSIGN			
ATN			
ATNH	n.a.	n.a.	COMPLEX
AXES		GRAPH	GRAPH
BASE	AP2.0	MAT	MAT
BEEP			
BINAND			
BINCMP			
BINEOR			
BINIOR			
BIT			
BREAK	AP2.0	IO	IO
CALL			
CAT			
CAUSE ERROR	n.a.	n.a.	
CDIAL	n.a.	n.a.	KBD
CHANGE	AP2.0	PDEV	EDIT & PDEV
CHECKREAD	AP2.0	MS	MS
CHGRP	n.a.	n.a.	HFS
CHOWN	n.a.	n.a.	HFS
CHR$			

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
CHRX	n.a.	n.a.	CRTX
CHRY	n.a.	n.a.	CTRX
CLEAR		IO	IO
CLEAR ERROR	n.a.	n.a.	
CLEAR LINE	n.a.	n.a.	CRTX
CLEAR SCREEN	n.a.	n.a.	CRTX
CLIP		GRAPH	GRAPH
CLS	n.a.	n.a.	CRTX
CMPLX	n.a.	n.a.	COMPLEX
COM			
COMPLEX	n.a.	n.a.	COMPLEX
CONJG	n.a.	n.a.	COMPLEX
CONT			
CONTROL			
COPY			
COPYLINES	AP2.0	PDEV	EDIT & PDEV
COS			
COSH	n.a.	n.a.	COMPLEX
CREATE	n.a.	n.a.	MS
CREATE ASCII			
CREATE BDAT			
CREATE DIR	SRM	SRM	
CRT	AP2.0		
CSIZE		GRAPH	GRAPH
DATA			
DATE	AP2.0	CLOCK	CLOCK
DATE$	AP2.0	CLOCK	CLOCK
DEALLOCATE			
DEF FN			
DEG			
DEL			
DELSUB			
DET	AP2.0	MAT	MAT
DIGITIZE	GRAPH2.0	GRAPHX	GRAPHX
DIM			

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
DISABLE			
DISABLE INTR		IO	IO
DISP			
DISPLAY FUNCTIONS ON/OFF	n.a.	n.a.	CRTX
DIV			
DOT	AP2.0	MAT	MAT
DRAW		GRAPH	GRAPH
DROUND			
DUMP ALPHA			
DUMP GRAPHICS		GRAPH	GRAPH
DUMP DEVICE IS		GRAPH	GRAPH
DVAL	AP2.0		
DVAL$	AP2.0		
EDIT			EDIT
EDIT KEY	AP2.0	KBD	KBD
ENABLE			
ENABLE INTR		IO	IO
END			
ENTER			
ERRDS	AP2.0		
ERRL			
ERRLN	n.a.	n.a.	
ERRM$	AP2.0		
ERRN			
ERROR RETURN	n.a.	n.a.	
ERROR SUBEXIT	n.a.	n.a.	
EXOR			
EXP			
FIND	AP2.0	PDEV	EDIT & PDEV
FN			
FOR...NEXT			
FRACT	AP2.0		
FRAME		GRAPH	GRAPH
GCLEAR		GRAPH	GRAPH
GESCAPE	GRAPH2.1	GRAPHX	GRAPHX

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
GET			
GINIT		GRAPH	GRAPH
GLOAD		GRAPH	GRAPH
GOSUB			
GOTO			
GRAPHICS		GRAPH	GRAPH
GRAPHICS INPUT IS	GRAPH2.0	GRAPHX	GRAPHX
GRID		GRAPH	GRAPH
GSEND	n.a.	n.a.	GRAPH
GSTORE		GRAPH	GRAPH
HILBUF$	n.a.	n.a.	KBD
HIL SEND	n.a.	n.a.	KBD
IDRAW		GRAPH	GRAPH
IF...THEN			
IMAG	n.a.	n.a.	COMPLEX
IMAGE			
IMOVE		GRAPH	GRAPH
INDENT	AP2.0	PDEV	PDEV
INITIALIZE			
INPUT			
INT			
INTEGER			
IPLOT		GRAPH	GRAPH
IPLOT array		GRAPHX	GRAPHX
IVAL	AP2.0		
IVAL$	AP2.0		
KBD	AP2.0		
KBD CMODE ON/OFF	n.a.	n.a.	CRTX
KBD LINE PEN	n.a.	n.a.	CRTX
KBD$			
KEY LABELS ON/OFF	n.a.	n.a.	CRTX
KEY LABELS PEN	n.a.	n.a.	CRTX
KNOBX			
KNOBY	n.a.		
LABEL		GRAPH	GRAPH

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
LDIR		GRAPH	GRAPH
LEN			
LET			
LEXICAL ORDER IS	AP2.0	LEX	LEX
LGT			
LINE TYPE		GRAPH	GRAPH
LINK	n.a.	n.a.	HFS
LINPUT			
LIST			EDIT
LIST BIN	n.a.		
LIST KEY	AP2.0	KBD	KBD
LOAD			
LOAD BIN			
LOAD KEY	AP2.0	KBD	KBD
LOADSUB			
LOCAL		IO	IO
LOCAL LOCKOUT		IO	IO
LOCK	SRM	SRM	SRM
LOG			
LOOP			
LORG		GRAPH	GRAPH
LWC$	AP2.0		
MASS STORAGE IS			
MAT	AP2.0	MAT	MAT
MAT REORDER	AP2.0	MAT	MAT
MAT SEARCH	n.a.	n.a.	MAT
MAT SORT	AP2.0	MAT	MAT
MAX	AP2.0	MAT	MAT
MAXLEN	n.a.	n.a.	
MAXREAL	n.a.		
MERGE ALPHA	n.a.	n.a.	GRAPH
MIN	AP2.0	MAT	MAT
MINREAL	n.a.		
MOD			
MODULO	n.a.		

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
MOVE		GRAPH	GRAPH
MOVELINES	AP2.0	PDEV	EDIT & PDEV
NOT			
NPAR			
NUM			
ON/OFF CDIAL	n.a.	n.a.	KBD
ON/OFF CYCLE	AP2.0	CLOCK	CLOCK
ON/OFF DELAY	AP2.0	CLOCK	CLOCK
ON/OFF END			
ON/OFF EOR	AP2.0	TRANS	TRANS
ON/OFF EOT	AP2.0	TRANS	TRANS
ON/OFF ERROR			
ON/OFF HIL EXT	n.a.	n.a.	KBD
ON/OFF INTR		IO	IO
ON/OFF KBD			
ON/OFF KEY			
ON/OFF KNOB			
ON/OFF SIGNAL	AP2.0	IO	IO
ON/OFF TIME	AP2.0	CLOCK	CLOCK
ON/OFF TIMEOUT			
ON			
OPTION BASE			
OR			
OUTPUT			
PASS CONTROL	AP2.0	IO	IO
PAUSE			
PEN		GRAPH	GRAPH
PENUP		GRAPH	GRAPH
PERMIT	n.a.	n.a.	HFS
PDIR	n.a.	GRAPH	GRAPH
PI			
PIVOT		GRAPH	GRAPH
PLOT		GRAPH	GRAPH
PLOT array	GRAPH2.1	GRAPHX	GRAPHX
PLOTTER IS		GRAPH	GRAPH

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
PLOTTER IS file	n.a.	GRAPH	GRAPH
POLYGON	GRAPH2.1	GRAPHX	GRAPHX
POLYLINE	GRAPH2.1	GRAPHX	GRAPHX
POS			
PPOLL		IO	IO
PPOLL CONFIGURE		IO	IO
PPOLL RESPONSE	AP2.0	IO	IO
PPOLL UNCONFIGURE		IO	IO
PRINT			
PRINT LABEL	n.a.	MS	MS
PRINT PEN	n.a.	n.a.	CRTX
PRINTALL IS			
PRINTER IS			
PRINTER IS file	n.a.		
PROTECT			
PROUND	AP2.0		
PRT	AP2.0		
PURGE			
RAD			
RANDOMIZE			
RANK	AP2.0	MAT	MAT
RATIO		GRAPH	GRAPH
READ			
READIO			
READ LABEL	n.a.	MS	MS
READ LOCATOR	GRAPH2.0	GRAPHX	GRAPHX
REAL (statement)			
REAL (function)	n.a.	n.a.	COMPLEX
RECTANGLE	GRAPH2.1	GRAPHX	GRAPHX
REDIM	AP2.0	MAT	MAT
REM			
REMOTE		IO	IO
REN			
RENAME			
REPEAT...UNTIL			

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
REQUEST	AP2.0	IO	IO
RE-SAVE			EDIT
RES	n.a.		
RESET	AP2.0	IO	IO
RESTORE			
RE-STORE			
RE-STORE BIN		n.a.	n.a.
RE-STORE KEY	AP2.0	KBD	KBD
RESUME INTERACTIVE			
RETURN			
REV$	AP2.0		
RND			
ROTATE			
RPLOT	GRAPH	GRAPH	GRAPH
RPLOT array	AP2.0	GRAPHX	GRAPHX
RPT$			
RUN			
SAVE			EDIT
SC	AP2.0		
SCRATCH			
SCRATCH R	n.a.	n.a.	
SCRATCH BIN	n.a.		
SCRATCH KEY	AP2.0	KBD	KBD
SECURE	n.a.	PDEV	PDEV
SELECT...CASE			
SEND		IO	IO
SEPARATE ALPHA	n.a.	n.a.	GRAPH
SET ALPHA MASK	n.a.	n.a.	CRTX
SET CHR	n.a.	n.a.	CRTX
SET DISPLAY MASK	n.a.	n.a.	CRTX
SET ECHO	GRAPH2.0	GRAPHX	GRAPHX
SET KEY	n.a.	n.a.	KBD
SET LOCATOR	n.a.	GRAPHX	GRAPHX
SET PEN	GRAPH2.1	GRAPHX	GRAPHX
SET TIME			

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
SET TIMEDATE			
SGN			
SHIFT			
SHOW		GRAPH	GRAPH
SIGNAL	AP2.0	IO	IO
SIN			
SINH	n.a.	n.a.	COMPLEX
SIZE	AP2.0	MAT	MAT
SOUND	n.a.	n.a.	KBD
SPOLL		IO	IO
SQR			
SQRT	n.a.	n.a.	
STATUS			
STOP			
STORE			
STORE BIN		n.a.	n.a.
STORE KEY	AP2.0	KBD	KBD
STORE SYSTEM	n.a.		
SUB			
SUBEXIT			
SUM	AP2.0	MAT	MAT
SUSPEND INTERACTIVE			
SYMBOL	GRAPH2.1	GRAPHX	GRAPHX
SYSBOOT	n.a.		
SYSTEM KEYS	n.a.	n.a.	CRTX
SYSTEM PRIORITY	AP2.0		
SYSTEM$	AP2.0		
("PLOTTER IS")	GRAPH2.0	GRAPH	GRAPH
("GRAPHICS INPUT IS")	GRAPH2.0	GRAPH	GRAPH
("LEXICAL ORDER IS")	AP2.0	LEX	LEX
("KEYBOARD LANGUAGE")	AP2.0	LEX	LEX
TAN			
TANH	n.a.	n.a.	COMPLEX
TIME	AP2.0	CLOCK	CLOCK
TIME$	AP2.0	CLOCK	CLOCK

Statement	BASIC 2.0/2.1	BASIC 3.0/4.0	BASIC 5.0/5.1
TIMEDATE			
TIMEZONE IS	n.a.	n.a.	
TRACE ALL		PDEV	PDEV
TRACE OFF		PDEV	PDEV
TRACE PAUSE		PDEV	PDEV
TRACK	GRAPH2.0	GRAPHX	GRAPHX
TRANSFER	AP2.0	TRANS	TRANS
TRIGGER		IO	IO
TRIM$	AP2.0		
UNLOCK	SRM	SRM	SRM
UPC$	AP2.0		
USER i KEYS	n.a.	n.a.	CRTX
VAL			
VAL$			
VIEWPORT		GRAPH	GRAPH
WAIT			
WAIT FOR EOR	AP2.0	TRANS	TRANS
WAIT FOR EOT	AP2.0	TRANS	TRANS
WHERE	GRAPH2.1	GRAPHX	GRAPHX
WHILE			
WINDOW		GRAPH	GRAPH
WRITEIO			
XREF	AP2.0	XREF	XREF

Notes

Glossary

access capability See "SRM password."

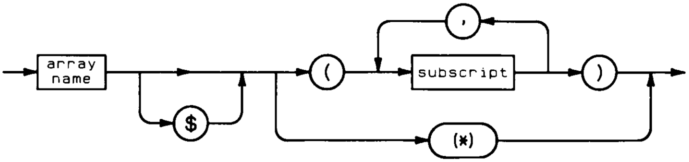
angle mode The current units used for expressing angles. Either degrees or radians may be specified, using the DEG or RAD statements, respectively. The default at power-on and SCRATCH A is radians.

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

array A structured data type that can be of type REAL, INTEGER, COMPLEX, or string. Arrays are created with the DIM, REAL, INTEGER, COMPLEX, ALLOCATE, or COM statements. Arrays have 1 to 6 dimensions; each dimension is allowed 32 767 elements. The lower and upper bounds for each dimension must fall in the range −32 767 (−32 768 for ALLOCATE) thru +32 767, and the lower bound must not exceed the upper bound. The default lower bound is the OPTION BASE value; the OPTION BASE statement can be used to specify 0 or 1 as the default lower bound. The default OPTION BASE in every environment is zero.

Each element in a string array is a string whose maximum length is specified in the declaring statement. The declared length of a string must be in the range 1 thru 32 767.

To specify an entire array, the characters `(*)` are placed after the array name. To specify a single element of an array, subscripts are placed in parentheses after the array name. Each subscript must not be less than the current lower bound or greater than the current upper bound of the corresponding dimension.



If an array is not explicitly dimensioned. it is implicitly given the number of dimensions used in its first occurrence, with an upper bound of 10. Undeclared strings have a default length of 18.

ASCII This is the acronym for "American Standard Code for Information Interchange". It is a commonly used code for representing letters, numerals, punctuation, special characters, and control characters. A table of the characters in the ASCII set and their code values can be found in the back of this manual.

bit This term comes from the words "binary digit". A bit is a single digit in base 2 that must be either a 1 or a 0.

byte A group of eight bits processed as a unit.

command A statement that can be typed on the input line and executed (see "statement").

COMPLEX A complex number is an ordered pair (x,y) denoted by Mathematicians as:

$$\mathrm{x} + \mathrm{y}i$$

where:

- x is the real part of the complex number.
- y is the imaginary part of the complex number. The product y*i* represents the value obtained by multiplying y and $\sqrt{-1}$. For example, the $\sqrt{-9}$ could be written as 3*i*.

context An instance of an environment. A context consists of a specific instance of all data types and system parameters that may be accessed by a program at a specific point in its execution. Context changes occur when subprograms are invoked or exited.

device selector A numeric expression used to specify the source or destination of an I/O operation. A device selector can be either an interface select code or a combination of an interface select code and an HP-IB primary address. To construct a device selector with a primary address, multiply the interface select code by 100 and add the primary address. For instance, a device selector that specifies the device at address 1 on interface select code 7 is 701. The device at address 0 on interface select code 14 is 1400. Device selector 1516 selects interface select code 15 and primary address 16.

Secondary addresses may be appended after a primary address by multiplying the device selector by 100 and adding the address. This may be repeated up to 6 times, adding a new secondary address each time. A device selector, once rounded, may contain a maximum of 15 digits. For example, 70502 selects interface 7, primary address 05, and secondary address 02.

When a device selector contains an odd number of digits, the leftmost digit is the interface select code. For an even number of digits, the leftmost two digits are the interface select code.

directory name A directory name specifies a directory of files on a hierarchically structured mass storage volume.

- A directory name on a Shared Resource Manager (SRM) volume consists of 1 to 16 characters, which may include all ASCII characters except "/" and ":" and "<". Spaces are ignored.
- A directory name on a Hierarchical File System (HFS) volume consists of 1 to 14 characters for short file name systems, and up to 255 characters for long file name systems (BASIC/UX only), which may include all ASCII characters except "/" and ":" and "<". Spaces are ignored.

dyadic operator An operator that performs its operation with *two* expressions. It is placed between the two expressions. The following dyadic operators are available:

Dyadic Operator	Operation
+	REAL, COMPLEX or INTEGER addition
-	REAL, COMPLEX or INTEGER subtraction
*	REAL, COMPLEX or INTEGER multiplication
/	REAL or COMPLEX division[1]
^	REAL, COMPLEX or INTEGER exponentiation[1]
&	String concatenation
DIV	Gives the integer quotient of a division
MOD	Gives the remainder of a division
MODULO	Gives the remainder of a division, similar to MOD
=	Comparison for equality
<>	Comparison for inequality
<	Comparison for less than
>	Comparison for greater than
<=	Comparison for less than or equal to
>=	Comparison for greater than or equal to
AND	Logical AND
OR	Logical inclusive OR
EXOR	Logical exclusive OR

file name A name used to identify a file. The length and characters allowed in a file name vary according to the format of the volume on which the file resides.

- A file name on a Logical Interchange Format (LIF) volume consists of 1 to 10 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, and national language characters [CHR$(161) through CHR$(254)]. The first character in a LIF-compatible file name must be a letter. Spaces are ignored. (Note that some LIF implementations do not allow lowercase letters.)

[1] INTEGER arguments are converted to REAL before any computation is done.

- A file name on a Hierarchical File System (HFS) volume consists of 1 to 14 characters on short file name systems, and up to 255 characters on long file name systems (BASIC/UX only), which may include all ASCII characters except "/" and ":" and "<". Spaces are ignored.
- A file name on a Shared Resource Manager (SRM) volume consists of 1 to 16 characters, which may include all ASCII characters except "/" and ":" and "<". Spaces are ignored.

function A procedural call that returns a value. The call can be to a user-defined-function subprogram (such as FNInvert) or a machine-resident function (such as COS or EXP). The value returned by the function is used in place of the function call when evaluating the expression containing the function call.

graphic display unit This is 1/100 of the shortest axis on the plotting device. Graphic display units are the same size on both the X and Y axes. Abbreviated "GDU".

hard clip limits These are the physical limits of the plotting device.

hierarchy When a numeric or string expression contains more than one operation, the order of operations is determined by a precedence system. Operations with the highest precedence are performed first. Multiple operations with the same precedence are performed left to right. The following tables show the hierarchy for numeric and string operations.

Math Hierarchy

Precedence	Operator
Highest	Parentheses: (may be used to force any order of operations)
	Functions: user-defined and machine-resident
	Exponentiation: ^
	Multiplication and division: `*` `/` `MOD` `DIV` `MODULO`
	Addition, subtraction, monadic plus and minus: `+` `-`
	Relational operators: `=` `<>` `<` `>` `<=` `>=`
	`NOT`
	`AND`
Lowest	`OR` `EXOR`

String Hierarchy

Precedence	Operator
Highest	Parentheses
	Functions (user-defined and machine-resident) and substring operations
Lowest	Concatenation: **&**

I/O path A combination of firmware and hardware that can be used during the transfer of data to and from a BASIC program. Associated with an I/O path is a unique table that describes the I/O path. This association table uses 148 bytes and is referenced when an I/O path name is used. For further details, see the ASSIGN statement.

INTEGER A numeric data type stored internally in two bytes. Two's-complement representation is used, giving a range of −32 768 thru +32 767. If a numeric variable is not explicitly declared as an INTEGER, it is a REAL.

integer A number with no fractional part; a whole number.

interface select code A numeric expression that selects an interface for an I/O operation. Interface select codes 1 thru 7 are reserved for internal interfaces. Interface select codes 8 thru 31 are used for external interfaces. The internal HP-IB interface with select code 7 can be specified in statements that are restricted to external devices. (Also see "device selector".)

keyword A group of uppercase ASCII letters that has a predefined meaning to the computer. Keywords may be typed using all lowercase or all uppercase letters.

LIF This is the acronym for "Logical Interchange Format". This HP standard defines the format of mass storage files and directories. It allows the interchange of data between different machines. Series 200/300 files of type ASCII are LIF compatable. See "file name" for file name restrictions.

LIF protect code A non-listable, two-character code kept with a file description in the directory of a LIF volume. It guards against accidental changes to an individual file. It may be any two characters, but must not contain a ">" since that is used to terminate the protect code. Blanks are trimmed from protect codes. When the result contains more than two characters, only the first two are used as the actual protect code. A protect code that is the null string (or all blanks) is interpreted as no protect code.

literal A string constant. When quote marks are used to delimit a literal, those quote marks are not part of the literal. To include a quote mark in a literal, type two consecutive quote marks (except in response to a LINPUT statement). The drawings showing literal forms of specifiers (such as file specifiers) show the quote marks required to delimit the literal.

logical pen See "pen".

long file name systems (LFN) An HFS file system that allows individual file names to be up to 255 characters long (BASIC/UX only).

monadic operator An operator that performs its operation with one expression. It is placed in front of the expression. The following monadic operators are available:

Monadic Operator	Operation
-	Reverses the sign of an expression
+	Identity operator
NOT	Logical complement

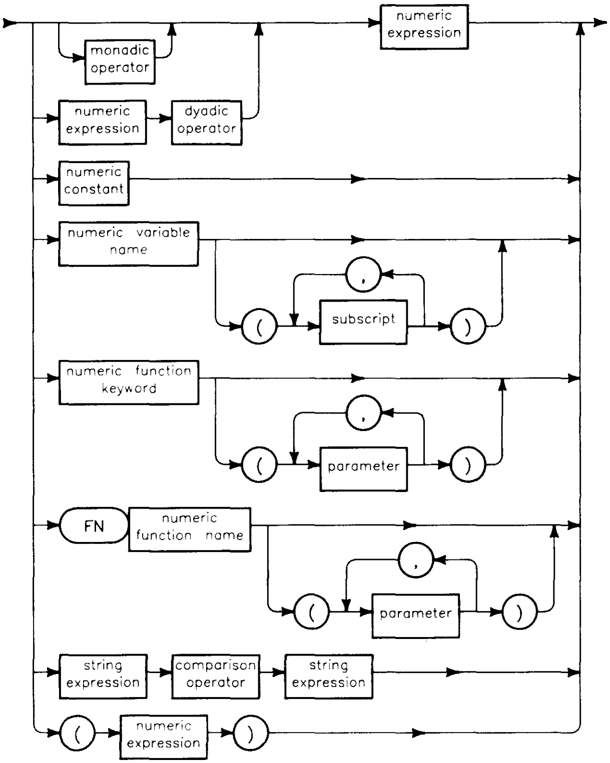
msus The acronym for "mass storage unit specifier". This archaic term is no longer used, because: it is not descriptive of newer mass storage devices which may have multiple *units* or multiple *volumes*; and it is not an industry-standard term. See the Glossary entry for **volume specifier**.

msvs The acronym for "mass storage volume specifier". See the Glossary entry for **volume specifier**.

name A name identifies one of the following: variable, line label, common block, I/O path, function, or subprogram. A name consists of one to fifteen characters. The first character must be an ASCII letter or one of the characters from CHR$(161) thru CHR$(254). The remaining characters, if any, can be ASCII letters, numerals, the underbar (_), or national language characters CHR$(161) thru CHR$(254). Names may be typed using any combination of uppercase and lowercase letters, unless the name uses the same letters as a keyword. Conflicts with keywords are resolved by mixing the letter case in the name. (Also see "file name", "directory name", and "volume name".)

node address An integer from 0 through 63 that identifies an SRM device (such as a workstation or controller).

numeric expression



Item	Description
monadic operator	An operator that performs its operation on the expression immediately to its right: `+ - NOT`
dyadic operator	An operator that performs its operation on the two expressions it is between: `^ * / MOD DIV + - = <> < > <= >= AND OR EXOR MODULO`
numeric constant	A numeric quantity whose value is expressed using numerals, decimal point, and optional exponent notation
numeric variable name	The name of a numeric variable or the name of a numeric array from which an element is extracted using subscripts
subscript	A numeric expression used to select an element of an array (see "array")
numeric function keyword	A keyword that invokes a machine-resident function which returns a numeric value
numeric function name	The name of a user-defined function that returns a numeric value
parameter	A numeric expression, string expression, or I/O path name that is passed to a function
comparison operator	An operator that returns a 1 (true) or a 0 (false) based on the result of a relational test of the operands it separates: `> < <= >= = <>`

password See "SRM password".

pen All graphical objects are "drawn" using mathematical representations in the computer's memory. This is done with the "logical pen". The logical pen creates five classes of objects: lines, polygons, labels, axes, and label locations (label locations are actually the position of an object, rather than an object).

Before these objects can be viewed, they are acted upon by various transformation matrices, such as scaling and pivoting. No single transformation affects all the objects, and no object is affected by all the transformations.

The output of the transformations is used to control the "physical pen". The physical pen creates the image that you actually see on the plotter or CRT. Since the graphics statements used to create objects act directly upon the logical pen, and you can see only the output of the physical pen, the location of the logical pen may not always be readily discernable from what you see.

The following table shows which transformations act upon which objects.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR	PDIR
Lines (generated by moves and draws)	X	X			*Note 4*
Polygons and rectangles	X	X			X
Characters (generated by LABEL)			X	X	
Axes (generated by AXES and 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.

permission A file-access permission on an HFS volume. See the PERMIT statement for details.

pipe A connection between programs that allows the output of one program as input to another. Thus you can chain programs together. Pipes are used in BASIC/UX (and HP-UX). For example,

```
PRINTER IS "|lp"
```

is the BASIC/UX command to pipe the output from CAT and LIST statements to the HP-UX command `lp`. which sends the CAT or LIST output to the printer.

primary address A numeric expession in the range of 0 thru 31 that specifies an individual device on an interface which is capable of servicing more than one device. The HP-IB interface can service multiple devices. (Also see "device selector".)

program line A statement that is preceded by a line number (and an optional line label) and stored with the ENTER, EXECUTE, or Return key into a program (see "statement").

protect code See "LIF protect code".

REAL A numeric data type that is stored internally in eight bytes using sign-and-magnitude binary representation. One bit is used for the number's sign, 11 bits for a biased exponent (bias = 1023), and 52 bits for a mantissa. On all values except 0, there is an implied "1." preceding the mantissa (this can be thought of as the 53rd bit). The range of REAL numbers is approximately:

−1.797 693 134 862 32 E+308 thru −2.225 073 858 507 2 E−308,
0, and
+2.225 073 858 507 2 E−308 thru +1.797 693 134 862 32 E+308.

If a numeric variable is not explicitly declared as INTEGER or COMPLEX, it is REAL.

record The records referred to in the Series 200/300 BASIC manuals are *defined* records. Defined records are the smallest unit of storage directly accessible on the mass storage media. The length of a record is different for various types of files. For ASCII files, the record length is the same as the sector size (256, 512, or 1024 bytes). For HP-UX files, defined records are always 1 byte long. For BDAT files, the defined record length is determined when a BDAT file is created by a CREATE BDAT statement. All records in a file are the same size.

There is another type of record called a "physical record" (or sector) which is the unit of storage handled by the mass storage device and the operating system. Physical records contain 256, 512, or 1024 bytes and are not accessible to the user via standard BASIC statements.

recursive See "recursive".

row-major order The order of accessing an array in which the right-most subscript varies the fastest.

secondary address A device-dependent command sent on HP-IB. It can be interpreted as a secondary address for the extended talker/listener functions or as part of a command sequence. (Also see "device selector".)

selector A numeric quantity used to identify or choose something from a number of possibilities. A selector is ususlly a numeric expression. For example: *device selector* is used to identify a device involved in a I/O operation, and *pen selector* is used to select a pen on a plotter.

short file name systems (SFN) An HFS file system that allows individual file names up to 14 characters long.

soft clip limits These are plotter clipping limits that are defined by the programmer. Lines drawn on a plotting device are drawn only inside the clipping limits.

specifier A string used to identify a method for handling an I/O operation. A specifier is usually a string expression. For example: *mass storage volume specifier* selects the proper drivers for a mass storage volume, and *plotter specifier* chooses the protocol of a plotting device.

SRM The acronym for Shared Resource Management.

SRM server The computer that controls access to the shared resources of the Shared Resource Management "file server" system.

SRM server's node address An integer in the range 0 through 63 that identifies the SRM server.

SRM interface The term used to describe the Resource Management Interface resident in an SRM *workstation* computer (not the interface in the SRM *server*).

SRM password A string of up to 16 characters that is used to protect a file on an SRM volume from being overwritten, purged, etc. It may be any 16 characters, but must not contain a ">" since that is used to terminate the password. Passwords are assigned by the PROTECT statement in BASIC or the Pascal Filer's Access command.

SRM volume name See "volume name".

SRM volume password See "volume password".

statement A keyword combined with any additional items that are allowed or required with that keyword. If a statement is placed after a line number and stored, it becomes a program line. If a statement is typed without a line number and executed, it is called a command.

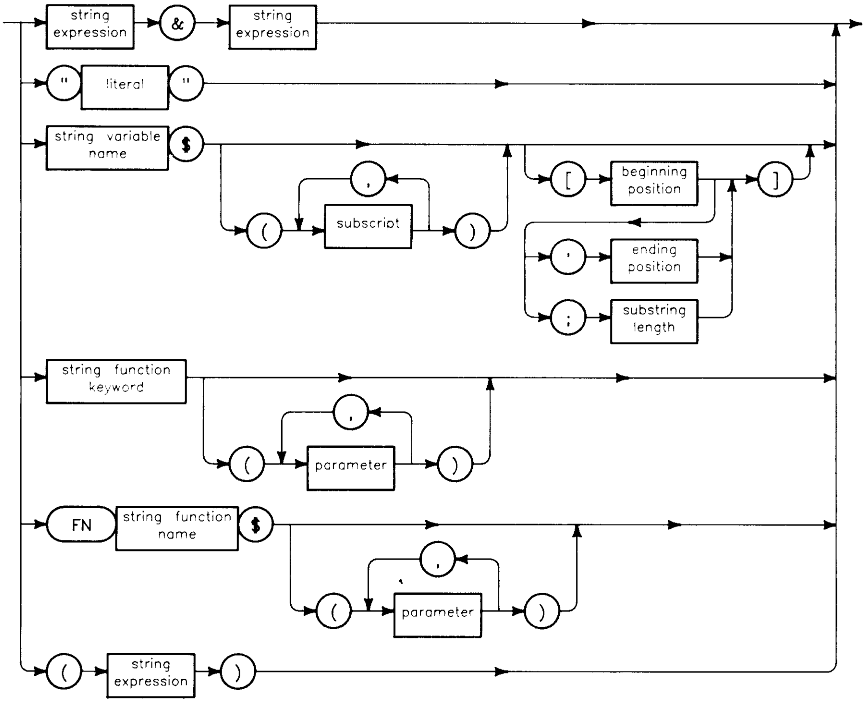
string A data type comprised of a contiguous series of characters. Strings require one byte of memory for each character of declared length, plus a two-byte length header. Characters are stored using an extended ASCII character set. The first character in a string is in position 1. The maximum length of a string is 32 767 characters. The current length of a string can never exceed the dimensioned length.

If a string is not explicitly dimensioned, it is implicitly dimensioned to 18 characters. Each element in an implicitly dimensioned string array is dimensioned to 18 characters.

When a string is empty, it has a current length of zero and is called a “null string”. All strings are null strings when they are declared. A null string can be represented as an empty literal (for example: `A$=""`) or as one of three special cases of substring. The substrings that represent the null string are:

1. Beginning position one greater than current length
2. Ending position one less than beginning position
3. Maximum substring length of zero

string expression



Item	Description
literal	A string constant composed of any characters available on the keyboard, including those generated with the ANY CHAR key.
string variable name	The name of a string variable or the name of a string array from which a string is extracted using subscripts.
subscript	A numeric expression used to select an element of an array (see "array").
beginning position	A numeric expression specifying the position of the first character in a substring (see "substring").
ending position	A numeric expression specifying the position of the last character in a substring (see "substring").
substring length	A numeric expression specifying the maximum number of characters to be included in a substring (see "substring").
string function keyword	A keyword that invokes a machine-resident function which returns a string value. String function keywords always end with a dollar sign.
string function name	The name of a user-defined function that returns a string value.
parameter	A numeric expression, string expression, or I/O path name that is is passed to a function.

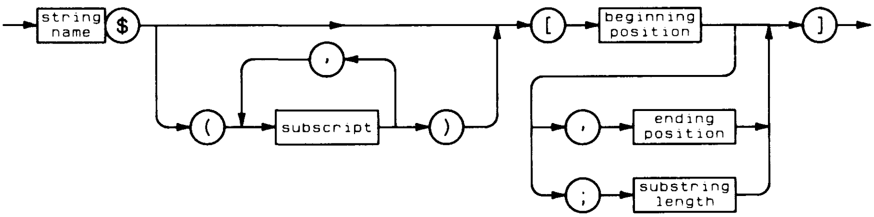
subprogram Can be a CSUB, a SUB subprogram or a user-defined-function subprogram (DEF FN). The first line in a SUB subprogram is a SUB statement. The last line in a SUB subprogram (except for comments) is a SUBEND statement. The first line in a function subprogram is a DEF FN statement. The last line in a function (except for comments) is an FNEND statement. Subprograms must follow the END statement of the main program.

SUB and CSUB subprograms are invoked by CALL. Function subprograms are invoked by an FN function occurring in an expression. A function subprogram returns a value that replaces the occurrence of the FN function when the expression is evaluated. Subprograms may alter the values of parameters passed by reference or variables in COM. It is recommended that you do not let function subprograms alter values in that way.

Invoking a subprogram establishes a new context. The new context remains in existence until the subprogram is properly exited or program execution is stopped. Subprograms can be recursive.

subroutine A program segment accessed by a GOSUB statement and ended with a RETURN statement.

substring



A substring is a contiguous series of characters that comprises all or part of a string. Substrings may be accessed by specifying a beginning position, or a beginning position and an ending position, or a beginning position and a maximum substring length.

The beginning position must be at least one and no greater than the current length plus one. When only the beginning position is specified, the substring includes all characters from that position to the current end of the string.

The ending position must be no less than the beginning position minus one and no greater than the dimensioned length of the string. When both beginning and ending positions are specified, the substring includes all characters from the beginning position to the ending position or current end of the string, whichever is less.

The maximum substring length must be at least zero and no greater than one plus the dimensioned length of the string minus the beginning position. When a beginning position and substring length are specified, the substring starts at the beginning position and includes the number of characters specified by the substring length. If there are not enough characters available, the substring includes only the characters from the beginning position to the current end of the string.

volume A named mass storage media, or portion thereof, which may contain several files. With BASIC, volumes are entities which are recognized by the disc controller. (This is in contrast to Workstation Pascal *logical* volumes, which are handled by the "Unitable" construct in the "TABLE" program; this program partitions a "hard" volume into several "logical" volumes by using byte offsets from sector number zero.)

volume name (or label) A name used to identify a mass storage volume. The volume name is assigned to the volume at initialization, but may be changed on LIF and HFS volumes with PRINT LABEL (and read with CAT and READ LABEL). With SRM volumes, you may only change it at the SRM console.

- LIF volume names consist of 1 to 6 characters which may be any ASCII character except "/", ":", ";", and "<".
- HFS volume names may contain 1 to 6 characters, which may be any ASCII character except "/" and ":" and "<". Spaces are ignored. (PRINT LABEL and READ LABEL are not supported for HFS disks under BASIC/UX.)
- SRM volume names may contain 1 to 16 characters, which may be any ASCII character except "/" and ":" and "<". Spaces are ignored.

volume password A "master" password on an SRM volume, assigned at initialization, that allows complete access to all files on that volume. SRM volume passwords consist of 1 to 16 characters. All ASCII characters except ">" are allowed. The volume password supercedes all access restrictions placed on files by the PROTECT statement in BASIC or the Pascal Filer's Access command.

volume specifier A string of information that identifies a mass storage volume. It consists of a device type (optional), device selector, unit number (optional; default=unit 0), and volume number (optional; default=volume number 0). Here are some examples:

```
:CS80, 700
:, 700
:,802, 0
:,1400,0,0
```

See MASS STORAGE IS for the complete syntax drawing.

Notes

Table of Contents

Appendix C: Interface Registers

Interface Registers C

This section lists the STATUS and CONTROL registers for I/O path names, interfaces, and pseudo select code 32.

I/O Path Registers

Registers for All I/O Paths

STATUS Register 0	0 = Invalid I/O path name 1 = I/O path name assigned to a device 2 = I/O path name assigned to a data file 3 = I/O path name assigned to a buffer 4 = I/O path name assigned to an HP-UX special file (See "Interface Registers" in the *Interfacing Techniques* manual.)

I/O Path Names Assigned to a Device

STATUS Register 1	Interface select code
STATUS Register 2	Number of devices
STATUS Register 3	Address of 1st device

If assigned to more than one device, the addresses of the other devices are available starting in STATUS Register 4.

I/O Path Registers (cont.)

I/O Path Names Assigned to an ASCII File

STATUS Register 1	File type = 3
STATUS Register 2	Device selector of mass storage device
STATUS Register 3	Number of records
STATUS Register 4	Bytes per record = 256
STATUS Register 5	Current record
STATUS Register 6	Current byte within record
STATUS Register 9	File I/O buffering in use (BASIC/UX only)

I/O Path Names Assigned to a BDAT File

STATUS Register 1	File type = 2
STATUS Register 2	Device selector of mass storage device
STATUS Register 3	Number of defined records
STATUS Register 4	Defined record length
STATUS Register 5	Current record
CONTROL Register 5	Set record
STATUS Register 6	Current byte within record
CONTROL Register 6	Set byte within record
STATUS Register 7	EOF record
CONTROL Register 7	Set EOF record
STATUS Register 8	Byte within EOF record
CONTROL Register 8	Set byte within EOF record

I/O Path Registers (cont.)

I/O Path Names Assigned to an HP-UX File

STATUS Register 1	File type = 4
STATUS Register 2	Device selector of mass storage device
STATUS Register 3	Number of defined records
STATUS Register 4	Defined record length (fixed record length = 1)
STATUS Register 5	Current record
CONTROL Register 5	Set record
STATUS Register 6	Current byte within record
CONTROL Register 6	Set byte within record
STATUS Register 7	EOF record
CONTROL Register 7	Set EOF record
STATUS Register 8	Byte within EOF record
CONTROL Register 8	Set byte within EOF record
STATUS Register 9	File I/O buffering in use (BASIC/UX only)
CONTROL Register 9	Set file I/O buffering (BASIC/UX only)

I/O Path Names Assigned to a Buffer

When the status of register 0 indicates a buffer (3), the status and control registers have the following meanings.

STATUS Register 1	Buffer type (1=named, 2=unnamed)
STATUS Register 2	Buffer size in bytes
STATUS Register 3	Current fill pointer
CONTROL Register 3	Set fill pointer
STATUS Register 4	Current number of bytes in buffer
CONTROL Register 4	Set number of bytes

I/O Path Registers (cont.)

STATUS Register 5	Current empty pointer
CONTROL Register 5	Set empty pointer
STATUS Register 6	Interface select code of inbound TRANSFER
STATUS Register 7	Interface select code of outbound TRANSFER
STATUS Register 8	If non-zero, inbound TRANSFER is continuous
CONTROL Register 8	Cancel continuous mode inbound TRANSFER if zero
STATUS Register 9	If non-zero, outbound TRANSFER is continuous
CONTROL Register 9	Cancel continuous mode outbound TRANSFER if zero
STATUS Register 10	Termination status for inbound TRANSFER

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANS-FER Active	TRANS-FER Aborted	TRANS-FER Error	Device Termination	Byte Count	Record Count	Match Character
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

STATUS Register 11 Termination status for outbound TRANSFER

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANS-FER Active	TRANS-FER Aborted	TRANS-FER Error	Device Termination	Byte Count	Record Count	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=0

STATUS Register 12	Total number of bytes transferred by last inbound TRANSFER
STATUS Register 13	Total number of bytes transferred by last outbound TRANSFER

Summary of CRT STATUS and CONTROL Registers

STATUS Register 0	Current print position (column)
CONTROL Register 0	Set print position (column). See also TAB and TABXY.
STATUS Register 1	Current print position (line)
CONTROL Register 1	Set print position (line). See also TABXY.
STATUS Register 2	Insert-character mode
CONTROL Register 2	Set insert character mode if non-0
STATUS Register 3	Number of lines “above screen”.
CONTROL Register 3	Undefined
STATUS Register 4	Display functions mode
CONTROL Register 4	Set display functions mode if non-0. To perform the same function, use the statement DISPLAY FUNCTIONS ON/OFF.

CRT Registers (cont.)

STATUS Register 5 Returns the CRT alpha color value set (or default). This does not reflect changes due to printing `CHR$(`*x*`)`, where 136≤*x*≤143.

CONTROL Register 5 Set default alpha color:
For Alpha Displays:

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 Displays:
Values 0 thru 255 which correspond to the graphics pens. These values are treated as MOD 2ˆ*n* where *n* is the number of display planes.

`CONTROL CRT,5;`*n* sets the values of the CRT registers 15, 16, and 17, but the converse is not true. That is, `STATUS CRT,5` may not accurately reflect the CRT state if `CONTROL 15, 16,` and/or `17` have been executed. Note that to perform the same function as `CONTROL CRT,5;`*n*, you can use the ALPHA PEN statement.

CRT Registers (cont.)

STATUS Register 6	ALPHA ON flag[1]
CONTROL Register 6	Undefined[1]
STATUS Register 7	GRAPHICS ON flag[1]
CONTROL Register 7	Undefined[1]
STATUS Register 8	Display line position[1] (column)
CONTROL Register 8	Set display line position[1] (column). See also TAB.
STATUS Register 9	Screenwidth (number of characters). Also available in the SYSTEM$("CRT ID") function result.
CONTROL Register 9	Undefined
STATUS Register 10	Cursor-enable flag[1]
CONTROL Register 10	Cursor-enable:[1] 0=invisible cursor. non-0=cursor visible.
STATUS Register 11	CRT character mapping flag
CONTROL Register 11	Disable CRT character mapping (if non-0)
STATUS Register 12	Key labels display made.[1]
CONTROL Register 12	Set key labels display mode:[1] 0 = typing-aid key labels displayed unless program is running. 1 = key labels always off (or use KEY LABELS OFF). 2 = key labels displayed at all times (or use KEY LABELS ON).
STATUS Register 13	CRT height (number of lines to be used for alpha display).
CONTROL Register 13	Set CRT height (must be >= 9). Alternately use the ALPHA HEIGHT statement.

[1] Error 713 is given if a window number is specified instead of a select code (BASIC/UX only).

CRT Registers (cont.)

STATUS Register 14 Display replacement rule currently in effect.[1]

CONTROL Register 14 Set display replacement rule[1]
(with bit-mapped alpha displays only)
0—0
1—source AND old
2—source AND NOT old
3—source;default
4—NOT source AND old
5—old
6—source EXOR old
7—source OR old
8—source NOR old
9—source EXNOR old
10—NOT old 11—source OR NOT old
12—NOT source
13—NOT source OR old
14—source NAND old
15—1

It is strongly recommended that you do not change the default display replacement rule.

STATUS Register 15 Return the value set (or the default) for the color in the `PRINT/DISP` area. This does not reflect changes due to printing `CHR$(`x`)`, where $136 \le x \le 143$.

CONTROL Register 15 Set `PRINT/DISP` color (or use the PRINT PEN statement). Similar to CRT control register 5 but specific to CRT `PRINT/DISP` areas; that is, it does not affect the areas covered by CRT registers 16 and 17.

[1] For BASIC/UX information in this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

CRT Registers (cont.)

STATUS Register 16 Return the value set (or the default) for the softkey label color.[1]

CONTROL Register 16 Set key labels color (or use the KEY LABELS PEN statement).[1] Similar to CRT control register 5 but only affects the softkey labels. Does not affect the areas covered by CRT registers 15 and 17. When running BASIC/UX in X Windows, this CONTROL register affects the Keyboard Line area and Message Line area of the display.

STATUS Register 17 Return the value set (or the default) for the color of the "non-enhance" area. This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen.

CONTROL Register 17 Set "non-enhance" color (or use the KBD LINE PEN statement). This includes the keyboard entry line, runlight, system message line, annunciators, and edit screen. Similar to CRT control register 5 but does not affect the areas covered by CRT control registers 15 and 16.

STATUS Register 18 Read the alpha write-enable mask.

CONTROL Register 18 Set alpha write-enable mask to a bit pattern (or use the SET ALPHA MASK statement). When running BASIC/UX in the X Window environment, this CONTROL register is **not** supported.

STATUS Register 19 Returns the maximum value for ALPHA MASK argument.

CONTROL Register 19 Undefined.

STATUS Register 20 Read the alpha display-enable mask.[1]

CONTROL Register 20 Set alpha display-enable mask to a bit pattern (or use the SET DISPLAY MASK statement).[1]

[1] Error 713 is given if a window number is specified instead of a select code (BASIC/UX only).

CRT Registers (cont.)

STATUS Register 21 Return compatibility mode (0 or 1).

CONTROL Register 21 Switch between the CRT compatibility mode (value≠0) and the native bit-mapped mode (value=0). That is, switch both alpha and graphics to non-bit-mapped display (if value≠0) or bit-mapped display (if value=0). It effectively initializes the alpha display and executes a `GINIT` and a `PLOTTER IS CRT,"INTERNAL"`.

STATUS Register 22 Undefined (BASIC/UX only).

CONTROL Register 22 Raises a window to the top of the window stack if non-zero; pushes a window to the bottom of the stack if zero (BASIC/UX only).

STATUS Register 23 Returns terminal compatibility mode (BASIC/UX only).

CONTROL Register 23 Sets terminal compatibility mode (BASIC/UX only).

[1] Error 713 is given if a window number is specified instead of a select code (BASIC/UX only).

Summary of Keyboard Status and Control Registers

STATUS Register 0	CAPS LOCK flag
CONTROL Register 0	Set CAPS LOCK if non-0
STATUS Register 1	PRINTALL flag
CONTROL Register 1	Set PRINTALL if non-0
STATUS Register 2	Function key menu.
CONTROL Register 2	Function key menu: 0 = System menu (or SYSTEM KEYS statement) 1-3 = User menu 1 thru 3 (or USER *n* KEYS statement along with the appropriate menu number)
STATUS Register 3	Undefined
CONTROL Register 3	Set auto-repeat interval. If 1 thru 255, repeat interval in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at power-on or SCRATCH A is 80ms.)[1]
STATUS Register 4	Undefined
CONTROL Register 4	Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at power-on or SCRATCH A is 700ms.)[1]
STATUS Register 5	KBD$ buffer overflow register. 1 = overflow Register is reset when read.
CONTROL Register 5	Undefined
STATUS Register 6	Typing aid expansion overflow register. 1 = overflow. Register is reset when read.
CONTROL Register 6	Undefined

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Keyboard Registers (cont.)

STATUS Register 7 Interrupt Status

Most Significant Bit **Least Significant Bit**

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	0	0	INITIALIZE Timeout Interrupt Disabled	Reserved For Future Use	Reserved For Future Use	RESET Key Interrupt Disabled	Keyboard and and Knob Interrupt Disabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 7 Interrupt Disable Mask

Most Significant Bit **Least Significant Bit**

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Not Used			INITIALIZE Timeout	Reserved For Future Use	Reserved For Future Use	RESET Key	Keyboard and Knob
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Keyboard Registers (cont.)

STATUS Register 8 Keyboard Language Jumper

0-US ASCII
1-French
2-German
3-Swedish
4-Spanish
5-Katakana
6-Canadian English
7-United Kingdom
8-Canadian French
9-Swiss French
10-Italian
11-Belgian
12-Dutch
13-Swiss German
14-Latin(Spanish)
15-Danish
16-Finnish
17-Norwegian
18-Swiss French*
19-Swiss German*

See also SYSTEM$("KEYBOARD LANGUAGE") which requires the LEX binary. Note that the STATUS statement when used with this register does not require the LEX binary.

CONTROL Register 8 Undefined

STATUS Register 9 Keyboard Type[1]

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Internal Use	Internal Use	1=HIL Keyboard Interface 0=non-HIL	1=No Keyboard 0=Key-board Present	1=n-Key Rollover 0=2 or less rollover	0	1=98203C Keyboard 0=Other Keyboard	1=98203A Keyboard 0=Other Keyboard
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Keyboard Registers (cont.)

Bits 5, 1, and 0 of STATUS Register 9 and the following table can be used to determine the Keyboard Type.

Bit 5	Bit 1	Bit 0	Keyboard Type
0	0	0	HP 98203B or built-in
0	0	1	HP 98203A
1	0	0	ITF (such as the HP 46020A and 46021A)
1	1	0	HP 98203C

CONTROL Register 9 Undefined

STATUS Register 10 Status at Last Knob Interrupt

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	CTRL Key Pressed	SHIFT Key Pressed
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Note that bit 1 is *always* 0 for keyboards connected to an HP-HIL interface, and with all HP-HIL mice and knobs (e.g. HP 46083A Rotary Control Knob, HP 46085 Control Dials, and HP 98203C Keyboard Knob).[1]

CONTROL Register 10 Undefined

STATUS Register 11 0=horizontal-pulse mode; 1=all-pulse mode.

CONTROL Register 11 Set knob pulse mode (0 is default). See the knob discussion in the "Porting to 3.0" chapter of *BASIC Programming Techniques.*[1]

STATUS Register 12 "Pseudo-EOI for CTRL-E " flag

CONTROL Register 12 Enable pseudo-EOI for CTRL-E if non-0

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Keyboard Registers (cont.)

STATUS Register 13 Katakana flag

CONTROL Register 13 Set Katakana if non-0

STATUS Register 14 Numbering of softkeys on ITF keyboard:
0 ⇒ f1 is key number 1 (default);
1 ⇒ f1 is key number 0;

CONTROL Register 14 Softkey numbering on ITF keyboard (see above register description).

STATUS Register 15 Currently in 98203 keyboard compatibility mode:
0⇒OFF (default)
1⇒ON

CONTROL Register 15 Turns "98203 keyboard compatibility mode" on ($\neq 0$) and off ($=0$)[1]. (See the chapter "Porting to Series 300" in the *Programming Techniques* manual for further information about using this mode.) Note that instead of using the CONTROL register 15 statement you can use the KBD CMODE statement to turn the "98203 keyboard compatibility mode" ON and OFF.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

STATUS Register 16 Returns the enabled/disabled status of the up and down arrow keys, [Prev], [Next], and [↖] (both shifted and un-shifted for all of these keys). If the status value is 1 it means these keys are deactivated. Note that the default value is 0.

CONTROL Register 16 Allows you to disable or re-enable the display scrolling keys mentioned for STATUS Register 16. This prevents accidental scrolling of the display screen. Executing a 1 with the CONTROL statement deactivates the print scrolling keys and a 0 activates them.

STATUS Register 17 Automatic menu switching:

1 ⇒ enabled (default)
0 ⇒ disabled

CONTROL Register 17 Automatic menu switching:

<>0 ⇒ enable
0 ⇒ disable

This register controls whether a system with an ITF keyboard will switch to (from) the User 2 Menu automatically on entering (leaving) EDIT mode.

Summary of HP-IB Status and Control Registers

Status Register 0 Card identification = 1

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 1 **Serial Poll Response Byte**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	SRQ 1=I did it 0=I didn't	Device Dependent Status					
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

HP-IB Registers (cont.)

Status Register 2

Busy Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Reserved For Future Use	Handshake In Progress	Interrupts Enabled	TRANSFER In Progress
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 2

Parallel Poll Response Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1=True	DIO7 1=True	DIO6 1=True	DIO5 1=True	DIO4 1=True	DIO3 1=True	DIO2 1=True	DIO1 1=True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Status Register 3

Controller Status and Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of HP-IB Interface				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 3

Set My Address

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

HP-IB Registers (cont.)

Status Register 4 **Interrupt Status**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value= −32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 4

Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

HP-IB Registers (cont.)

Status Register 5 **Interrupt Enable Mask**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value= −32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Comman	Secondary Command While Addresse	Clear Received	Unrecognized Addresse Comman	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Control Register 5 **Parallel Poll Response Mask**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Unconfigure	Logic Sense	Data Bit Used for Response		
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

HP-IB Registers (cont.)

Status Register 6 **Interface Status**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value= −32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

* Least-significant bit of last address recognized

Status Register 7 **Bus Control and Data Lines**

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC[1] True	NRFD[1] True	EOI True	SRQ[2] True	IFC True	REN True
Value= −32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

[1] Only if currently Addressed to Talk, else not valid.

[2] Only if currently Active Controller, else not valid.

HP-IB Registers (cont.)

Interrupt Enable Register (ENABLE INTR)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value= −32 768	Value= 16 384	Value= 8 192	Value= 4 096	Value= 2 048	Value= 1 024	Value= 512	Value= 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshak Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Summary of RS-232C Serial STATUS and CONTROL Registers

General Notes: Most Control registers accept values in the range of zero through 255. Some registers accept only specified values as indicated, or higher values for baud rate settings. Values less than zero are not accepted. Higher-order bits not needed by the interface are discarded if the specified value exceeds the valid range.

Reset value is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

See the *Interfacing Techniques* manual for "Modifications to RS232 and Datacomm Registers".

STATUS Register 0 Card Identification

Value returned: 2 indicates a 98626 (if 130 is returned, the Remote jumper wire has been removed from the interface card); 66 indicates a 98644 (194 if the Remote jumper has been removed).

CONTROL Register 0 Interface Reset

Any value from 1 thru 255 resets the card. Execution is immediate; any data transfers in process are aborted and any buffered data is destroyed. A value of 0 causes no action.

STATUS Register 1 Interrupt Status

Bit 7 set: Interface hardware interrupt to CPU enabled.

Bit 6 set: Card is requesting interrupt service.

Bits 5&4:

00	Interrupt Level 3
01	Interrupt Level 4
10	Interrupt Level 5
11	Interrupt Level 6

Bits 3 thru 0 not used.

RS-232C Serial Registers (cont.)

CONTROL Register 1 Transmit BREAK[1]

Any non-zero causes a BREAK to be sent.

STATUS Register 2 Interface Activity Status

Bit 7 thru 4 are not used.
Handshake ended with an escape.

Bit 3 set: Error condition.
Handshake ended with an escape.

Bit 2 set: Handshake in progress. This occurs only during multi-line function calls.

Bit 1 set: Firmware interrupts enabled (ENABLE INTR active for this select code).[1]

Bit 0: TRANSFER in Progress.

STATUS Register 3 Current Baud Rate

Returns one of the values listed under CONTROL Register 3.

CONTROL Register 3 Set New Baud Rate

Use any one of the following values:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134.5 (or 134)	600	3600	19200

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

RS-232C Serial Registers (cont.)

STATUS Register 4 Current Character Format

See CONTROL Register 4 for function of individual bits.

CONTROL Register 4 Set New Character Format[1]

Table 13-8. Character Format and Parity Settings

Parity Sense[2] (Switches 5&4)	Parity Enable (Switch 3)	Stop Bits (Switch 2)	Character Length (Switches 1&0)
00 ODD parity 01 EVEN parity 10 Always ONE 11 Always ZERO	0 Disabled 1 Enabled	0 1 stop bit 1 1.5 stop bits (if 5 bits/char), or 2 stop bits (if 6, 7, or 8 bits/char).	00 5 bits/char 01 6 bits/char 10 7 bits/char 11 8 bits/char

Bits 7 and 6 are reserved for future use.[1]

STATUS Register 5 Current Status of Modem Control Lines

Returns CURRENT line state values. See CONTROL Register 5 for function of each bit.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

[2] Parity sense valid only if parity is enabled (bit 3=1). If parity is disabled, parity sense is meaningless.

RS-232C Serial Registers (cont.)

CONTROL Register 5 Set Modem Control Line States

Sets Modem Control lines or interface state as follows:

Bit 4 set: Enables loopback mode for diagnostic tests.[1]

Bit 3 set: Set Secondary Request-to-Send modem line to[1] active state.

Bit 2 set: Set Data Rate Select modem line to active state.

Bit 1 set: Force Request-to-Send modem line to fixed active state.

Bit 1 clear: Toggle RTS line as in normal OUTPUT operations.

Bit 0 set: Force Data Terminal Ready modem line to fixed active state.

Bit 0 clear: Toggle DTR line as in normal OUTPUT and ENTER operations.

STATUS Register 6 Data In (not supported on BASIC/UX)

Reads character from input buffer. Buffer contents is not destroyed, but bit 0 of STATUS Register 10 is cleared.

CONTROL Register 6 Data Out (not supported on BASIC/UX)

Sends character to transmitter holding register. This register is sometimes used to transmit protocol control characters or other characters without using OUTPUT statements. Modem control lines are not affected.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

RS-232C Serial Registers (cont.)

STATUS Register 7 Optional Receiver/Driver Status (not supported on BASIC/UX)

Returns current value of optional circuit drivers or receivers as follows:

Bit 3: Optional Circuit Driver 3 (OCD3).

Bit 2: Optional Circuit Driver 4 (OCD4).

Bit 1: Optional Circuit Receiver 2 (OCR2).

Bit 0: Optional Circuit Receiver 3 (OCR3).

Other bits are not used (always 0).

CONTROL Register 7 Set New Optional Driver States (not supported on BASIC/UX)

Sets (bit=1) or clears (bit=0) optional circuit drivers as follows:

Bit 3: Optional Circuit Driver 3 (OCD3),

Bit 2: Optional Circuit Driver 4 (OCD4).

Other bits are not used.

STATUS Register 8 Current Interrupt Enable Mask (not supported on BASIC/UX)

Returns value of interrupt mask associated with most recent ENABLE INTR statement. Bit functions are as follows:

Bit 3: Enable interrupt on modem line change. STATUS Register 11 shows which modem line has changed.

Bit 2: Enable interrupt on UART status error. This bit is used to trap ERROR 167 caused by UART error conditions. STATUS Register 10, bits 4 thru 1, show cause of error.

Bit 1: Enable interrupt when Transmitter Holding Register is empty.

Bit 0: Enable interrupt when Receiver Buffer is full.

RS-232C Serial Registers (cont.)

STATUS Register 9

Cause of Current Interrupt (not supported on BASIC/UX)

Returns cause of interrupt as follows:

Bits 2&1: Return cause of interrupt

11=UART error (BREAK, parity, framing, or overrun error). See STATUS Register 10.

10=Receiver Buffer full. Cleared by STATUS to Register 6.

01=Transmitter Holding Register empty. Cleared by CONTROL Register 6 or STATUS to Register 9.

00=Interrupt caused by change in modem status line(s). See STATUS Register 11.

Bit 0: Set when no active interrupt requests from UART are pending. Clear until all pending interrupts have been serviced.

STATUS Register 10

UART Status (not supported on BASIC/UX)

Bit set indicates UART status or detected error as follows:

Bit 7: Not used.

Bit 6: Transmit Shift Register empty.

Bit 5: Transmit Holding Register empty.

Bit 4: Break received.

Bit 3: Framing error detected.

Bit 2: Parity error detected.

Bit 1: Receive Buffer Overrun error.

Bit 0: Receiver Buffer full.

RS-232C Serial Registers (cont.)

STATUS Register 11

Modem Status (not supported on BASIC/UX)

Bit set indicates that the specified modem line or condition is active.

Bit 7: Data Carrier Detect (DCD) modem line active.

Bit 6: Ring Indicator (RI) modem line active.

Bit 5: Data Set Ready (DSR) modem line active.

Bit 4: Clear-to-Send (CTS) modem line active.

Bit 3: Change in DCD line state detected.

Bit 2: RI modem line changed from true to false.

Bit 1: Change in DSR line state detected.

Bit 0: Change in CTS line state detected.

RS-232C Serial Registers (cont.)

STATUS Register 12 (not supported on BASIC/UX) Modem Handshake Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable[1]	0	Data Set Ready Disable[2]	Clear to Send Disable[3]	0	0	0	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 12 (not supported on BASIC/UX) Modem Handshake Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable[1]	Not Used	Data Set Ready Disable[2]	Clear to Send Disable[3]	Not Used			
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Interrupt Enable Register (ENABLE INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

[1] 0 = Wait for Carrier Detect on Enter Operations; 1 = Don't wait.

[2] 0 = Wait for Data Set Ready on Enter and Output Operations; 1 = Don't wait.

[3] 0 = Wait for Clear to Send on Output Operations; 1 = Don't wait.

RS-232C Serial Registers (cont.)

STATUS Register 13 Read 98644 “SCRATCH A default” baud rate[1]

Returns the baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 3).

CONTROL Register 13 Set 98644 “SCRATCH A default” baud rate[1]

Sets both the “current” and the “default” baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 3). Default value in this register is 9600 baud.

STATUS Register 14 Read 98644 “SCRATCH A default” character format[1]

Returns the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 4).

CONTROL Register 14 Set 98644 “SCRATCH A default” character format[1]

Sets the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 4). Default value in this register specifies a character format of 8 bits/character, 1 stop bit, and parity disabled.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Overview of Datacomm Status and Control Registers

Unless indicated otherwise, the Status Register returns the current value for a given parameter; the Control Register sets a new value.

See the *Interfacing Techniques* manual for changes to RS232 and Datacomm Registers.

Register	Function
0	Control: Interface Reset; Status: Interface Card ID
1 (Status only)	Hardware Interrupt Status: 1=Enabled, 0=Disabled (not supported on BASIC/UX)
2 (Status only)	Datacomm activity: 0=inactive, 1=ENTER in process, 2=OUTPUT in process[1]
3	Select Protocol: 1= Async, 2= Data Link[1]
4 (Status only)	Cause of ON INTR program branch (not supported on BASIC/UX)
5	Control: Terminate transmission; Status: Inbound queue status (not supported on BASIC/UX)
6	Control: Send BREAK to remote; Status: 1=BREAK pending (not supported on BASIC/UX)
7 (,Status only)	Current modem receiver line states
8	Modem driver line states
9 (Status only)	Control block TYPE (not supported on BASIC/UX)
10 (Status only)	Control block MODE (not supported on BASIC/UX)
11 (Status only)	Available outbound queue space (not supported on BASIC/UX)
12	Control: Connect/Disconnect line; Status: Line connection status (not supported on BASIC/UX)
13	ON INTR mask (not supported on BASIC/UX)
14	Control Block mask (not supported on BASIC/UX)
15	Modem Line interrupt mask (not supported on BASIC/UX)
16	Connection timeout limit
17	No Activity timeout limit
18	Lost Carrier timeout limit
19	Transmit timeout limit
20	Async: Transmit baud rate (line speed) Data Link: Set Transmit/Receive baud rate (line speed)

[1] For BASIC/UX information on this register see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Overview of Datacomm Registers (cont.)

Register	Function
21	Async: Incoming (receiver) baud rate (line speed) (not supported on BASIC/UX) Data Link: GID address (0 thru 26 corresponds to "@" thru "Z")
22	Async: Protocol handshake type (not supported on BASIC/UX) Data Link: DID address (0 thru 26 corresponds to "@" thru "Z")
23	Hardware handshake type: ON/OFF, HALF/FULL duplex, (not supported on BASIC/UX) Modem/Non-modem
24	Async: Control Character mask (not supported on BASIC/UX) Data Link: Block Size limit
25 (Status only)	Number of received errors since last interface reset (not supported on BASIC/UX)
26	Async: First protocol character (ACK/DC1) (not supported on BASIC/UX) Data Link: NAKs received since last interface reset

For the BASIC Workstation, registers 27-35, 37, and 39 are used with Async protocol only. They are not accessible during Data Link operations. Note that registers 27-33 and 37-39 are not supported on BASIC/UX and that BASIC/UX does not support Data Link operations.

Register	Function
27	Second protocol handshake character (ENQ/DC3)
28	Number of characters in End-of-line sequence
29	First character in EOL sequence
30	Second character in EOL sequence
31	Number of characters in PROMPT sequence
32	First character in PROMPT sequence
33	Second character in PROMPT sequence
34	Data bits per character excluding start, stop and parity
35	Stop bits per character (0=1, 1=1.5, and 2=2 stop bits)
36	Parity sense: 0=NONE, 1=ODD, 2= EVEN, 3=ZERO, 4=ONE Data Link: 0=NONE (HP 1000 host), 1=ODD (HP 3000 host)
37	Inter-character time gap in character times (Async only)
38 (Status only)	Transmit queue status (1=empty)
39	BREAK time in character times (Async only)

Summary of Datacomm Interface Status and Control Registers

General Notes: Control registers accept values in the range of zero through 255. Some registers require specified values, as indicated. Illegal values or values less than zero or greater than 255, cause ERROR 327.

Reset value, shown for various Control Registers, is the default value used by the interface after a reset or power-up until the value is overridden by a CONTROL statement.

Status 0 Card Identification
Value returned: 52 (if 180 is returned, check select code switch cluster and make sure switch R is ON).

Control 0 Card Reset
Any value, 1 thru 255, resets the card. Immediate execution. Data in queues is destroyed.

Status 1 Hardware Interrupt Status (not used in most applications) 1 = Enabled 0 = Disabled[1]

Status 2 Datacomm Activity[1]
0 = No activity pending on this select code.
Bit 0 set: ENTER in process.
Bit 1 set: OUTPUT in process.
(Non-zero ONLY during multi-line function calls.)

Status 3 Current Protocol Identification:[1]
1 = Async, 2 = Data Link Protocol

Control 3 Protocol to be used after next card reset (`CONTROL Sc,0;1`)[1]
1 = Async Protocol 2 = Data Link Protocol
This register overrides default switch configuration.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Datacomm Registers (cont.)

Status 4 Cause of ON INTR program branch (not supported on BASIC/UX).

Bit	Function: Async Protocol	Function: Data Link Protocol
0	Data and/or Control Block available	Data Block Available
1	Prompt received	Space available for a new transmission block
2	Framing and/or parity error	Receive or transmit error
3	Modem line change	Modem line change
4	No Activity timeout (forces a disconnect)	No Activity timeout (forces a disconnect)
5	Lost carrier or connection timeout (forces a disconnect)	Lost carrier or connection timeout (forces a disconnect)
6	End-of-line received	Not Used
7	Break received	Not used

Contents of this register are cleared when a STATUS statement is executed to it.

Status 5 Inbound queue status (not supported on BASIC/UX)

Value	Interpretation
0	Queue is empty
1	Queue contains data but no control blocks
2	Queue contains one or more control blocks but no data
3	Queue contains both data and one or more control blocks

Datacomm Registers (cont.)

Control 5 Terminate Transmission (not supported on BASIC/UX)
`OUTPUT S,5;0` is equivalent to `OUTPUT S;END`

Data Link: Sends previous data as a single block with an ETX terminator, then idles the line with an EOT.

Async: Tells card to turn half-duplex line around. Does nothing when line is full-duplex. The next data OUTPUT automatically regains control of the line by raising the RTS (request-to-send) modem line.

Status 6 Break status: 1 = BREAK transmission pending, 0 = no BREAK pending. (not supported on BASIC/UX)

Control 6 Send Break; causes a Break to be sent as follows:[1]

Data Link Protocol: Send Reverse Interrupt (RVI) reply to inbound block, or CN character instead of data in next outbound block.

Async Protocol: Transmit Break. Length is defined by Control Register 39.

Note that the value sent to the register is arbitrary.

Status 7 Modem receiver line states (values shown are for male cable connecter option for connection to modems).

Bit 0: Data Mode (Data Set Ready) line
Bit 1: Receive ready (Data Carrier Detect line)
Bit 2: Clear-to-send (CTS) line
Bit 3: Incoming call (Ring Indicator line)[1]
Bit 4: Depends on cable option or adapter used[1]

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Datacomm Registers (cont.)

Status 8 Returns modem driver line states.

Control 8 Sets modem driver line states (values shown are for male cable connector option for connection to modems).

Bit 0: Request-to-send (RS or RTS) line	1 = line set (active)
Bit 1: Data Terminal Ready (DTR) line	0 = line clear (inactive)

Bit 2: Driver 1: Data Rate Select
Bit 3: Driver 2: Depends on cable option[1] or adapter used
Bit 4: Driver 3: Depends on cable option or adapter used
Bit 5: Driver 4: Depends on cable option or adapter used
Bit 6,7: Not used

Reset value=0 prior to connect. Post-connect value is handshake dependent. Note that RTS line cannot be altered (except by OUTPUT or OUTPUT...END) for half-duplex modem connections.

Status 9 Returns control block TYPE if last ENTER terminated on a control block. See Status Register 10 for values (not supported on BASIC/UX).

Status 10 Returns control block MODE if last ENTER terminated on a control block (not supported on BASIC/UX).

Async Protocol Control Blocks

Type	Mode	Interpretation
250	1	Break received (Channel A)
251	1[2]	Framing error in the following character
251	2[2]	Parity error in the following character
251	3[2]	Parity and framing errors in the following character
252	1	End-of-line terminator detected
253	1	Prompt received from remote
0	0	No Control Block encountered

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

[2] Parity/framing error control blocks are not generated when characters with parity and/or framing errors are replaced by an underscore (_) character.

Datacomm Registers (cont.)

Data Link Protocol Control Blocks

Type	Mode	Interpretation
254	1	Preceding block terminated by ETB character
254	2	Preceding block terminated by ETX character
253[1]	···	(see following table for Mode interpretation)
0	0	No Control Block encountered

Mode Bit(s)	Interpretation
0	1 = Transparent data in following block 0 = Normal data in following block
2,1	00 = Device select 01 = Group select 10 = Line select
3	1 = Command channel 2 = Data channel

Status 11 Returns available outbound queue space (in bytes), provided there is sufficient space for at least three control blocks. If not, value is zero (not supported on BASIC/UX).

Status 12 Datacomm Line connection status (not supported on BASIC/UX)

Value	Interpretation
0	Disconnected
1	Attempting Connection
2	Dialing
3	Connected[2]
4	Suspended
5	Currently receiving data (Data Link only)
6	Currently transmitting data (Data Link only)

[1] This type is used primarily in specialized applications.

[2] When using Data Link: Connected - datacomm idle

Datacomm Registers (cont.)

NOTE

When the datacomm line is suspended, CLEAR, ABORT, or RESET must be executed before the line can be reconnected.

Reset value — 0 if $\overline{R}$ on interface select code switch cluster is ON (1).

Control 12 Connects, initiates auto-dial sequence, and disconnects interface from datacomm line (not supported on BASIC/UX).

Value	Interpretation
0	Disconnected from datacomm line
1	Connected to datacomm line (set DTR & RTS)
2	Start auto dial. (Followed by OUTPUT of telephone numbers)

Status 13 Returns current ON INTR mask (not supported on BASIC/UX)

Control 13 Sets ON INTR mask[1] (not supported on BASIC/UX)

Data Link Protocol

Bit	Value	Enables interrupt when:
0	1	A full block is available in receive queue
1	2	Transmit queue is empty
2	4	Receive or transmit error detected
3	8	A modem line changed
4	16[2]	No Activity timeout forced a disconnection
5	32[2]	Lost Carrier or Connection timeout caused a disconnection

[1] If a CONTROL statement is used to access this register, the control block is placed in the outbound queue. If the ENABLE INTR... statement is used with a mask, the mask value is placed directly in the control register, bypassing any queue delays.

[2] If bits 4 and 5 are not set, the corresponding errors can be trapped by using an ON ERROR statement.

Datacomm Registers (cont.)

Async Protocol

Bit	Value	Enables interrupt when:
0	1	Data or control block available in receive queue
1	2	Prompt received from remote device
2	4	Framing or parity error detected in incoming data
3	8	A modem line changed
4	16[1]	No Activity timeout forced a disconnection
5	32[1]	Lost Carrier or Connection timeout caused a disconnection
6	64	End-of-line received
7	128	Break received

Reset value = 0

Status 14 Returns current Control Block mask (not supported on BASIC/UX)

Control 14 Sets Control Block mask. Control block information is queued sequentially with incoming data as follows (not supported on BASIC/UX):

Bit	Value	Async Control Block Passed	Data Link Control Block Passed
0	1	Prompt position	Transparent/Normal Mode[2]
1	2	End-of-line position	ETX Block Terminator[3]
2	4	Framing and/or Parity error[4]	ETB Block Terminator[3]
3	8	Break received	

Reset Value: 0 (Control Blocks disabled) 6 (ETX/ETB Enabled)

Bits 4, 5, 6, and 7 are not used.

[1] If bits 4 and 5 are not set, the corresponding errors can be trapped by using an ON ERROR statement.

[2] Transparent/Normal format identification control block occurs at the **beginning** of a given block of data in the receive queue.

[3] ETX and ETB Block Termination identification control blocks occur at the END of a given block of data in the receive queue.

[4] This control block precedes each character containing a parity or framing error.

Datacomm Registers (cont.)

Status 15 Returns current modem line interrupt mask (not supported on BASIC/UX).

Control 15 Sets modem line interrupt mask. Enables an interrupt to ON INTR when Bit 3 of Control Register 13 is set as follows (not supported on BASIC/UX):

Bit	Value	Modem Line to Cause Interrupt
0	1	Data Mode (Data Set Ready)
1	2	Receive Ready (Data Carrier Detect)
2	4	Clear-to-send
3	8	OCR1, Incoming Call (Ring Indicator)
4	16	OCR2, Cable or adapter dependent

Reset Value= 0

Note that bit functions are the same as for STATUS register 7. Functions shown are for male connector cable option for modem connections.

Status 16 Returns current connection timeout limit.

Control 16 Sets Attempted Connection timeout limit. Acceptable values: 1 thru 255 seconds. 0=timeout disabled.
Reset Value=25 seconds

Status 17 Returns current No Activity timeout limit.

Control 17 Sets No Activity timeout limit.
Acceptable values: 1 thru 255 minutes. 0=timeout disabled.
Reset Value=10 minutes (disabled if Async, non-modem handshake).

Status 18 Returns current Lost Carrier timeout limit.

Control 18 Sets Lost Carrier timeout limit in units of 10 ms.
Acceptable values: 1 thru 255. 0=timeout disabled.
Reset Value=40 (400 milliseconds)

Datacomm Registers (cont.)

Status 19 Returns current Transmit timeout limit.

Control 19 Sets Transmit timeout limit (loss of clock or CTS not returned by modem when transmission is attempted).
Acceptable values: 1 thru 255.0=timeout disabled.
Reset Value=10 seconds

Status 20 Returns current transmission speed (baud rate). See table for values.[1]

Control 20 Sets transmission speed (baud rate) as follows:[1]

Register Value	Baud Rate	Register Value	Baud Rate
0	External Clock	8	600
*1	50	9	1200
*2	75	10	1800
*3	110	11	2400
*4	134.5	12	3600
*5	150	13	4800
*6	200	14	9600
7	300	15	19200

* Async only. These values cannot be used with Data Link. These values set transmit speed ONLY for Async; transmit AND receive speed for Data Link. Default value is defined by the interface card configuration switches.

Status 21 Protocol dependent. Returns receive speed (Async) or GID address (Data Link) as specified by Control Register 21 (not supported on BASIC/UX).

Control 21 Protocol dependent. Functions are as follows:[1]

Data Link: Sets Group IDentifier (GID) for terminal. Values 0 thru 26 correspond to identifiers @, A, B,...Y, Z, respectively. Other values cause an error. Default value is 1 ("A").

Async: Sets datacomm receiver speed (baud rate). Values and defaults are the same as for Control Register 20.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Status 22 Protocol dependent. Returns DID (Data Link) or protocol handshake type (Async) as specified by Control Register 22.[1]

Control 22 Protocol dependent. Functions are as follows:[1]

Data Link: Sets Device IDentifier (DID) for terminal. Values are the same as for Control Register 21. Default is determined by interface card configuration switches.

Async: Defines protocol handshake type that is to be used.

Value	**Handshake type**
0	Protocol handshake disabled
1	ENQ/ACK with desktop computer as the host
2	ENQ/ACK, desktop computer as a terminal
3	DC1/DC3, desktop computer as host
4	DC1/DC3, desktop computer as a terminal
5	DC1/DC3, desktop computer as both host and terminal

Status 23 Returns current hardware handshake type (not supported on BASIC/UX).

Control 23 Sets hardware handshake type as follows (not supported on BASIC/UX):
0=Handshake OFF, non-modem connection.
1=FULL-DUPLEX modem connection.
2=HALF-DUPLEX modem connection.
3=Handshake ON, non-modem connection.
Reset Value is determined by interface configuration switches.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Datacomm Registers (cont.)

Status 24 Protocol dependent. Returns value set by preceding CONTROL statement to Control Register 24 (not supported on BASIC/UX).

Control 24 Protocol dependent. Functions as follows (not supported on BASIC/UX): Data Link protocol: Set outbound block size limit.

Value	Block size	Value	Block size
0	512 bytes	4	8 bytes
1	2 bytes	.	.
2	4 bytes	.	.
3	6 bytes	255	510 bytes

Reset outbound block size limit=512 bytes

Async Protocol: Set mask for control characters included in receive data message queue. Bit set: transfer character(s).
Bit cleared: delete character(s).

Bit set	Value	Character(s) passed to receive queue
0	1	Handshake characters (ENQ, ACK, DC1, DC3)
1	2	Inbound End-of-line character(s)
2	4	Inbound Prompt character(s)
3	8	NUL (CHR$(0))
4	16	DEL (CHR$(127))
5	32	CHR$(255)
6	64	Change parity/framing errors to underscores (_) if bit is set.
7	128	Not used

Reset value=127 (bits 0 thru 6 set)

Status 25 Returns number of received errors since power up or reset (not supported on BASIC/UX).

Note

Control Registers 26 through 35, Status Registers 27 through 35, and Control and Status Registers 37 and 39 are used for ASYNC protocol ONLY. They are not available during Data Link operation.

Datacomm Registers (cont.)

Status 26 Protocol dependent (not supported on BASIC/UX)

Data Link protocol:	Returns number of transmit errors (NAKs received) since last interface reset.
Async protocol:	Returns first protocol handshake character (ACK or DC1).

Control 26
(Async only)
Sets first protocol handshake character as follows (not supported on BASIC/UX):
6=ACK, 17=DC1. Other values used for special applications only.
Reset value=17
(DC1). Use ACK when Control Register 22 is set to 1 or 2. Use DC1 when Control Register
22 is set to 3, 4, or 5.

Status 27
(Async only)
Returns second protocol handshake character (not supported on BASIC/UX).

Control 27
(Async only)
Sets second protocol handshake character as follows (not supported on BASIC/UX):
5=ENQ, 19=DC3. Other values used for special applications only.
Reset value=19
(DC3). Use ENQ when Control Register 22 is set to 1 or 2. Use DC3 when Control Register
22 is set to 3, 4, or 5.

Status 28
(Async only)
Returns number of characters in inbound End-of-line delimiter sequence (not supported on BASIC/UX).

Control 28
(Async only)
Sets number of characters in End-of-line delimiter sequence (not supported on BASIC/UX)
Acceptable values are 0 (no EOL delimiter), 1, or 2.
Reset Value=2

Status 29
(Async only)
Returns first End-of-line character (not supported on BASIC/UX).

Control 29
(Async only)
Sets first End-of-line character (not supported on BASIC/UX).
Reset Value=13 (carriage return)

Datacomm Registers (cont.)

Status 30
(Async only)
Returns second End-of-line character (not supported on BASIC/UX).

Control 30
(Async only)
Sets second End-of-line character (not supported on BASIC/UX).
Reset Value=10 (line feed)

Status 31
(Async only)
Returns number of characters in Prompt sequence (not supported on BASIC/UX).

Control 31
(Async only)
Sets number of characters in Prompt sequence. Acceptable values are 0 (Prompt disabled), 1 or 2 (not supported on BASIC/UX).
Reset Value=1

Status 32
(Async only)
Returns first character in Prompt sequence
(not supported on BASIC/UX).

Control 32
(Async only)
Sets first character in Prompt sequence (not supported on BASIC/UX).
Reset Value=17 (DC1)

Status 33
(Async only)
Returns second character in Prompt sequence
(not supported on BASIC/UX).

Control 33
(Async only)
Sets second character in Prompt sequence
(not supported on BASIC/UX).
Reset Value=0 (null)

Status 34
(Async only)
Returns the number of bits per character.

Control 34
(Async only)
Sets the number of bits per character as follows:[1]
0=5 bits/character   2=7 bits/character
1=6 bits/character   3=8 bits/character)
When 8 bits/char, parity must be NONE, ODD, or EVEN.
Reset Value is determined by interface card default switches.

Status 35
(Async only)
Returns the number of stop bits per character.

Control 35
(Async only)
Sets the number of stop bits per character as follows:[1]
0=1 stop bit   1=1.5 stop bits   2=2 stop bits
Reset Value: 2 stop bits if 150 baud or less, otherwise 1 stop bit.
Reset Value is determined by interface configuration switch settings.

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques*

Status 36 Returns current Parity setting.

Control 36 Sets Parity for transmitting and receiving as follows:[1]

Data Link Protocol: 0=NO Parity; Network host is HP 1000 Computer.
1=ODD Parity; Network host is HP 3000 Computer. **Reset Value=0**

Async Protocol: 0=NONE; no parity bit is included with any characters.
1=ODD; Parity bit SET if there is an EVEN number of "1"s in the character body.
2=EVEN; Parity bit OFF if there is an ODD number of "1"s in the character body.
3="0"; Parity bit is always ZERO, but parity is not checked.
4="1"; Parity bit is always SET, but parity is not checked.

Default is determined by interface configuration switches. If 8 bits per character, parity must be NONE, ODD, or EVEN.

Status 37 (Async only) Returns inter-character time gap in character times (not supported on BASIC/UX).

Control 37 (Async only) Sets inter-character time gap in character times (not supported on BASIC/UX).
Acceptable values: 1 thru 255 character times.
0=No gap between characters. **Reset Value=0**

Status 38 Returns Transmit queue status (not supported on BASIC/UX).
If returned value=1, queue is empty, and there are no pending transmissions.

Status 39 (Async only) Returns current Break time (in character times) (not supported on BASIC/UX).

Control 39 (Async only) Sets Break time in character times (not supported on BASIC/UX).
Acceptable values are: 2 thru 255. **Reset Value=4.**

[1] For BASIC/UX information on this register, see Volume 2 of the *BASIC/UX Interfacing Techniques* manual.

Summary of Powerfail Status and Control Registers

This section lists all STATUS and CONTROL registers of the Powerfail-Protection Interface, which is permanently assigned to interface select code 5. This section does not apply to BASIC/UX.

STATUS Register 0 Card Identification is always 5.

CONTROL Register 0 Shut Down. Any non-zero value written to this register will turn off both battery and ac-line power to the computer, which conserves battery power after the service routine has finished responding to a powerfail. If ac-line power is on when this statement is executed, the computer will be turned back on in the normal power-up sequence.

STATUS Register 1 Powerfail Interrupt Cause

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					One Second Left	Power Is Back	Power Has Failed
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 2 — **One Second Left** indicates that approximately one second of battery power remains. The computer will automatically power itself down, even if power is restored before one second has expired.

Bit 1 — **Power Is Back** indicates that ac-line power has been restored.

Bit 0 — **Power Has Failed** indicates that ac-line power has failed (even though it may be back now).

CONTROL Register 1 Not Used.

STATUS Register 2 Interrupt Mask has bit definitions identical to the preceding register (Powerfail Interrupt Cause).

CONTROL Register 2 Not Used

Powerfail Registers (cont.)

STATUS Register 3 Powerfail Status

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Failed Self-Test	Not used			One Second Left	Currently Using Battery	Ac Is Down	In the Powerfail State
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 7 — **Failed Self Test** indicates the outcome of the self test: a 1 indicates failure, and 0 indicates successful results.

Bit 3 — **One Second Left** indicates that approximately one second of battery power remains. The computer will automatically power itself down, even if power is restored before one second has expired.

Bit 2 — **Currently Using Battery** indicates whether or not the battery is being used: 1 indicates it is currently being used for computer power, and 0 indicates that it is not.

Bit 1 — **Ac Is Down** indicates the current status of ac-line power: a 1 indicates that ac power is completely gone. If bit 2 is a 1 and this bit is 0, the battery is being used because ac power is not completely gone but has dropped below an acceptable level; in this case, a "brown-out" condition is indicated.

Bit 0 — **In the Powerfail State** indicates whether or not the computer is currently in the Powerfail State: a 1 indicates Powerfail State, and 0 indicates that the computer is not currently in the Powerfail State. The Powerfail State is exited when power is back and the Power Back Timer reaches the value of the Power Back Delay.

CONTROL Register 3 Not Used.

Powerfail Registers (cont.)

STATUS Register 4 Overheat Protection Timer contains the amount of battery time used during this Powerfail State (in tens of milliseconds). For every second the power is down, it must be back for two seconds to ensure adequate cooling for the machine. Thus, the value of this register bounds the maximum amount of time that can be obtained from the battery, even though 60 seconds may have been specified as the protection time (CONTROL Register 6).

CONTROL Register 4 Not Used.

STATUS Register 5 Power Back Timer contains the time elapsed since power was restored after the last powerfail (in tens of milliseconds).

CONTROL Register 5 Power Back Delay. The value of this register determines the amount of time (in tens of milliseconds) that the computer will delay, after power is back, before leaving the powerfail state (i.e., before generating a "Power Is Back" interrupt). The power-on default value is 50 (500 milliseconds).

STATUS Register 6 Powerfail Timer contains the time elapsed since the last powerfail (in tens of milliseconds).

CONTROL Register 6 Protection Time. The value of register determines the maximum amount of time (in tens of milliseconds) that the computer is to have battery backup. Power-on default is 6000 (60 seconds).

STATUS Register 7 Not Used.

CONTROL Register 7 Powerfail Delay Timer. The contents of this register determine the amount of time (in tens of milliseconds) that the Powerfail-Protection Interface will wait, after a powerfail, before generating a "Power Has Failed" interrupt. Power-on default is 10 (100 milliseconds).

STATUS Registers 8 thru 71 Continuous-Memory Registers contain the 64 bytes of data written by the last CONTROL statement directed to these registers.

CONTROL Registers 8 thru 71 Continuous-Memory Registers. These sixty-four, single-byte registers can be filled with any desired data, one byte (ASCII character) per register.

Summary of GPIO STATUS and CONTROL Registers

STATUS Register 0 Card Identification. Always 3.

CONTROL Register 0 Interface Reset. Any non-zero value causes a reset.

STATUS Register 1 Interrupt and DMA Status.

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Are Enabled	An Interrupt Is Currently Requested	Interrupt Level Switches (Hardware Priority)		Burst-Mode DMA	Word-Mode DMA	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 1 Set PCTL Line. Any non-zero value sets the line.

GPIO Registers (cont.)

STATUS Register 2

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Process	Interrupts Are Enabled	Transfer In Progress
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 2 Peripheral Control

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used					PSTS Error (1=Report; 0=Ignore)	Set CTL1 (1=Low; 0=High)	Set CTL0 (1=Low; 0=High)
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

GPIO Registers (cont.)

STATUS Register 3	Data In (16 bits)
CONTROL Register 3	Data Out (16 bits)
STATUS Register 4	Interface Ready. Interface is Ready for a subsequent data transfer: 1=Ready, 0=Busy.
STATUS Register 5	Peripheral Status

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Ok	EIR Line Low	STI1 Line Low	STI0 Line Low
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not used						Enable Interface Ready Interrupts	Enable EIR Interrupts
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Summary of BCD Status and Control Registers

This section does not apply to BASIC/UX.

STATUS Register 0 Card Identification = 4.

CONTROL Register 0 Reset Interface (if non-zero value sent).

STATUS Register 1 Interrupt Status

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts are enabled	Interrupt Request	Hardware Interrupt Level Switches		0	0	0	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 1 Reset driver pointer (if non-zero value sent).

STATUS Register 2 Busy Bit

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake in progress	Interrupts Enabled	0
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Bit 0 is 1 when a handshake is currently in progress.

CONTROL Register 2 Request data by Setting CTLA and CTLB (if a non-zero value is sent); this operation also clears an Interrupt Request (clears bit 6 of Status Register 1).

BCD Registers (cont.)

STATUS Register 3 Binary Mode: 1 if the interface is currently operating in Binary mode, and 0 if in BCD mode.

CONTROL Register 3 Set Binary Mode: set Binary Mode if non-zero value sent, and BCD Mode if zero sent.

STATUS Register 4 Switch and Line States

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OF Switch Is ON	DATA Switch Is ON	SGN1 Switch Is ON	SGN2 Switch Is ON	OVLD Switch Is ON	SGN1 Input Is True	SGN2 Input Is True	OVLD Input Is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 4 Data Out Lines

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Set DO-7 True	Set DO-6 True	Set DO-5 True	Set DO-4 True	Set DO-3 True	Set DO-2 True	Set DO-1 True	Set DO-0 True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

BCD Registers (cont.)

STATUS Register 5 BCD Digits DI1 and DI2

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI1-8 is True	DI1-4 is True	DI1-2 is True	DI1-1 is True	DI2-8 is True	DI2-4 is True	DI2-2 is True	DI2-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

STATUS Register 6 BCD Digits DI3 and DI4

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI3-8 is True	DI3-4 is True	DI3-2 is True	DI3-1 is True	DI4-8 is True	DI4-4 is True	DI4-2 is True	DI4-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

STATUS Register 7 BCD Digits DI5 and DI6

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI5-8 is True	DI5-4 is True	DI5-2 is True	DI5-1 is True	DI6-8 is True	DI6-4 is True	DI6-2 is True	DI6-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

BCD Registers (cont.)

STATUS Register 8 BCD Digits DI7 and DI8

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7-8 is True	DI7-4 is True	DI7-2 is True	DI7-1 is True	DI8-8 is True	DI8-4 is True	DI8-2 is True	DI8-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

STATUS Register 9 BCD Digits DI9 and DI10

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI9-8 is True	DI9-4 is True	DI9-2 is True	DI9-1 is True	DI10-8 is True	DI10-4 is True	DI10-2 is True	DI10-1 is True
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

Summary of EPROM Programmer STATUS and CONTROL Registers

This section does not apply to BASIC/UX.

STATUS Register 0 ID Register. This register contains a value of 27 (decimal) which is the ID of an EPROM Programmer card.

Most Significant Bit **Least Significant Bit**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	1	1	0	1	1
Value=128	Value=64	Value=32	Value=16	Value=8	Value=4	Value=2	Value=1

CONTROL Register 0 Interface Reset. Writing any non-zero value into this register resets the card; writing a value of zero causes no action.

STATUS Register 1 Read Program Time. A value of 0 indicates that the program time is 52.5 milliseconds for each 16-bit word (default); a non-zero value indicates that the program time is 13.1 milliseconds.

CONTROL Register 1 Set Program Time. Writing a value of 0 into this register sets the program time to 52.5 milliseconds for each 16-bit word; any non-zero value sets program time to 13.1 milliseconds.

STATUS Register 2 Read Target Address. This register contains the offset address (relative to the card's base address) at which the next word of data will be read (via STATUS Register 3) or written (via CONTROL Register 3). The default address is 0, which is the address of the first byte on the card.

CONTROL Register 2 Set Target Address. Writing to this register sets the offset address at which the next word of data will be read (via STATUS Register 3) or written (via CONTROL Register 3). The target address must always be an *even* number.

EPROM Programmer Registers (cont.)

STATUS Register 3 Read Word at Target Address. This register contains the 16-bit word at the current target address.

CONTROL Register 3 Write Word at Target Address. Writing a data word to this register programs a 16-bit word at the current target address. The target address must be set (via CONTROL register 2) before every word is written. Automatic verification is also performed after the word is programmed.

STATUS Register 4 Current Memory Card Capacity (in bytes). This register contains the current capacity of a *fully loaded* card in bytes; it also indirectly indicates which type of EPROM devices are being used on the card. If 262 144 is returned, then 27128 EPROMs are being used; if 131 072 is returned, then 2764 devices are being used. A 0 is returned if the programmer card is not currently connected to any EPROM memory card.

CONTROL Register 4 Undefined.

STATUS Register 5 Number of Contiguous, Erased Bytes. Reading this register causes the system to begin counting the number of subsequent bytes, beginning at the current target address, that are erased (or are empty sockets). The counting is stopped when a programmed byte (i.e., one containing at least one logical 0) is found or when the end of the card is reached. If the byte at the current target address is not FF, then a count of 0 is returned. Error 84 is reported if the programmer card is not currently connected to any EPROM card.

CONTROL Register 5 Undefined.

STATUS Register 6 Base Address of EPROM Memory Card. This register contains the (absolute) base address of the EPROM memory card to which the programmer card is currently connected; this base address is also the absolute address of the first word on the card. Error 84 is reported if the programmer card is not currently connected to any EPROM memory card.

CONTROL Register 6 Undefined.

Parity, Cache, Float, and Clock STATUS and CONTROL Registers (Pseudo Select Code 32)

STATUS Register 0 Parity Checking for Memory Is Currently Enabled/Disabled

0 = currently disabled;
1 = currently enabled

CONTROL Register 0 Enable/Disable Parity Checking for Memory (not supported on BASIC/UX)

0 = disable;
1 = enable

STATUS Register 1 External Cache Is Currently Enabled/Disabled

0 = currently disabled;
1 = currently enabled

CONTROL Register 1 Enable/Disable External Cache (not supported on BASIC/UX)

0 = disable;
1 = enable

STATUS Register 2 Floating-Point Math Hardware Is Currently Enabled/Disabled (HP 98635 Card or MC68881/68882 Co-Processor)

0 = currently disabled;
1 = currently enabled

CONTROL Register 2 Enable/Disable Floating-Point Math (HP 98635 Card or MC68881/68882 Co-Processor)

0 = disable;
1 = enable

Cache and Clock Registers (cont.)

STATUS Register 3 Internal MC68020/68030 Cache Is Currently Enabled/Disabled

0 = currently disabled;
1 = currently enabled

CONTROL Register 3 Enable/Disable Internal MC68020/68030 Cache (not supported on BASIC/UX)

0 = disable;
non-0 = enable

Note

With computers that have a MC68030 processor, enabling or disabling this internal cache also enables/disables the external cache (since they are not independent). To determine which processor you have, use SYSTEM$ ("SYSTEM ID"). A result of S300:20 indicates you have a 68020, and S300:30 indicates a 68030 processor.

STATUS Register 4 Battery-Backed Clock Type

0 = No battery-backed clock present;
1 = Series 200 (98270) battery-backed clock present;
2 = Series 300 (HP-HIL) battery-backed clock present

SRM Interface STATUS Registers

BASIC/UX supports SRM STATUS Registers 3 and 6. All other registers either cause an error or return a value, depending on the current ERRORMODE setting (specified on the rmb command line or in the configuration file). If ERRORMODE is off, then STATUS Register 0 returns 52, and all other registers return 0.

STATUS Register 0 **Card Identification**

52 if the Remote Control switch (R) is set to 0 (closed);
180 if switch is set to 1 (open).

STATUS Register 1 **Interface Interrupts**

1=interrupts enabled:
0=interrupts disabled.

STATUS Register 2 **Interface Busy**

1=busy;
0=not busy.

STATUS Register 3 **Interface Firmware ID**

Always 3 (the firmware ID of the SRM interface).

STATUS Register 4 **Not Implemented**

STATUS Register 5 **Data Availability**

0=receiver buffer empty;
1=receiver data available but no control blocks buffered:
2=receiver control blocks available but no data buffered;
3=both control blocks and data available.

SRM Registers (cont.)

STATUS Register 6 **Node Address of SRM Interface**

Node address of the SRM interface installed in **this** computer which is set to the specified select code. The range of node addresses is 0 through 63.

STATUS Register 7 **CRC Errors**

Total number of cyclic redundancy check (CRC) errors detected by the interface since powerup or Reset (RESET).

STATUS Register 8 **Number of Buffer Overflows**

Total number of times the receive buffer has overflowed since powerup or Reset (RESET).

STATUS Register 11 **Available space**

Amount of available space (number of bytes) in the transmit-data buffer.

STATUS Register 12 **Number of Retries**

Number of transmission retries performed since powerup or Reset (RESET).

EXT Signal Registers

STATUS Register 0 Last un-caught EXT Signal 0

STATUS Register 1 Status of EXT Signal 1

-1=Not catchable
0=Disabled
1=Disabled

STATUS Register 2 Status of EXT Signal 2

⋮

STATUS Register 32 Status of EXT Signal 32

Notes

Table of Contents

Appendix D: Useful Tables

Useful Tables D

Option Numbers

These option numbers are displayed when ERROR 1 is reported.

Option Number	Binary	Option Number	Binary
1	BASIC Main	23	EPROM
2	GRAPH	24	HP 9885
3	GRAPHX	25	HPIB
4	IO	26	FHPIB
5	BASIC Main	27	SERIAL
6	TRANS	28	GPIO
7	MAT	27	BCD
8	PDEV	30	DCOMM
9	XREF	31-40	Reserved
10	KBD	41	"Unavailable"
11	CLOCK	42	CRTB
12	LEX	43	CRTA
13	BASIC Main	44	CRTC
14	MS	45	Reserved
15	SRM	46	COMPLEX
16	Compiler	47	CRTX
17	PCIB[1]	48	EDIT
18	KNB2_0	49	Reserved
19	ERR	50	HFS
20	DISC	51	RMB
21	CS80	53	MUX
22	BUBBLE		

[1] This binary is included in the support software for the HP 98647 PC Instruments Interface. It is not supplied with the BASIC 5.0 system.

Interface Select Codes

Internal Select Codes

Select Code	Device or Interface
1	Display (alpha)
2	Keyboard
3	Display (graphics)
4	Internal floppy-disc drive
5	Optional powerfail protection interface
6	Display (Graphics for bit mapped)/Windows
7	HP-IB interface (built-in)

Factory Presets for External Interfaces

Select Code	Device or Interface
8	HP-IB
9	RS-232
10	(not used)
11	BCD
12	GPIO
14	HP-IB "High-Speed" Disc Interface
20	Data Communications
21	Shared Resource Management
27	EPROM Programmer
28	RGB Color Video
30	Bubble Memory
32	Parity, Cache, Floating-point math hardware, and battery-backed clock (Pseudo Select Code)
33	EXT SIGNAL Registers

Display-Enhancement Characters

Displaying these characters on the CRT (with OUTPUT CRT, PRINT, or DISP, etc.) produce special effects.

Monochrome Enhancements

These characters produce special effects on most monochrome displays.

Character Code	Action Resulting from Displaying the Character
128	All enhancements off.
129	Inverse mode on.
130	Blinking mode on. *
131	Inverse and Blinking modes on. *
132	Underline mode on.
133	Underline and Inverse modes on.
134	Underline and Blinking modes on. *
135	Underline, Inverse, and Blinking modes on. *

* *Blinking not available on bit-mapped alpha displays.*

Color Enhancements

These characters change the alpha pen color on color displays.

Character Code	Model 236C Display	Bit-mapped Alpha Display
136	White	PEN 1
137	Red	PEN 2
138	Yellow	PEN 3
139	Green	PEN 4
140	Cyan	PEN 5
141	Blue	PEN 6
142	Magenta	PEN 7
143	Black	PEN 0

CRT CONTROL registers 5 and 15 through 17 also provide a method of changing the alpha color.

`PRINT`ing `CHR$(`x`)`, where $136 \leq x \leq 143$, will provide the same colors as on the Model 236C as long as the color map contains default values and the alpha write-enable mask includes planes 0 through 2. A user-defined color map which changes the values of pens 0 to 7 will change the meaning of `CHR$(`x`)`.

U.S. ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
NUL	0	00000000	000	00	
SOH	1	00000001	001	01	GTL
STX	2	00000010	002	02	
ETX	3	00000011	003	03	
EOT	4	00000100	004	04	SDC
ENQ	5	00000101	005	05	PPC
ACK	6	00000110	006	06	
BEL	7	00000111	007	07	
BS	8	00001000	010	08	GET
HT	9	00001001	011	09	TCT
LF	10	00001010	012	0A	
VT	11	00001011	013	0B	
FF	12	00001100	014	0C	
CR	13	00001101	015	0D	
SO	14	00001110	016	0E	
SI	15	00001111	017	0F	
DLE	16	00010000	020	10	
DC1	17	00010001	021	11	LLO
DC2	18	00010010	022	12	
DC3	19	00010011	023	13	
DC4	20	00010100	024	14	DCL
NAK	21	00010101	025	15	PPU
SYNC	22	00010110	026	16	
ETB	23	00010111	027	17	
CAN	24	00011000	030	18	SPE
EM	25	00011001	031	19	SPD
SUB	26	00011010	032	1A	
ESC	27	00011011	033	1B	
FS	28	00011100	034	1C	
GS	29	00011101	035	1D	
RS	30	00011110	036	1E	
US	31	00011111	037	1F	

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
space	32	00100000	040	20	LA0
!	33	00100001	041	21	LA1
"	34	00100010	042	22	LA2
#	35	00100011	043	23	LA3
$	36	00100100	044	24	LA4
%	37	00100101	045	25	LA5
&	38	00100110	046	26	LA6
'	39	00100111	047	27	LA7
(	40	00101000	050	28	LA8
)	41	00101001	051	29	LA9
*	42	00101010	052	2A	LA10
+	43	00101011	053	2B	LA11
,	44	00101100	054	2C	LA12
–	45	00101101	055	2D	LA13
.	46	00101110	056	2E	LA14
/	47	00101111	057	2F	LA15
0	48	00110000	060	30	LA16
1	49	00110001	061	31	LA17
2	50	00110010	062	32	LA18
3	51	00110011	063	33	LA19
4	52	00110100	064	34	LA20
5	53	00110101	065	35	LA21
6	54	00110110	066	36	LA22
7	55	00110111	067	37	LA23
8	56	00111000	070	38	LA24
9	57	00111001	071	39	LA25
:	58	00111010	072	3A	LA26
;	59	00111011	073	3B	LA27
<	60	00111100	074	3C	LA28
=	61	00111101	075	3D	LA29
>	62	00111110	076	3E	LA30
?	63	00111111	077	3F	UNL

STD-LL-60182

U.S. ASCII Character Codes

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
@	64	01000000	100	40	TA0
A	65	01000001	101	41	TA1
B	66	01000010	102	42	TA2
C	67	01000011	103	43	TA3
D	68	01000100	104	44	TA4
E	69	01000101	105	45	TA5
F	70	01000110	106	46	TA6
G	71	01000111	107	47	TA7
H	72	01001000	110	48	TA8
I	73	01001001	111	49	TA9
J	74	01001010	112	4A	TA10
K	75	01001011	113	4B	TA11
L	76	01001100	114	4C	TA12
M	77	01001101	115	4D	TA13
N	78	01001110	116	4E	TA14
O	79	01001111	117	4F	TA15
P	80	01010000	120	50	TA16
Q	81	01010001	121	51	TA17
R	82	01010010	122	52	TA18
S	83	01010011	123	53	TA19
T	84	01010100	124	54	TA20
U	85	01010101	125	55	TA21
V	86	01010110	126	56	TA22
W	87	01010111	127	57	TA23
X	88	01011000	130	58	TA24
Y	89	01011001	131	59	TA25
Z	90	01011010	132	5A	TA26
[	91	01011011	133	5B	TA27
\	92	01011100	134	5C	TA28
]	93	01011101	135	5D	TA29
^	94	01011110	136	5E	TA30
_	95	01011111	137	5F	UNT

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Dec	Binary	Oct	Hex	
`	96	01100000	140	60	SC0
a	97	01100001	141	61	SC1
b	98	01100010	142	62	SC2
c	99	01100011	143	63	SC3
d	100	01100100	144	64	SC4
e	101	01100101	145	65	SC5
f	102	01100110	146	66	SC6
g	103	01100111	147	67	SC7
h	104	01101000	150	68	SC8
i	105	01101001	151	69	SC9
j	106	01101010	152	6A	SC10
k	107	01101011	153	6B	SC11
l	108	01101100	154	6C	SC12
m	109	01101101	155	6D	SC13
n	110	01101110	156	6E	SC14
o	111	01101111	157	6F	SC15
p	112	01110000	160	70	SC16
q	113	01110001	161	71	SC17
r	114	01110010	162	72	SC18
s	115	01110011	163	73	SC19
t	116	01110100	164	74	SC20
u	117	01110101	165	75	SC21
v	118	01110110	166	76	SC22
w	119	01110111	167	77	SC23
x	120	01111000	170	78	SC24
y	121	01111001	171	79	SC25
z	122	01111010	172	7A	SC26
{	123	01111011	173	7B	SC27
|	124	01111100	174	7C	SC28
}	125	01111101	175	7D	SC29
~	126	01111110	176	7E	SC30
DEL	127	01111111	177	7F	SC31

U.S./European Display Characters

These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
NU	0	00000000		32	00100000	@	64	01000000	`	96	01100000
SH	1	00000001	!	33	00100001	A	65	01000001	a	97	01100001
SX	2	00000010	"	34	00100010	B	66	01000010	b	98	01100010
EX	3	00000011	#	35	00100011	C	67	01000011	c	99	01100011
ET	4	00000100	$	36	00100100	D	68	01000100	d	100	01100100
EQ	5	00000101	%	37	00100101	E	69	01000101	e	101	01100101
AK	6	00000110	&	38	00100110	F	70	01000110	f	102	01100110
BL	7	00000111	'	39	00100111	G	71	01000111	g	103	01100111
BS	8	00001000	(	40	00101000	H	72	01001000	h	104	01101000
HT	9	00001001	)	41	00101001	I	73	01001001	i	105	01101001
LF	10	00001010	*	42	00101010	J	74	01001010	j	106	01101010
VT	11	00001011	+	43	00101011	K	75	01001011	k	107	01101011
FF	12	00001100	,	44	00101100	L	76	01001100	l	108	01101100
CR	13	00001101	-	45	00101101	M	77	01001101	m	109	01101101
SO	14	00001110	.	46	00101110	N	78	01001110	n	110	01101110
SI	15	00001111	/	47	00101111	O	79	01001111	o	111	01101111
DL	16	00010000	0	48	00110000	P	80	01010000	p	112	01110000
D1	17	00010001	1	49	00110001	Q	81	01010001	q	113	01110001
D2	18	00010010	2	50	00110010	R	82	01010010	r	114	01110010
D3	19	00010011	3	51	00110011	S	83	01010011	s	115	01110011
D4	20	00010100	4	52	00110100	T	84	01010100	t	116	01110100
NK	21	00010101	5	53	00110101	U	85	01010101	u	117	01110101
SY	22	00010110	6	54	00110110	V	86	01010110	v	118	01110110
EB	23	00010111	7	55	00110111	W	87	01010111	w	119	01110111
CN	24	00011000	8	56	00111000	X	88	01011000	x	120	01111000
EM	25	00011001	9	57	00111001	Y	89	01011001	y	121	01111001
SB	26	00011010	:	58	00111010	Z	90	01011010	z	122	01111010
EC	27	00011011	;	59	00111011	[	91	01011011	{	123	01111011
FS	28	00011100	<	60	00111100	\	92	01011100	|	124	01111100
GS	29	00011101	=	61	00111101	]	93	01011101	}	125	01111101
RS	30	00011110	>	62	00111110	^	94	01011110	~	126	01111110
US	31	00011111	?	63	00111111	_	95	01011111	▒	127	01111111

STD-LL-60182

U.S./European Display Characters

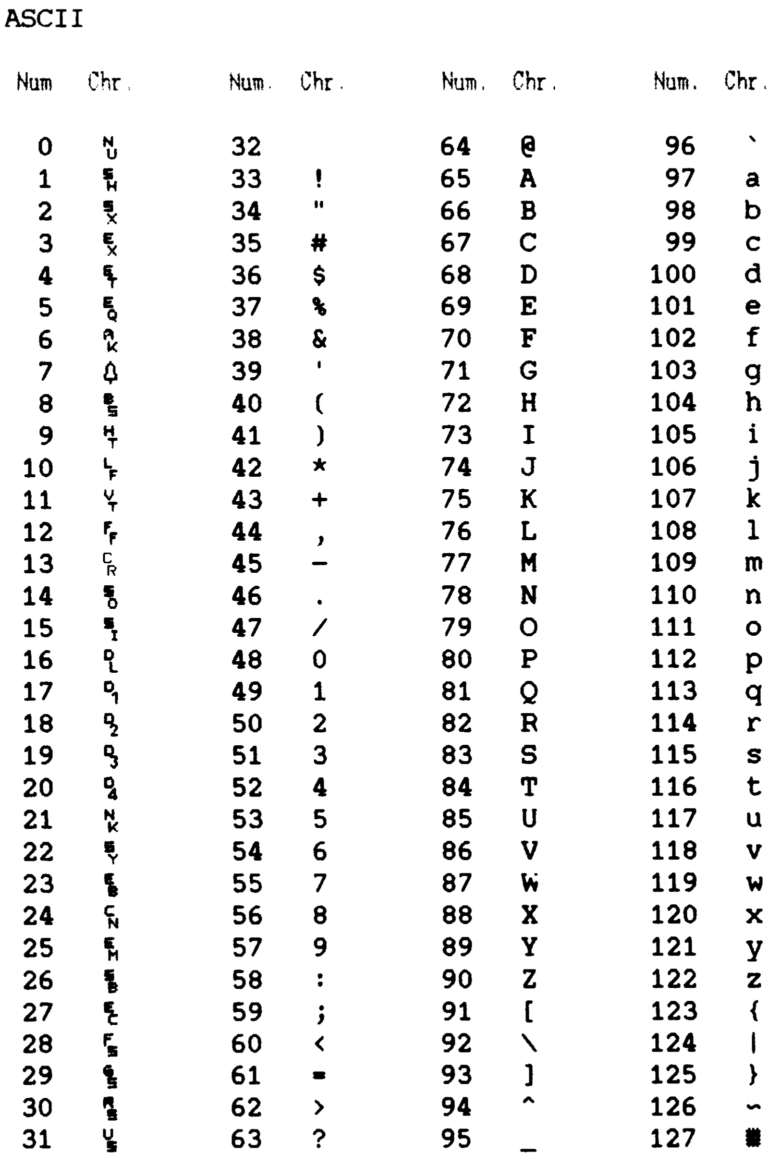
These characters can be displayed on the alpha screens of Models 216, 220 (with a 98204A display), 226, and 236 Computers.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
⊦F	128	10000000	⊦F	160	10100000	â	192	11000000	Á	224	11100000
⊦F	129	10000001	À	161	10100001	ê	193	11000001	Ã	225	11100001
⊦F	130	10000010	Â	162	10100010	ô	194	11000010	ã	226	11100010
⊦F	131	10000011	È	163	10100011	û	195	11000011	Đ	227	11100011
⊦F	132	10000100	Ê	164	10100100	á	196	11000100	đ	228	11100100
⊦F	133	10000101	Ë	165	10100101	é	197	11000101	Í	229	11100101
⊦F	134	10000110	Î	166	10100110	ó	198	11000110	Ì	230	11100110
⊦F	135	10000111	Ï	167	10100111	ú	199	11000111	Ó	231	11100111
⊦F	136	10001000	´	168	10101000	à	200	11001000	Ò	232	11101000
⊦F	137	10001001	`	169	10101001	è	201	11001001	Õ	233	11101001
⊦F	138	10001010	ˆ	170	10101010	ò	202	11001010	õ	234	11101010
⊦F	139	10001011	¨	171	10101011	ù	203	11001011	Š	235	11101011
⊦F	140	10001100	˜	172	10101100	ä	204	11001100	š	236	11101100
⊦F	141	10001101	Ù	173	10101101	ë	205	11001101	Ú	237	11101101
⊦F	142	10001110	Û	174	10101110	ö	206	11001110	Ÿ	238	11101110
⊦F	143	10001111	₤	175	10101111	ü	207	11001111	ÿ	239	11101111
⊦F	144	10010000	¯	176	10110000	Å	208	11010000	Ÿ	240	11110000
⊦F	145	10010001	Ý	177	10110001	î	209	11010001	ÿ	241	11110001
⊦F	146	10010010	ý	178	10110010	Ø	210	11010010	⊦F	242	11110010
⊦F	147	10010011	°	179	10110011	Æ	211	11010011	⊦F	243	11110011
⊦F	148	10010100	Ç	180	10110100	å	212	11010100	⊦F	244	11110100
⊦F	149	10010101	ç	181	10110101	í	213	11010101	⊦F	245	11110101
⊦F	150	10010110	Ñ	182	10110110	ø	214	11010110	⊦F	246	11110110
⊦F	151	10010111	ñ	183	10110111	æ	215	11010111	⊦F	247	11110111
⊦F	152	10011000	¡	184	10111000	Ä	216	11011000	⊦F	248	11111000
⊦F	153	10011001	¿	185	10111001	ì	217	11011001	⊦F	249	11111001
⊦F	154	10011010	¤	186	10111010	Ö	218	11011010	⊦F	250	11111010
⊦F	155	10011011	£	187	10111011	Ü	219	11011011	⊦F	251	11111011
⊦F	156	10011100	⊦F	188	10111100	É	220	11011100	⊦F	252	11111100
⊦F	157	10011101	§	189	10111101	ï	221	11011101	⊦F	253	11111101
⊦F	158	10011110	⊦F	190	10111110	ß	222	11011110	⊦F	254	11111110
⊦F	159	10011111	⊦F	191	10111111	Ô	223	11011111	▓	255	11111111

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.



ASCII

Num	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	NU	32		64	@	96	`
1	SH	33	!	65	A	97	a
2	SX	34	"	66	B	98	b
3	EX	35	#	67	C	99	c
4	ET	36	$	68	D	100	d
5	EQ	37	%	69	E	101	e
6	AK	38	&	70	F	102	f
7	🔔	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DL	48	0	80	P	112	p
17	D1	49	1	81	Q	113	q
18	D2	50	2	82	R	114	r
19	D3	51	3	83	S	115	s
20	D4	52	4	84	T	116	t
21	NK	53	5	85	U	117	u
22	SY	54	6	86	V	118	v
23	EB	55	7	87	W	119	w
24	CN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SB	58	:	90	Z	122	z
27	EC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	▓

U.S./European Display Characters

These characters can be displayed on the alpha screen of Series 200 Model 217, 220 (with 98204B display), and 237 computers, and on Series 300 computers using a 98546 Display Compatibility Interface or 98700 Display Controller.

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	CL	160		192	â	224	Á
129	IV	161	À	193	ê	225	Ã
130	BG	162	Â	194	ô	226	ã
131	IB	163	È	195	û	227	Đ
132	UL	164	Ê	196	á	228	đ
133	IV	165	Ë	197	é	229	Í
134	BG	166	Î	198	ó	230	Ì
135	IV	167	Ï	199	ú	231	Ó
136	WH	168	´	200	à	232	Ò
137	RD	169	`	201	è	233	Õ
138	YE	170	^	202	ò	234	õ
139	GR	171	¨	203	ù	235	Š
140	CY	172	~	204	ä	236	š
141	BU	173	Ù	205	ë	237	Ú
142	MG	174	Û	206	ö	238	Ÿ
143	BK	175	£	207	ü	239	ÿ
144	90	176	‾	208	Å	240	Þ
145	91	177	B1	209	î	241	þ
146	92	178	B2	210	Ø	242	F2
147	93	179	°	211	Æ	243	F3
148	94	180	Ç	212	å	244	F4
149	95	181	ç	213	í	245	IO
150	96	182	Ñ	214	ø	246	–
151	97	183	ñ	215	æ	247	¼
152	98	184	¡	216	Ä	248	½
153	99	185	¿	217	ì	249	ª
154	9A	186	¤	218	Ö	250	º
155	9B	187	£	219	Ü	251	«
156	9C	188	¥	220	É	252	■
157	9D	189	§	221	ï	253	»
158	9E	190	ƒ	222	ß	254	±
159	9F	191	¢	223	Ô	255	K

U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

ASCII

Num	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
0	NU	32		64	@	96	`
1	SH	33	!	65	A	97	a
2	SX	34	"	66	B	98	b
3	EX	35	#	67	C	99	c
4	ET	36	$	68	D	100	d
5	EQ	37	%	69	E	101	e
6	AK	38	&	70	F	102	f
7	🔔	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DL	48	0	80	P	112	p
17	D1	49	1	81	Q	113	q
18	D2	50	2	82	R	114	r
19	D3	51	3	83	S	115	s
20	D4	52	4	84	T	116	t
21	NK	53	5	85	U	117	u
22	SY	54	6	86	V	118	v
23	EB	55	7	87	W	119	w
24	CN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SB	58	:	90	Z	122	z
27	EC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	▒

U.S./European Display Characters

These characters can be displayed on the screen of Series 300 computers (except with a 98546 Display Compatibility Interface or 98700 Display Controller; see the preceding table).

ASCII

Num.	Chr.	Num.	Chr.	Num.	Chr.	Num.	Chr.
128	CL	160		192	â	224	Á
129	IV	161	À	193	ê	225	Ã
130	BG	162	Â	194	ô	226	ã
131	IB	163	È	195	û	227	Đ
132	UL	164	Ê	196	á	228	đ
133	IV	165	Ë	197	é	229	Í
134	BG	166	Î	198	ó	230	Ì
135	IV	167	Ï	199	ú	231	Ó
136	WH	168	´	200	à	232	Ò
137	RD	169	`	201	è	233	Õ
138	YE	170	ˆ	202	ò	234	õ
139	GR	171	¨	203	ù	235	Š
140	CY	172	˜	204	ä	236	š
141	BU	173	Ù	205	ë	237	Ú
142	MG	174	Û	206	ö	238	Ÿ
143	BK	175	₤	207	ü	239	ÿ
144	90	176	¯	208	Ȧ	240	Þ
145	91	177	Ý	209	î	241	þ
146	92	178	ý	210	Ø	242	·
147	93	179	°	211	Æ	243	µ
148	94	180	Ç	212	ȧ	244	¶
149	95	181	ç	213	í	245	IO
150	96	182	Ñ	214	ø	246	—
151	97	183	ñ	215	æ	247	¼
152	98	184	¡	216	Ä	248	½
153	99	185	¿	217	ì	249	ª
154	9A	186	¤	218	Ö	250	º
155	9B	187	£	219	Ü	251	«
156	9C	188	¥	220	É	252	■
157	9D	189	§	221	ï	253	»
158	9E	190	ƒ	222	ß	254	±
159	9F	191	¢	223	Ô	255	🅺

Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
␀	0	00000000		32	00100000	@	64	01000000	`	96	01100000
␁	1	00000001	!	33	00100001	A	65	01000001	a	97	01100001
␂	2	00000010	"	34	00100010	B	66	01000010	b	98	01100010
␃	3	00000011	#	35	00100011	C	67	01000011	c	99	01100011
␄	4	00000100	$	36	00100100	D	68	01000100	d	100	01100100
␅	5	00000101	%	37	00100101	E	69	01000101	e	101	01100101
␆	6	00000110	&	38	00100110	F	70	01000110	f	102	01100110
␇	7	00000111	'	39	00100111	G	71	01000111	g	103	01100111
␈	8	00001000	(	40	00101000	H	72	01001000	h	104	01101000
␉	9	00001001	)	41	00101001	I	73	01001001	i	105	01101001
␊	10	00001010	*	42	00101010	J	74	01001010	j	106	01101010
␋	11	00001011	+	43	00101011	K	75	01001011	k	107	01101011
␌	12	00001100	,	44	00101100	L	76	01001100	l	108	01101100
␍	13	00001101	-	45	00101101	M	77	01001101	m	109	01101101
␎	14	00001110	.	46	00101110	N	78	01001110	n	110	01101110
␏	15	00001111	/	47	00101111	O	79	01001111	o	111	01101111
␐	16	00010000	0	48	00110000	P	80	01010000	p	112	01110000
␑	17	00010001	1	49	00110001	Q	81	01010001	q	113	01110001
␒	18	00010010	2	50	00110010	R	82	01010010	r	114	01110010
␓	19	00010011	3	51	00110011	S	83	01010011	s	115	01110011
␔	20	00010100	4	52	00110100	T	84	01010100	t	116	01110100
␕	21	00010101	5	53	00110101	U	85	01010101	u	117	01110101
␖	22	00010110	6	54	00110110	V	86	01010110	v	118	01110110
␗	23	00010111	7	55	00110111	W	87	01010111	w	119	01110111
␘	24	00011000	8	56	00111000	X	88	01011000	x	120	01111000
␙	25	00011001	9	57	00111001	Y	89	01011001	y	121	01111001
␚	26	00011010	:	58	00111010	Z	90	01011010	z	122	01111010
␛	27	00011011	;	59	00111011	[	91	01011011	{	123	01111011
␜	28	00011100	<	60	00111100	¥	92	01011100	|	124	01111100
␝	29	00011101	=	61	00111101	]	93	01011101	}	125	01111101
␞	30	00011110	>	62	00111110	^	94	01011110	~	126	01111110
␟	31	00011111	?	63	00111111	_	95	01011111	▒	127	01111111

STD-LL-60182

Katakana Display Characters

These characters can be displayed on the screen of Model 216, 217, 220, 226, and 236 computers, and on Series 300 computers using a 98546 Display Compatibility Interface.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
FP	128	10000000	FP	160	10100000	ﾀ	192	11000000	FP	224	11100000
FP	129	10000001	｡	161	10100001	ﾁ	193	11000001	FP	225	11100001
FP	130	10000010	｢	162	10100010	ﾂ	194	11000010	FP	226	11100010
FP	131	10000011	｣	163	10100011	ﾃ	195	11000011	FP	227	11100011
FP	132	10000100	､	164	10100100	ﾄ	196	11000100	FP	228	11100100
FP	133	10000101	･	165	10100101	ﾅ	197	11000101	FP	229	11100101
FP	134	10000110	ｦ	166	10100110	ﾆ	198	11000110	FP	230	11100110
FP	135	10000111	ｧ	167	10100111	ﾇ	199	11000111	FP	231	11100111
FP	136	10001000	ｨ	168	10101000	ﾈ	200	11001000	FP	232	11101000
FP	137	10001001	ｩ	169	10101001	ﾉ	201	11001001	FP	233	11101001
FP	138	10001010	ｪ	170	10101010	ﾊ	202	11001010	FP	234	11101010
FP	139	10001011	ｫ	171	10101011	ﾋ	203	11001011	FP	235	11101011
FP	140	10001100	ｬ	172	10101100	ﾌ	204	11001100	FP	236	11101100
FP	141	10001101	ｭ	173	10101101	ﾍ	205	11001101	FP	237	11101101
FP	142	10001110	ｮ	174	10101110	ﾎ	206	11001110	FP	238	11101110
FP	143	10001111	ｯ	175	10101111	ﾏ	207	11001111	FP	239	11101111
FP	144	10010000	ｰ	176	10110000	ﾐ	208	11010000	FP	240	11110000
FP	145	10010001	ｱ	177	10110001	ﾑ	209	11010001	FP	241	11110001
FP	146	10010010	ｲ	178	10110010	ﾒ	210	11010010	FP	242	11110010
FP	147	10010011	ｳ	179	10110011	ﾓ	211	11010011	FP	243	11110011
FP	148	10010100	ｴ	180	10110100	ﾔ	212	11010100	FP	244	11110100
FP	149	10010101	ｵ	181	10110101	ﾕ	213	11010101	FP	245	11110101
FP	150	10010110	ｶ	182	10110110	ﾖ	214	11010110	FP	246	11110110
FP	151	10010111	ｷ	183	10110111	ﾗ	215	11010111	FP	247	11110111
FP	152	10011000	ｸ	184	10111000	ﾘ	216	11011000	FP	248	11111000
FP	153	10011001	ｹ	185	10111001	ﾙ	217	11011001	FP	249	11111001
FP	154	10011010	ｺ	186	10111010	ﾚ	218	11011010	FP	250	11111010
FP	155	10011011	ｻ	187	10111011	ﾛ	219	11011011	FP	251	11111011
FP	156	10011100	ｼ	188	10111100	ﾜ	220	11011100	FP	252	11111100
FP	157	10011101	ｽ	189	10111101	ﾝ	221	11011101	FP	253	11111101
FP	158	10011110	ｾ	190	10111110	ﾞ	222	11011110	FP	254	11111110
FP	159	10011111	ｿ	191	10111111	ﾟ	223	11011111	█	255	11111111

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS		ASCII Char.	EQUIVALENT FORMS	
	Dec	Binary		Dec	Binary		Dec	Binary		Dec	Binary
NU	0	00000000		32	00100000	@	64	01000000	`	96	01100000
SH	1	00000001	!	33	00100001	A	65	01000001	a	97	01100001
SX	2	00000010	"	34	00100010	B	66	01000010	b	98	01100010
EX	3	00000011	#	35	00100011	C	67	01000011	c	99	01100011
ET	4	00000100	$	36	00100100	D	68	01000100	d	100	01100100
EQ	5	00000101	%	37	00100101	E	69	01000101	e	101	01100101
AK	6	00000110	&	38	00100110	F	70	01000110	f	102	01100110
	7	00000111	'	39	00100111	G	71	01000111	g	103	01100111
BS	8	00001000	(	40	00101000	H	72	01001000	h	104	01101000
HT	9	00001001	)	41	00101001	I	73	01001001	i	105	01101001
LF	10	00001010	*	42	00101010	J	74	01001010	j	106	01101010
VT	11	00001011	+	43	00101011	K	75	01001011	k	107	01101011
FF	12	00001100	,	44	00101100	L	76	01001100	l	108	01101100
CR	13	00001101	-	45	00101101	M	77	01001101	m	109	01101101
SO	14	00001110	.	46	00101110	N	78	01001110	n	110	01101110
SI	15	00001111	/	47	00101111	O	79	01001111	o	111	01101111
DL	16	00010000	0	48	00110000	P	80	01010000	p	112	01110000
D1	17	00010001	1	49	00110001	Q	81	01010001	q	113	01110001
D2	18	00010010	2	50	00110010	R	82	01010010	r	114	01110010
D3	19	00010011	3	51	00110011	S	83	01010011	s	115	01110011
D4	20	00010100	4	52	00110100	T	84	01010100	t	116	01110100
NK	21	00010101	5	53	00110101	U	85	01010101	u	117	01110101
SY	22	00010110	6	54	00110110	V	86	01010110	v	118	01110110
EB	23	00010111	7	55	00110111	W	87	01010111	w	119	01110111
CN	24	00011000	8	56	00111000	X	88	01011000	x	120	01111000
EM	25	00011001	9	57	00111001	Y	89	01011001	y	121	01111001
SB	26	00011010	:	58	00111010	Z	90	01011010	z	122	01111010
EC	27	00011011	;	59	00111011	[	91	01011011	{	123	01111011
FS	28	00011100	<	60	00111100	¥	92	01011100	|	124	01111100
GS	29	00011101	=	61	00111101	]	93	01011101	}	125	01111101
RS	30	00011110	>	62	00111110	^	94	01011110	‾	126	01111110
US	31	00011111	?	63	00111111	_	95	01011111	▒	127	01111111

STD-LL-60182

Katakana Display Characters

These characters can be displayed on the Model 237 and on all Series 300 bit-mapped alpha displays.

ASCII	EQUIVALENT FORMS		ASCII	EQUIVALENT FORMS		ASCII	EQUIVALENT FORMS		ASCII	EQUIVALENT FORMS	
Char.	Dec	Binary	Char.	Dec	Binary	Char.	Dec	Binary	Char.	Dec	Binary
CL	128	10000000	LF	160	10100000	ﾀ	192	11000000	E0	224	11100000
IV	129	10000001	｡	161	10100001	ﾁ	193	11000001	E1	225	11100001
BG	130	10000010	｢	162	10100010	ﾂ	194	11000010	E2	226	11100010
IB	131	10000011	｣	163	10100011	ﾃ	195	11000011	E3	227	11100011
UL	132	10000100	､	164	10100100	ﾄ	196	11000100	E4	228	11100100
IV	133	10000101	･	165	10100101	ﾅ	197	11000101	E5	229	11100101
BG	134	10000110	ｦ	166	10100110	ﾆ	198	11000110	E6	230	11100110
IV	135	10000111	ｧ	167	10100111	ﾇ	199	11000111	E7	231	11100111
WH	136	10001000	ｨ	168	10101000	ﾈ	200	11001000	E8	232	11101000
RD	137	10001001	ｩ	169	10101001	ﾉ	201	11001001	E9	233	11101001
YE	138	10001010	ｪ	170	10101010	ﾊ	202	11001010	EA	234	11101010
GR	139	10001011	ｫ	171	10101011	ﾋ	203	11001011	EB	235	11101011
CY	140	10001100	ｬ	172	10101100	ﾌ	204	11001100	EC	236	11101100
BU	141	10001101	ｭ	173	10101101	ﾍ	205	11001101	ED	237	11101101
MG	142	10001110	ｮ	174	10101110	ﾎ	206	11001110	EE	238	11101110
BK	143	10001111	ｯ	175	10101111	ﾏ	207	11001111	EF	239	11101111
90	144	10010000	ｰ	176	10110000	ﾐ	208	11010000	F0	240	11110000
91	145	10010001	ｱ	177	10110001	ﾑ	209	11010001	F1	241	11110001
92	146	10010010	ｲ	178	10110010	ﾒ	210	11010010	F2	242	11110010
93	147	10010011	ｳ	179	10110011	ﾓ	211	11010011	F3	243	11110011
94	148	10010100	ｴ	180	10110100	ﾔ	212	11010100	F4	244	11110100
95	149	10010101	ｵ	181	10110101	ﾕ	213	11010101	I0	245	11110101
96	150	10010110	ｶ	182	10110110	ﾖ	214	11010110	F6	246	11110110
97	151	10010111	ｷ	183	10110111	ﾗ	215	11010111	F7	247	11110111
98	152	10011000	ｸ	184	10111000	ﾘ	216	11011000	F8	248	11111000
99	153	10011001	ｹ	185	10111001	ﾙ	217	11011001	F9	249	11111001
9A	154	10011010	ｺ	186	10111010	ﾚ	218	11011010	FA	250	11111010
9B	155	10011011	ｻ	187	10111011	ﾛ	219	11011011	FB	251	11111011
9C	156	10011100	ｼ	188	10111100	ﾜ	220	11011100	■	252	11111100
9D	157	10011101	ｽ	189	10111101	ﾝ	221	11011101	FD	253	11111101
9E	158	10011110	ｾ	190	10111110	ﾞ	222	11011110	FE	254	11111110
9F	159	10011111	ｿ	191	10111111	ﾟ	223	11011111	▨	255	11111111

STD-LL-60182

Note 1: Characters 128 thru 135 produce highlights on machines with monochrome highlights when used in PRINT and DISP statements.
Note 2: Characters 136 thru 143 change the color of text printed or displayed on machines capable of displaying text in color.
Note 3: Characters 144 thru 159 are ignored by PRINT and DISP statements.

Master Reset Table

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
CRT														
CRT DISP Line	Clear	Clear	—	—	Clear	—	—	—	—	—	—	—	—	—
CRT Display Functions	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
CRT Message Line	Ready	Clear	Clear	Clear	Reset	—	—	—	—	—	—	Clear	—	—
CRT Input Line (Note 6)	Clear	Clear	Clear	—	Clear	—	—	—	—	—	—	—	—	—
CRT Printout Area	Clear	Clear	—	—	—	—	—	—	—	—	—	—	—	—
CRT Print Position (TABXY)	1,1	1,1	—	—	Note 15	—	—	—	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—	—	—	—	—	—	—
KEYBOARD														
Keyboard Recall Buffer	Clear	—	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Result Buffer	Empty	Empty	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Knob Mode	↕	↕	↕	↕	↕	—	—	—	—	—	—	—	—	—
Tabs On Input Line	None	None	—	—	—	—	—	—	—	—	—	—	—	—
Typing Aid Labels	Note 16	Note 16	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Katakana Mode	Off	Off	Off	—	Off	—	—	—	—	—	—	—	—	—
SUSPEND INTERACTIVE	Off	Off	Off	Off	Off	Off	Off	Off	Off	Off	—	Off	—	—
PRINTING														
Print column	1	1	—	—	1	—	—	—	—	—	—	—	—	—
PRINTALL	Off	Off	—	—	Off	—	—	—	—	—	—	—	—	—
PRINTALL IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
PRINTER IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
ENVIRONMENTS & VARIABLES														
Allocated Variables	None	None	None	None	Note 1	Note 1	None	None	None	None	—	None	None	Pre-ent
Normal Variables	None	None	None	None	—	—	None	None	None	None	—	Note 11	Note 11	Pre-ent
COM Variables	None	None	—	None	—	—	—	Note 9	—	Note 9	—	—	—	—
OPTION BASE	0	0	0	—	—	—	—	Note 9	—	Note 9	—	Note 9	Note 9	Pre-ent
I/O Path Names	None	Closed	Closed	Closed	None	Closed	Closed	Closed	Closed	Closed	—	Closed	—	sub clsd
I/O Path Names in COM	None	Closed	—	Closed	None	—	Note 10	Note 10	Note 10	Note 10	—	—	—	—
Keyboard Variable Access	No	No	No	No	Main	Main	No	In cnt.	No	In cnt.	In cnt.	Main	SUB	Pre-ent
BASIC Program Lines	None	None	None	—	—	—	Note 4	Note 4	Note 4	Note 4	Note 4	—	—	—
BASIC Program Environment	Main	Main	Main	Main	Main	Main	Main	Main	Main	Main	—	Main	SUB	Pre-ent
Normal Binary Programs	None	None	—	—	—	—	Note 5	Note 5	—	—	—	—	—	—
SUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Push	Pop
NPAR	0	0	0	0	0	0	0	0	0	0	—	0	Actual	Pre-ent
CONTINUE Allowed	No	No	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
ON <event> ACTIONS														
ON <event> Log	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	Empty	Note 8	Note 8
System Priority	0	0	0	0	0	0	0	0	0	0	—	0	Note 7	Pre-ent
ON KEY Labels	None	None	None	None	None	None	None	None	None	None	—	None	—	Pre-ent
ENABLE/DISABLE	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	—	Enable	—	—
KNOBX & KNOBY	0	0	0	0	0	0	0	0	0	0	—	0	—	—
ON EXT SIGNAL	Dflt	Dflt	Dflt	-	Dflt	Dflt	-	-	-	-	Dflt	Dflt	Note 8	Note 8

Note 20: For SRM files, RESET closes the file. For LIF and HFS files, RESET removes the I/O path name, but does not close the file. All other I/O path names at RESET are removed without any other action.

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
MISC.														
GOSUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Local	Pre-ent
TIMEDATE	Note 14	—	—	—	—	—	—	—	—	—	—	—	—	—
ERRL, ERRN, and ERRDS	0	0	—	—	—	—	—	0	—	0	—	0	—	—
ERRM$	Null	Null	—	—	—	—	—	Null	—	Null	—	Null	—	—
DATA Pointer	None	None	None	None	None	None	None	1st main	None	1st main	—	1st main	1st sub	Pre-ent
LEXICAL ORDER IS	Stand.	Stand.	—	—	—	—	—	—	—	—	—	—	—	—
MASS STORAGE IS	Note 12	Note 12	—	—	—	—	—	—	—	—	—	—	—	—
CHECKREAD ON/OFF	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
Angle Mode	RAD	RAD	RAD	RAD	—	—	RAD	RAD	RAD	RAD	—	RAD	—	Pre-ent
Random Number Seed	Note 13	Note 13	Note 13	—	—	—	—	Note 13	—	Note 13	—	Note 13	—	—
DET	0	0	0	—	—	—	—	—	—	—	—	0	—	—
TRANSFER	None	Aborts	Note 17	Waits	Aborts	Waits	None	Note 18	None	Waits	—	None	—	Note 19
TRACE ALL	Off	Off	Off	—	—	—	—	—	—	—	—	—	—	—

— = Unchanged
Pre-ent = As existed previous to entry into the subprogram.
In cnt. = Access to variables in current context only.
1st main = Pointer set to first DATA statement in main program.
1st sub = Pointer set to first DATA statement in subprogram.
sub clsd = All local I/O path names are closed at subexit.
Waits = Operation waits until TRANSFER completes.

Note 1: Only those allocated in the main program are available.

Note 2: Pressing the STOP key is identical in function to executing STOP. Editing or altering a paused program causes the program to go into the stopped state.

Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.

Note 4: Modified according to the statement or command parameters and file contents.

Note 5: Any new binary programs in the file are loaded.

Note 6: Includes cursor position, INS CHR mode, ANY CHAR sequence state, but not tabs, auto-repeat rate, and auto-repeat delay. (These last three are defaulted only at SCRATCH A and Power On.)

Note 7: The system priority changes at SUB entry if the subroutine was invoked by an ON <event> CALL.

Note 8: See the appropriate keyword.

Note 9: As specified by the new environment or program.

Note 10: A COM mismatch between programs will close I/O path names. If I/O path names exist in a labeled COM, and a LOAD or GET brings in a program which does not contain that labeled COM, those I/O path names are closed.

Note 11: Numeric variables are set to 0, and string lengths are set to 0.

Note 12: The default mass storage device is INTERNAL (the right-hand drive) on the 9826 and 9836. See the 9816 Installation Manual for information on its default mass storage device.

Further Comments

Note 13: The default random number seed is INT(PI × $(2^{31} - 2)/180$). This is equal to 37 480 660.

Note 14: The default TIMEDATE is 2.086 629 12 E + 11 (midnight March 1, 1900, Julian time).

Note 15: After a RESET, the CRT print position is in column one of the next line below the print position before the RESET.

Note 16: Typing aid labels are displayed unless a program is in the RUN state.

Note 17: Operation waits until TRANSFER completes unless both I/O path names are in COM.

Note 18: Operation waits until TRANSFER completes unless both I/O path names are in a COM area preserved during the LOAD.

Note 19: Operation waits until TRANSFER completes if the TRANSFER uses a local I/O path name.

The PAUSE key, the programmed PAUSE statement, and executing PAUSE from the keyboard all have identical effects. The only permanent effects of the sequence "PAUSE...CONTINUE" on a running program are:

1. Delay in execution.
2. Second and subsequent interrupt events of a given type are ignored.
3. INPUT, LINPUT, and ENTER 2 statements will be restarted.
4. ON KEY and ON KNOB are temporarily deactivated (i.e. not logged or executed) during the pause.
5. A TRANSFER may complete during the pause, causing ON EOT to be serviced at the next end-of-line.

Fatal program errors (i.e. those not trapped by ON ERROR) have the following effects:

— a PAUSE
— a beep
— an error message in the message line
— setting the values of the ERRL, the ERRN, and possibly the ERRDS functions
— setting the default EDIT line number to the number of the line in which the error occurred.

Autostart is equivalent to: Power On, LOAD "AUTOST", RUN.

CLR IO terminates ENTER and OUTPUT on all interfaces, handshake setup operations, HP-IB control operations, DISP, ENTER from CRT or keyboard, CAT, LIST, external plotter output, and output to the PRINTER IS, PRINTALL IS, and DUMP DEVICE IS devices when they are external. CLR IO does not terminate CONTROL, STATUS, READIO, WRITEIO, TRANSFER, real-time clock operations, mass storage operations (other than CAT), OUTPUT 2 (keyboard), or message line output.

CLR IO clears any pending closure key action.

If CLR IO is used to abort a DUMP GRAPHICS to an external device, the external device may be in the middle of an escape-code sequence. Thus, it might be counting characters to determine when to return to normal mode (from graphics mode). This means that a subsequent I/O operation to the same device may yield "strange" results. Handling this situation is the responsibility of the user and is beyond the scope of the firmware provided with the product. Sending 75 ASCII nulls is one way to "clear" the 9876 Graphics Printer.

Graphic Reset Table

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END STOP	GINIT	Main Prerun
PLOTTER IS	CRT	CRT	—	—	CRT	—	CRT	—
Graphics Memory	Clear	Clear	—	—	Note 1	—	Note 1	—
VIEWPORT	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
X and Y Scaling (unit of measure)	GDU	GDU	—	—	GDU	—	GDU	—
Soft Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
Current Clip	hrd clip	hrd clip	—	—	hrd clip	—	hrd clip	—
CLIP ON OFF	Off	Off	—	—	Off	—	Off	—
PIVOT	0	0	—	—	0	—	0	—
AREA PEN	1	1	—	—	1	—	1	—
PEN	1	1	—	—	1	—	1	—
LINE TYPE	1.5	1.5	—	—	1.5	—	1.5	—
Pen Position	0.0	0.0	—	—	0.0	—	0.0	—
LORG	1	1	—	—	1	—	1	—
CSIZE	5..6	5..6	—	—	5..6	—	5..6	—
LDIR	0	0	—	—	0	—	0	—
PDIR	0	0	—	—	0	—	0	—
GRAPHICS ON OFF	Off	Off	—	—	—	—	—	—
ALPHA ON OFF (Note 3)	On	On	On	On	On	On	—	—
DUMP DEVICE IS	701	701	—	—	—	—	—	—
GRAPHICS INPUT IS	None	None	—	—	None	—	None	—
TRACK ... ON OFF	Off	Off	—	—	Off	—	Off	—
Color Map (Note 4)	Off	Off	—	—	Note 5	—	Note 5	—
Drawing Mode	Norm	Norm	—	—	Norm	—	Norm	—

— = Unchanged
hrd clip = The default hard clip boundaries of the CRT.

Note 1: Although RESET leaves the graphics memory unchanged, it will be cleared upon execution of the next graphics statement that sets a default plotter following the RESET.

Note 2: Pressing the STOP key is identical to executing STOP. Altering a paused program causes the program to go into the stopped state.

Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.

Note 4: With color map off, 8 standard colors are available. With color map on, 16 user-defined colors are available. See PLOTTER IS.

Note 5: Although the color map remains unchanged, it is changed if a graphics statement selects the device as a default plotter.

Interface Reset Table

		SCRATCH A	SCRATCH		Note 5			Note 6				
	Power			BASIC	END/	LOAD	GET	Reset	Main	SUB	SUB	CLR
	On			RESET	STOP			Cmd	Prerun	Entry	Exit	I/O
GPIO Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card (PRESET)	Reset	Note 1	Note 1	Reset	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
PSTS Error Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
RS-232 Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Data Rate/Character Format	Swtch	Swtch	—	—	—	—	—	—	—	—	—	—
RTS-DTR Latch	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Request to Send Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Data Terminal Ready Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Line Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Modem Status Register	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Data-In Buffer	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	—	Empty
Error Pend. Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
HP-IB												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
User Interrupt Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Serial Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
Parallel Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
My Address Register	Note 4	Note 4	—	—	—	—	—	—	—	—	—	—
IFC Sent	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
REN Set True	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
Data Communications												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 7	—	Reset	—	—	—	Note 7	—	—	—	—
Line State	Dscon	Dscor	—	Dscon	—	—	—	Dscon	—	—	—	—
Data Buffers	Empty	Empty	—	Empty	—	—	—	Empty	—	—	—	—
Protocol Selection (Async or Data Link)	Swtch	Note 8	—	Swtch	—	—	—	Note 8	—	—	—	—
Protocol Options	Swtch	Swtch	—	Swtch	—	—	—	Swtch	—	—	—	—

					Note 5			Note 6				
	Power	SCRATCH A	SCRATCH	BASIC	END/	LOAD	GET	Reset	Main	SUB	SUB	CLR
	On			RESET	STOP			Cmd	Prerun	Entry	Exit	I/O
BCD Card												
Interrupt Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 1	Note 1	Note 1	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
Rewind Driver	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	Rwd	—	—	Rwd
BCD/Binary Mode	Swtch	Swtch	—	—	—	—	—	—	—	—	—	—
EPROM Programmer												
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Programming Time Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—
Target Address Register	Clear	Clear	—	—	—	—	—	Clear	—	—	—	—

— = Unchanged

Swtch = Set according to the switches on the interface card

Dscon = A disconnect is performed

Note 1: Reset only if card is not ready.

Note 2: Cleared only if corresponding modem control line is not set.

Note 3: Sent only if System Controller.

Note 4: If System Controller and Active Controller, address is set to 21. Otherwise, it is set to 20.

Note 5: Pressing the STOP key is identical in function to executing STOP or END. Editing or altering a paused program causes the program to go into the stopped state.

Note 6: Caused by sending a non-zero value to CONTROL register 0.

Note 7: This is a "soft reset," which does not include an interface self-test or a reconfiguration of protocol.

Note 8: Set according to the value used in the most recent CONTROL statement directed to Register 3. If there has been no CONTROL 3 statement, the switch settings are used.

Second Byte of Non-ASCII Key Sequences (String)

Holding the CTRL key and pressing a non-ASCII key generates a two-character sequence on the CRT. For example,

CTRL - Clear line

produces the following characters on the CRT:

K%

Non-ASCII keypresses can be simulated by outputting these two-byte sequences to the keyboard. For example,

```
OUTPUT KBD;CHR$(255)&"%";
```

produces the same result as shown above. The decimal value of the first byte is 255 (on some computers this is the "inverse-video" K).

The following table can be used to look up the key that corresponds to the second character of the sequence. (On the small HP 98203A keyboard some non-ASCII keys generate ASCII characters when they are pressed while holding the CTRL key down.)

Normally on an ITF keyboard, f1 corresponds to ON KEY 1 ..., f2 corresponds to ON KEY 2 ..., etc. However, you can use CONTROL KBD,14;1 to change this relationship so that f1 corresponds to ON KEY 0..., f2 corresponds to ON KEY 1, etc.

With 98203 keyboard compatibility (KBD CMODE ON), the ITF keyboard softkeys f1 thru f4, the Menu and System keys, and f5 thru f8 correspond to 98203 softkeys k0 thru k9, respectively. See "Porting to Series 300" chapter of *BASIC Programming Techniques* for further information about this mode.

The terms System and User in the **ITF Key** column refer to the softkey menu which is currently active on an ITF keyboard.

Char.	Val.	ITF Key	98203 Key	Closure Key
space	32	[1]	[1]	
!	33	Shift-Stop	SHIFT-CLR I/O	Yes
"	34	[1]	[1]	
#	35	Shift-Clear line	CLR LN	
$	36	System f7	ANY CHAR	Yes
%	37	Clear line	CLR→END	Yes
&	38	Select[4]	[2]	
'	39	Prev	[2]	Yes
(	40	Shift-Tab	SHIFT-TAB	
)	41	Tab	TAB	
*	42	Insert line	INS LN	Yes
+	43	Insert char	INS CHR	
,	44	Next	[2]	Yes
-	45	Delete char	DEL CHR	
.	46	[2]	[2]	
/	47	Delete line	DEL LN	Yes
0	48	User 3 f8	k0	Yes
1	49	User 1 f1	k1	Yes
2	50	User 1 f2	k2	Yes
3	51	User 1 f3	k3	Yes
4	52	User 1 f4	k4	Yes
5	53	User 1 f5	k5	Yes
6	54	User 1 f6	k6	Yes
7	55	User 1 f7	k7	Yes
8	56	User 1 f8	k8	Yes
9	57	User 2 f1	k9	Yes

[1] These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode**.).

[2] Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error **is not** reported. Instead, the system will perform as much of the indicated action as possible.

[4] These keys have no system meaning, and will BEEP if not trapped by ON KBD.

Char.	Val.	ITF Key	98203 Key	Closure Key
:	58	System Shift-f6 [4]	[2]	
;	59	System Shift-f7 [4]	[2]	
<	60	◄	←	
=	61	Result[3]	RESULT	
>	62	►	→	
?	63	Recall[3] [6]	RECALL	
@	64	Shift-Recall[3] [7]	SHIFT-RECALL	
A	65	System f4	PRT ALL	Yes
B	66	Back space	BACK SPACE	
C	67	System f2	CONTINUE	
D	68	[2]	EDIT	
E	69	Enter [8]	ENTER	Yes
F	70	System f6	DISPLAY FCTNS	Yes
G	71	Shift-►	SHIFT-→	
H	72	Shift-◄	SHIFT-←	
I	73	Break	CLR I/O	
J	74	(Katakana)[2]	(Katakana)[2]	
K	75	Clear display	CLR SCR	Yes
L	76	Graphics [3]	GRAPHICS	Yes
M	77	Alpha [3]	ALPHA	Yes
N	78	Dump Graph [3]	DUMP GRAPHICS	Yes
O	79	Dump Alpha [3] [9]	DUMP ALPHA	Yes

[2] Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error **is not** reported. Instead, the system will perform as much of the indicated action as possible.

[3] This ITF key is located in the System Control Key Group just above the Numeric Keypad Group. Note that these keys have no labels on their keycaps; however, they do have labels on the BASIC keyboard overlay for the ITF keyboard. For information on the BASIC keyboard overlay for the ITF keyboard, read the manual entitled *Installing, Using, and Maintaining the BASIC System.*

[4] These keys have no system meaning, and will BEEP if not trapped by ON KBD.

[6] also System f8

[7] also System Shift-f8

[8] Or Return

[9] Also Print

Char.	Val.	ITF Key	98203 Key	Closure Key
P	80	Stop	PAUSE	Yes
Q	81	[1]	[1]	
R	82	System f3	RUN	Yes
S	83	System f1	STEP	Yes
T	84	Shift-▼	SHIFT-↓	Yes
U	85	Caps	CAPS LOCK	Yes
V	86	▼	↓	Yes
W	87	Shift-▲	SHIFT-↑	Yes
X	88	[2]	EXECUTE	Yes
Y	89	(Roman)[2]	(Roman)[2]	Yes
Z	90	[1]	[1]	
[	91	System f5	CLR TAB	
\	92	↖	[2]	Yes
]	93	System Shift-f5	SET TAB	
^	94	▲	↑	Yes
_	95	System Shift-↖	[2]	Yes
`	96	[1]	[1]	
a	97	User 2 f2	SHIFT-k0	Yes
b	98	User 2 f3	SHIFT-k1	Yes
c	99	User 2 f4	SHIFT-k2	Yes
d	100	User 2 f5	SHIFT-k3	Yes
e	101	User 2 f6	SHIFT-k4	Yes
f	102	User 2 f7	SHIFT-k5	Yes
g	103	User 2 f8	SHIFT-k6	Yes
h	104	User 3 f1	SHIFT-k7	Yes

[1] These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode.**).

[2] Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error **is not** reported. Instead, the system will perform as much of the indicated action as possible.

Char.	Val.	ITF Key	98203 Key	Closure Key
i	105	User 3 f2	SHIFT-k8	Yes
j	106	User 3 f3	SHIFT-k9	Yes
k	107	User 3 f4	[2]	Yes
l	108	User 3 f5	[2]	Yes
m	109	User 3 f6	[2]	Yes
n	110	User 3 f7	[2]	Yes
o	111	System Shift-f1 [4]	[2]	
p	112	System Shift-f2 [4]	[2]	
q	113	System Shift-f3 [4]	[2]	
r	114	System Shift-f4 [4]	[2]	
s	115	User Shift-f1 [4] [5]	[2]	
t	116	User Shift-f2 [4] [5]	[2]	
u	117	User Shift-f3 [4] [5]	[2]	
v	118	User Shift-f4 [4]	[2]	
w	119	User Shift-f5 [4]	[2]	
x	120	User Shift-f6 [4]	[2]	
y	121	User Shift-f7 [4]	[2]	
z	122	User Shift-f8 [4]	[2]	
{	123	User	[2]	Yes
|	124	Menu	[2]	Yes
}	125	System	[2]	Yes
~	126	Shift-Menu	[2]	Yes
▒	127	[1]	[1]	

[1] These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR$(255) in an output to the keyboard, an error is reported (**Error 131 Bad non-alphanumeric keycode**.).

[2] Cannot generate this keycode from this keyboard. If this character is OUTPUT to the keyboard, an error **is not** reported. Instead, the system will perform as much of the indicated action as possible.

[4] These keys have no system meaning, and will BEEP if not trapped by ON KBD.

[5] These keys are also generated by the HP 46060A/B and HP 46095A (HP Mouse devices) buttons unless GRAPHICS INPUT IS is using them.

Selected High-Precision Metric Conversion Factors

English Units	Metric Units	To convert from English to Metric, multiply by:	To convert from Metric to English, multiply by:
Length			
mil	micrometre (micron)	2.54 x 10^{1}☆	3.937 007 874 x 10^{-2}
inch	millimetre	2.54 x 10^{1}☆	3.937 007 874 x 10^{-2}
foot	metre †	3.048 x 10^{-1}☆	3.280 839 895
mile (intl.)	kilometre	1.609 344☆	6.213 711 922 x 10^{-1}
Area			
inch^2	millimetre^2	6.451 6 x 10^{2}☆	1.550 003 100 x 10^{-3}
foot^2	metre^2	9.290 304 x 10^{-2}☆	1.076 391 042 x 10^{1}
mile^2	kilometre^2	2.589 988 110	3.861 021 585 x 10^{-1}
acre (U.S. survey)	hectare	4.046 873 x 10^{-1}	2.471 044
Volume			
inches^3	millimetres^3	1.638 706 4 x 10^{4}☆	6.102 374 409 x 10^{-5}
feet^3	metres^3	2.831 684 659 x 10^{-2}	3.531 466 672 x 10^{1}
ounces (U.S. fluid)	centimetres^3	2.957 353 x 10^{1}	3.381 402 x 10^{-2}
gallon (U.S. fluid)	litre ‡	3.785 412	2.641 721 x 10^{-1}
Mass			
pound (avdp.)	kilogram	4.535 923 7 x 10^{-1}☆	2.204 622 622
ton (short)	ton (metric)	9.071 847 4 x 10^{-1}☆	1.102 311 311
Force			
ounce (force)	dyne	2.780 138 510 x 10^{4}	3.596 943 090 x 10^{-5}
pound (force)	newton	4.448 221 615	2.248 089 431 x 10^{-1}
Pressure			
psi	pascal	6.894 757 293 x 10^{3}	1.450 377 377 x 10^{-4}
inches of Hg (at 32°F)	millibar	3.386 4 x 10^{1}	2.952 9 x 10^{-2}
Energy			
BTU (IST)	Calorie (kg, thermochem.)	2.521 644 007 x 10^{-1}	3.965 666 831
BTU (IST)	watt-hour	2.930 710 702 x 10^{-1}	3.412 141 633
BTU (IST)	joule §	1.055 055 853 x 10^{3}	9.478 171 203 x 10^{-4}
ft•lb	joule	1.355 817 948	7.375 621 493 x 10^{-1}
Power			
BTU (IST)/hr	watt	2.930 710 702 x 10^{-1}	3.412 141 633
horsepower (mechanical)	watt	7.456 998 716 x 10^{2}	1.341 022 090 x 10^{-3}
horsepower (electric)	watt	7.46 x 10^{2}☆	1.340 482 574 x 10^{-3}
ft•lb/s	watt	1.355 817 948	7.375 621 493 x 10^{-1}
Temperature			
°Rankine	kelvin	1.8☆	5.555 555 556 x 10^{-1}
°Fahrenheit	°Celsius	°C=(°F−32) / 1.8☆	°F = (°C x 1.8)+32☆

☆ Exact conversion
† Conversion redefined in 1959
‡ Conversion redefined in 1964
§ Conversion redefined in 1956

Note: The preferred metric unit for force is the newton; for pressure, the pascal; and for energy, the joule.

Prefix	Symbol	Multiplier
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	k	10^{3}
hecto	h	10^{2}
deka	da	10^{1}

Prefix	Symbol	Multiplier
deci	d	10^{-1}
centi	c	10^{-2}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}
atto	a	10^{-18}

Sources

American Society for Testing and Materials (ASTM), "Standard for Metric Practice". Reprinted from Annual Book of ASTM Standards.

U.S. Department of Commerce, National Bureau of Standards, "NBS Guidelines for the Use of the Metric System". Reprinted from Dimensions/NBS. (October 1977).

Notes

Error Messages

1 Missing option or configuration error.

- If a statement requires an option which is not loaded, the option number or option name (see table below) is given along with error 1.
- Error 1 without an option number indicates other configuration errors.

Option Number	Binary	Option Number	Binary
1	BASIC Main	21	CS80
2	GRAPH	22	BUBBLE
3	GRAPHX	23	EPROM
4	IO	24	HP 9885
5	BASIC Main	25	HPIB
6	TRANS	26	FHPIB
7	MAT	27	SERIAL
8	PDEV	28	GPIO
9	XREF	29	BCD
10	KBD	30	DCOMM
11	CLOCK	31-40	Reserved
12	LEX	41	"Unavailable"
13	BASIC Main	42	CRTB
14	MS	43	CRTA
15	SRM	44-45	Reserved
16	Reserved	46	COMPLEX
17	PCIB[1]	47	CRTX
18	KNB2_0	48	EDIT
19	ERR	49	Reserved
20	DISC	50	HFS

[1] This binary is included in the support software for the HP 98647 PC Instruments Interface. It is not supplied with the BASIC 5.0 system.

2 Memory overflow. If you get this error while loading a file, the program is too large for the computer's memory. If the program loads, but you get this error when you press RUN, then the overflow was caused by the variable declarations. Either way, you need to modify the program or add more read/write memory.

3 Line not found in current context. Could be a GOTO or GOSUB that references a non-existent (or deleted) line, or an EDIT command that refers to a non-existent line label.

4 Improper RETURN. Executing a RETURN statement without previously executing an appropriate GOSUB or function call. Also, a RETURN statement in a user-defined function with no value specified.

5 Improper context terminator. You forgot to put an END statement in the program. Also applies to SUBEND and FNEND.

6 Improper FOR...NEXT matching. Executing a NEXT statement without previously executing the matching FOR statement. Indicates improper nesting or overlapping of the loops.

7 Undefined function or subprogram. Attempt to call a SUB or user-defined function that is not in memory. Look out for program lines that assumed an optional CALL.

8 Improper parameter matching. A type mismatch between a pass parameter and a formal parameter of a subprogram.

9 Improper number of parameters. Passing either too few or too many parameters to a subprogram. Applies only to non-optional parameters.

10 String type required. Attempting to return a numeric from a user-defined string function.

11 Numeric type required. Attempting to return a string from a user-defined numeric function.

12 Attempt to redeclare variable. Including the same variable name twice in declarative statements such as DIM or INTEGER.

13 Array dimensions not specified. Using the `(*)` symbol after a variable name when that variable has never been declared as an array.

14 OPTION BASE not allowed here. The OPTION BASE statement must appear before any declarative statements such as DIM or INTEGER. Only one OPTION BASE statement is allowed in one context.

15 Invalid bounds. Attempt to declare an array with more than 32 767 elements or with upper bound less than lower bound.

16 Improper or inconsistent dimensions. Using the wrong number of subscripts when referencing an array element.

17 Subscript out of range. A subscript in an array reference is outside the current bounds of the array.

18 String overflow or substring error. String overflow is an attempt to put too many characters into a string (exceeding dimensioned length). This can happen in an assignment, an ENTER an INPUT, or a READ. A substring error is an attempted violation of the rules for substrings. Watch out for null strings where you weren't expecting them.

19 Improper value or out of range. A value is too large or too small. Applies to items found in a variety of statements. Often occurs when the number builder overflows (or underflows) during an I/O operation.

20 INTEGER overflow. An assignment or result exceeds the range allowed for INTEGER variables. Must be −32 768 thru 32 767.

22 REAL overflow. An assignment or result exceeds the range allowed for REAL variables.

24 Trig argument too large for accurate evaluation. Out-of-range argument for a function such as TAN or LDIR.

25 Magnitude of ASN or ACS argument is greater than 1. Arguments to these functions must be in the range −1 thru +1.

26 Zero to non-positive power. Exponentiation error.

27 Negative base to non-integer power. Exponentiation error.

28 LOG or LGT of a non-positive number.

29 Illegal floating point number. Does not occur as a result of any calculations, but is possible when a FORMAT OFF I/O operation fills a REAL variable with something other than a REAL number.

30 SQR of a negative number.

31 Division (or MOD) by zero.

32 String does not represent a valid number. Attempt to use "non-numeric" characters as an argument for VAL, data for a READ, or in response to an INPUT statement requesting a number.

33 Improper argument for NUM or RPT$. Null string not allowed.

34 Referenced line not an IMAGE statement. A USING clause contains a line identifier, and the line referred to is not an IMAGE statement.

35 Improper image. See IMAGE or the appropriate keyword in the *BASIC Language Reference.*

36 Out of data in READ. A READ statement is expecting more data than is available in the referenced DATA statements. Check for deleted lines, proper OPTION BASE, proper use of RESTORE, or typing errors.

38 TAB or TABXY not allowed here. The tab functions are not allowed in statements that contain a USING clause. TABXY is allowed only in a PRINT statement.

40 Improper REN, COPYLINES, or MOVELINES command. Line numbers must be whole numbers from 1 to 32 766. This may also result from a COPYLINES or MOVELINES statement whose destination line numbers lie within the source range.

41 First line number greater than second line number. Parameters out of order in a statement like SAVE, LIST, or DEL.

43 Matrix must be square. The MAT functions: IDN, INV, and DET require the array to have equal numbers of rows and columns.

44 Result cannot be an operand. Attempt to use a matrix as both result and argument in a MAT TRN or matrix multiplication.

46 Attempting a SAVE when there is no program in memory.

47 COM declarations are inconsistent or incorrect. Includes such things as mismatched dimensions, unspecified dimensions, and blank COM occurring for the first time in a subprogram.

49 Branch destination not found. A statement such as ON ERROR or ON KEY refers to a line that does not exist. Branch destinations must be in the same context as the ON...statement.

51 File not currently assigned. Attempting an ON/OFF END statement with an unassigned I/O path name.

52 Improper mass storage volume specifier. The characters used for a msvs do not form a valid specifier. This could be a missing colon, too many parameters, illegal characters, etc.

53 Improper file name. The file name is too long or has characters that are not allowed. LIF file names are limited to 10 characters; SRM file names to 16 characters; HFS file names to 14 characters for Short File Name systems (SFN) and 255 for Long File Name systems (LFN). Foreign characters are allowed, but punctuation (in commands, etc.) is not.

54 Duplicate file name. The specified file name already exists in directory. It is illegal to have two files with the same name on one LIF volume or in the same SRM or HFS directory.

55 Directory overflow. Although there may be room on the media for the file, there is no room in the directory for another file name. LIF Discs initialized by BASIC have room for over 100 entries in the directory, but other systems may make a directory of a different size.

56 File name is undefined. The specified file name does not exist in the directory. Check the contents of the disc with a CAT command.

58 Improper file type. Many mass storage operations are limited to certain file types. For example, LOAD is limited to PROG files and ASSIGN is limited to ASCII, BDAT, and HP-UX files.

59 End of file, buffer or pipe found. For files: No data left when reading a file, or no space left when writing a file. For buffers: No data left for an ENTER, or no buffer space left for an OUTPUT. Also, WORD-mode TRANSFER terminated with odd number of bytes. For pipes: Inbound pipe was closed.

60 End of record found in random mode. Attempt to ENTER or OUTPUT a field that is larger than a defined record.

62 Protect code violation. Failure to specify the protect code of a protected file, or attempting to protect a file of the wrong type.

64 Mass storage media overflow. The disc is full. (There is not enough free space for the specified file size, or not enough contiguous free space on a LIF disc.)

65 Incorrect data type. The array used in a graphics operation, such as GLOAD, is the wrong type (INTEGER or REAL).

66 INITIALIZE failed. Too many bad tracks found. The disc is defective, damaged, or dirty.

67 Illegal mass storage parameter. A mass storage statement contains a parameter that is out of range, such as a negative record number or an out of range number of records.

68 Syntax error occurred during GET. One or more lines in the file could not be stored as valid program lines. The offending lines are usually listed on the system printer. Also occurs if the first line in the file does not start with a valid line number.

72 Disc controller not found or bad controller address. The msus contains an improper device selector, or no external disc is connected. For BASIC/UX, this can also mean the necessary device file in `/dev/rmb` was not present (LIF only).

73 Improper device type in mass storage volume specifier. The msvs has the correct general form, but the characters used for a device type are not recognized.

76 Incorrect unit number in mass storage volume specifier. The msvs contains a unit number that does not exist on the specified device.

77 Operation not allowed on open file. The specified file is assigned to an I/O path name which has not been closed.

78 Invalid mass storage volume label. Usually indicates that the media has not been initialized on a compatible system. Could also be a bad disc.

79 File open on target device. Attempt to copy an entire volume with a file open on the destination disc.

80 Disc changed or not in drive. Either there is no disc in the drive or the drive door was opened while a file was assigned.

81 Mass storage hardware failure. Also occurs when the disc is pinched and not turning. Try reinserting the disc.

82 Mass storage volume not present. Hardware problem or an attempt to access a left-hand drive on the Model 226.

83 Write protected. Attempting to write to a write-protected disc. This includes many operations such as PURGE, INITIALIZE, CREATE, SAVE, OUTPUT, etc.

84 Record not found. Usually indicates that the media has not been initialized.

85 Media not initialized. (Usually not produced by the internal drive.)

87 Record address error. Usually indicates a problem with the media.

88 Read data error. The media is physically or magnetically damaged, and the data cannot be read.

89 Checkread error. An error was detected when reading the data just written. The media is probably damaged.

90 Mass storage system error. Usually a problem with the hardware or the media. In BASIC/UX, it could also be a problem in the BASIC/UX mas memory system software.

93 Incorrect volume code in msvs. The msvs contains a volume number that does not exist on the specified device.

100 Numeric IMAGE for string item.

101 String IMAGE for numeric item.

102 Numeric field specifier is too large. Specifying more than 256 characters in a numeric field.

103 Item has no corresponding IMAGE. The image specifier has no fields that are used for item processing. Specifiers such as `# X /` are not used to process the data for the item list. Item-processing specifiers include things like `K D B A`.

105 Numeric IMAGE field too small. Not enough characters are specified to represent the number.

106 IMAGE exponent field too small. Not enough exponent characters are specified to represent the number.

107 IMAGE sign specifier missing. Not enough characters are specified to represent the number. Number would fit except for the minus sign.

117 Too many nested structures. The nesting level is too deep for such structures as FOR, SELECT, IF, LOOP, etc.

118 Too many structures in context. Refers to such structures as FOR/NEXT, IF/THEN/ELSE, SELECT/CASE, WHILE, etc.

120 Not allowed while program running. The program must be stopped before you can execute this command.

121 Line not in main program. The run line specified in a LOAD or GET is not in the main context. 122 Program is not continuable. The program is in the stopped state, not the paused state. CONT is allowed only in the paused state.

122 Program is not continuable.

125 Program not running.

126 Quote mark in unquoted string. Quote marks must be used in pairs.

127 Statements which affect the knob mode are out of order.

128 Line too long during GET.

131 Unrecognized non-ASCII keycode. An output to the keyboard contained a CHR$(255) followed by an illegal byte.

132 Keycode buffer overflow. Trying to send too many characters to the keyboard buffer with an OUTPUT 2 statement.

133 DELSUB of non-existent or busy subprogram.

134 Improper SCRATCH statement.

135 READIO/WRITEIO to nonexistent memory location.

136 REAL underflow. The input or result is closer to zero than $10^{/308}$ (approximately).

140 Too many symbols in the program. Symbols are variable names, I/O path names, COM block names, subprogram names, and line identifiers.

141 Variable cannot be allocated. It is already allocated.

142 Variable not allocated. Attempt to DEALLOCATE a variable that was not allocated.

143 Reference to missing OPTIONAL parameter. The subprogram is trying to use an optional parameter that didn't have any value passed to it. Use NPAR to check the number of passed parameters.

145 May not build COM at this time. Attempt to add or change COM when a program is running. For example, a program does a LOADSUB and the COM in the new subprogram does not match existing COM.

146 Duplicate line label in context. There cannot be two lines with the same line label in one context.

150 Illegal interface select code or device selector. Value out of range.

152 Parity error.

153 Insufficient data for ENTER. A statement terminator was received before the variable list was satisfied.

154 String greater than 32 767 bytes in ENTER.

155 Improper interface register number. Value out of range or negative.

156 Illegal expression type in list. For example, trying to ENTER into a constant.

157 No ENTER terminator found. The variable list has been satisfied, but no statement terminator was received in the next 256 characters. The # specifier allows the statement to terminate when the last item is satisfied.

158 Improper image specifier or nesting images more than 8 deep. The characters used for an image specifier are improper or in an improper order.

159 Numeric data not received. When entering characters for a numeric field, an item terminator was encountered before any "numeric" characters were received.

160 Attempt to enter more than 32 767 digits into one number.

163 Interface not present. The intended interface is not present, set to a different select code, or is malfunctioning. In BASIC/UX, the MUX channel is not opened.

164 Illegal BYTE/WORD operation. Attempt to ASSIGN with the WORD attribute to a non-word device.

165 Image specifier greater than dimensioned string length.

167 Interface status error. Exact meaning depends upon the interface type. With HP-IB, this can happen when a non-controller operation by the computer is aborted by the bus.

168 Device timeout occurred and the ON TIMEOUT branch could not be taken.

170 I/O operation not allowed. The I/O statement has the proper form, but its operation is not defined for the specified device. For example, using an HP-IB statement on a non-HP-IB interface or directing a LIST to the keyboard.

171 Illegal I/O addressing sequence. The secondary addressing in a device selector is improper, primary address is too large for specified device, or primary address is required for MUX interface.

172 Peripheral error. PSTS line is false. If used, this means that the peripheral device is down. If PSTS is not being used, this error can be suppressed by using control register 2 of the GPIO.

173 Active or system controller required. The HP-IB is not active controller and needs to be for the specified operation.

174 Nested I/O prohibited. An I/O statement contains a user-defined function. Both the original statement and the function are trying to access the same file or device.

177 Undefined I/O path name. Attempting to use an I/O path name that is not assigned to a device or file.

178 Trailing punctuation in ENTER. The trailing comma or semicolon that is sometimes used at the end of OUTPUT statements is not allowed at the end of ENTER statements.

180 HFS disc may be corrupt.

181 No room in HFS buffers.

182 Not supported by HFS.

183 Permission denied. You have insufficient access rights for the specified operation.

186 Cannot open the specified directory.

187 Cannot link across devices.

188 Renaming using ., .., or / not allowed.

189 Too many open files.

190 File size exceeds the maximum allowed.

191 Too many links to a file.

192 Networking error.

193 Resource deadlock would occur.

194 Operation would block.

195 Too many levels of a symbolic link.

196 Amigo disk file is busy.

197 Incorrect device type in device file.

198 Initialize procedure killed by signal.

301 Cannot do while connected.

303 Not allowed when trace active.

304 Too many characters without terminator.

306 Interface card failure. The datacomm card has failed self-test.

308 Illegal character in data.

310 Not connected.

313 USART receive buffer overflow. Overrun error detected. Interface card is unable to keep up with incoming data rate. Data has been lost.

314 Receive buffer overflow. Program is not accepting data fast enough to keep up with incoming data rate. Data has been lost.

315 Missing data transmit clock. A transmit timeout has occurred because a missing data clock prevented the card from transmitting. The card has disconnected from the line.

316 CTS false too long. The interface card was unable to transmit for a predetermined period of time because Clear-To-Send was false on a half-duplex line. The card has disconnected from the line.

317 Lost carrier disconnect. Data Set Ready (DSR) or Data Carrier Detect (if full duplex) went inactive for too long.

318 No activity disconnect. The card has disconnected from the line because no data was transmitted or received for a predetermined length of time.

319 Connection not established. Data Set Ready or Data Carrier Detect (if full duplex) did not become active within a predetermined length of time.

324 Card trace buffer overflow.

325 Illegal databits/parity combination. Attempting to program 8 bits-per-character and a parity of "1" or "0".

326 Register address out of range. A control or status register access was attempted to a non-existent register.

327 Register value out of range. Attempting to place an illegal value in a control register.

328 USART Transmit underrun.

330 User-defined LEXICAL ORDER IS table size exceeds array size.

331 Repeated value in pointer. A MAT REORDER vector has repeated subscripts. This error is not always caught.

332 Non-existent dimension given. Attempt to specify a non-existent dimension in a MAT REORDER operation.

333 Improper subscript in pointer. A MAT REORDER vector specifies a non-existent subscript.

334 Pointer size is not equal to the number of records. A MAT REORDER vector has a different number of elements than the specified dimension of the array.

335 Pointer is not a vector. Only single-dimension arrays (vectors) can be used as the pointer in a MAT REORDER or a MAT SORT statement.

337 Substring key is out-of-range. The specified substring range of the sort key exceeds the dimensioned length of the elements in the array.

338 Key subscript out-of-range. Attempt to specify a subscript in a sort key outside the current bounds of the array.

340 Mode table too long. User-defined LEXICAL ORDER IS mode table contains more than 63 entries.

341 Improper mode indicator. User-defined LEXICAL ORDER IS table contains an illegal combination of mode type and mode pointer.

342 Not a single-dimension integer array. User-defined LEXICAL ORDER IS mode table must be a single-dimension array of type INTEGER.

343 Mode pointer is out of range. User-defined LEXICAL ORDER IS table has a mode pointer greater than the existing mode table size.

344 1 for 2 list empty or too long. A user-defined LEXICAL ORDER IS table contains an entry indicating an improper number of 1 for 2 secondaries.

345 CASE expression type mismatch. The SELECT statement and its CASE statements must refer to the same general type, numeric or string.

346 INDENT parameter out-of-range. The parameters must be in the range: 0 thru eight characters less than the screen width.

347 Structures improperly matched. There is not a corresponding number of structure beginnings and endings. Usually means that you forgot a statement such as END IF, NEXT, END SELECT, etc.

349 CSUB has been modified. A contiguous block of compiled subroutines has been modified since it was loaded. A single module that shows as multiple CSUB statements has been altered because program lines were inserted or deleted.

353 Data link failure.

369-398 Errors in this range are reported if a run-time Pascal error occurs in a CSUB. To determine the Pascal error number. subtract 400 from the BASIC error number. Information on the Pascal error can be found in the *Pascal Workstation System* manual.

401 Bad system function argument. An invalid argument was given to a time, date, base conversion, or SYSTEM$ function.

403 Copy failed; program modification incomplete. An error occurred during a COPYLINES or MOVELINES resulting in an incomplete operation. (Some lines may not have been copied or moved.)

427 Priority may not be lowered.

435 EXEC not allowed on this Binary.

450 Volume not found—SRM error.

451 Volume labels do not match—SRM error.

453 File in use—SRM error.

454 Directory formats do not match—SRM error.

455 Possibly corrupt file—SRM error.

456 Unsupported directory operation—SRM or HFS error.

457 Passwords not supported—SRM error.

458 Unsupported directory format—SRM error.

459 Specified file is not a directory—SRM or HFS error.

460 Directory not empty—SRM or HFS error.

461 Duplicate passwords not allowed.

462 Invalid password—SRM error.

465 Invalid rename across volumes—SRM or HFS error.

466 Duplicate volume entries.

471 TRANSFER not supported by the interface.

481 File locked or open exclusively—SRM or HFS error.

482 Cannot move a directory with a RENAME operation—SRM or HFS error.

483 System down—SRM error.

484 Password not found—SRM error.

485 Invalid volume copy—SRM error.

488 DMA hardware required. HP 9885 disc drive requires a DMA card or is malfunctioning.

511 The result array in a MAT INV must be of type REAL.

516 Search key: improper dimensions.

517 Search start out of range.

519 HIL SEND Cmd arg out of range.

520 Cmd not supported on active dev.

521 Device sent Register I/O Error.

522 Device not present.

523 Statement requires HIL interface.

526 Source: improper dimensions.

527 Source subscript out of range.

528 Source: upper bound < lower bound.

531 Source/destination mismatch.

536 Dest.: improper dimensions.

537 Dest. subscript out of range.

538 Dest. upper bound < lower bound.

540 HIL bus error.

541 Keyboard interrupts disabled.
Operation requires bit 0 of KBD STATUS/CONTROL register 7 to be 0.

543 Redim error: improper dimensions.

544 Redim not allowed on source.

600 Attribute cannot be modified. The WORD/BYTE mode cannot be changed after assigning the I/O path name.

601 Improper CONVERT lifetime. When the CONVERT attribute is included in the assignment of an I/O path name, the name of a string variable containing the conversion is also specified. The conversion string must exist as long as the I/O path name is valid.

602 Improper BUFFER lifetime. The variable designated as a buffer during an I/O path name assignment must exist as long as the I/O path name is valid.

603 Variable was not declared as a BUFFER. Attempt to assign a variable as a buffer without first declaring the variable as a BUFFER.

604 Bad source or destination for a TRANSFER statement. Transfers are not allowed to the CRT, keyboard, or tape backup on CS80 drives. Buffer to buffer or device to device transfers are not allowed.

605 BDAT or HP-UX file type required. Only a BDAT or HP-UX file can be used in a TRANSFER operation.

606 Improper TRANSFER parameters. Conflicting or invalid TRANSFER parameters were specified, such as RECORDS without and EOR clause, or DELIM with an outbound TRANSFER.

607 Inconsistent attributes. Such as CONVERT or PARITY with FORMAT OFF.

609 IVAL or DVAL result too large. Attempt to convert a binary, octal, decimal, or hexadecimal string into a value outside the range of the function.

610 Premature TRANSFER termination.

612 BUFFER pointers in use. Attempt to change one or more buffer pointers while a TRANSFER is in progress.

613 Cannot store a ROM system.

620 COMPLEX value not allowed.

623 ATN is undefined at +i and -i.

624 ACSH/ATNH arg out of range.

625 Bad SEARCH condition on COMPLEX.

700 Improper plotter specifier. The characters used as a plotter specifier are not recognized. May be misspelled or contain illegal characters.

702 CRT graphics hardware missing. Hardware problem.

704 Upper bound not greater than lower bound. Applies to P2<=P1 or VIEWPORT upper bound and CLIP limits. 705 VIEWPORT or CLIP beyond hard clip limits.

705 VIEWPORT or CLIP off surface.

708 Device not initialized.

713	Request not supported by dev.
715	Graphics open failed on CRT device.
733	GESCAPE opcode not recognized.
810	Feature not supported on HP-UX.
811	Memory allocation failed.
812	Out of semaphores.
813	Semaphore deallocation error.
814	Cannot access rmb lockfile.
815	Canot access HP-UX time.
816	Invalid opcode in program.
817	Cannot spawn new process.
818	Kernel error setting signals.
825	Default EXT SIGNAL received.
826	EXECUTE process status failure.
827	String too long for EXECUTE.
830	Cannot open the pipe.
831	Write to a broken pipe.
832	Cannot seek on the pipe.
833	Wrong directory data transfer in pipe.
840	HIL mask error.
841	CSUB run-time error.
842	CSUB relocatio error.
843	Invalid CSUB version number.
844	Invalid CSUB binary format.
850	Iomap of device failed.
851	Iounmap of device failed.
852	Iomap device file size wrong.

860 Unknown display type.

861 PRINTER IS device not assigned.

862 Window parameter out of range.

863 Not in a window system.

864 Window specifier out of range.

865 Window already exists.

866 Window does not exist.

867 Cannot create window.

880 Current CRT is not bitmapped.

881 Array is not INTEGER type.

882 CHRX not matched by array dim.

883 CHRY not matched by array dim.

897 Array is not 1-dimensional.

898 Typing aid is too long.

899 Key number out of range.

900 Undefined typing aid key.

901 Typing aid memory overflow.

902 Must delete entire context. Attempt to delete a SUB or DEF FN statement without deleting its entire context. Easiest way to delete is with DELSUB.

903 No room to renumber. While EDIT mode was renumbering during an insert, all available line numbers were used between insert location and end of program.

904 Null FIND or CHANGE string.

905 CHANGE would produce a line too long for the system. Maximum line length is two lines on the CRT.

906 SUB or DEF FN not allowed here. Attempt to insert a SUB or DEF FN statement into the middle of a context. Subprograms must be appended at the end.

909 May not replace SUB or DEF FN. Similar to deleting a SUB or DEF FN. Attempted to insert lines: between a CSUB statement and the following SUB, DEF FN, or CSUB statement; or after a final CSUB statement at the end of the program.

910 Identifier not found in this context. The keyboard-specified variable does not already exist in the program. Variables cannot be created from the keyboard; they must be created by running a program.

911 Improper I/O list.

920 Numeric constant not allowed.

921 Numeric identifier not allowed.

922 Numeric array element not allowed.

923 Numeric expression not allowed.

924 Quoted string not allowed.

925 String identifier not allowed.

926 String array element not allowed.

927 Substring not allowed.

928 String expression not allowed.

929 I/O path name not allowed.

930 Numeric array not allowed.

931 String array not allowed.

932 Excess keys specified. A sort key was specified following a key which specified the entire record.

935 Identifier is too long: 15 characters maximum.

936 Unrecognized character. Attempt to store a program line containing an improper name or illegal character.

937 Invalid OPTION BASE. Only 0 and 1 are allowed.

939 OPTIONAL appears twice. A parameter list may have only one OPTIONAL keyword. All parameters listed before it are required, all listed after it are optional. 940 Duplicate formal parameter name.

940 Duplicate formal param name.

942 Invalid I/O path name. The characters after the `@` are not a valid name. Names must start with a letter.

943 Invalid function name. The characters after the `FN` are not a valid name. Names must start with a letter.

946 Dimensions are inconsistent with previous declaration. The references to an array contain a different number of subscripts at different places in the program.

947 Invalid array bounds. Value out of range, or more than 32 767 elements specified.

948 Multiple assignment prohibited. You cannot assign the same value to multiple variables by stating `X=Y=Z=0`. A separate assignment must be made for each variable.

949 Syntax error at cursor. The statement you typed contains elements that don't belong together, are in the wrong order, or are misspelled.

950 Must be a positive integer.

951 Incomplete statement. This keyword must be followed by other items to make a valid statement.

954 Improper default specification.

955 No range given.

956 Source/destination mismatch.

961 CASE expression type mismatch. The CASE line contains items that are not the same general type, numeric or string.

962 Programmable only: cannot be executed from the keyboard.

963 Command only: cannot be stored as a program line.

977 Statement is too complex. Contains too many operators and functions. Break the expression down so that it is performed by two or more program lines.

980 Too many symbols in this context. Symbols include variable names, I/O path names, COM block names, subprogram names, and line identifiers.

982 Too many subscripts: maximum of six dimensions allowed.

983 Wrong type or number of parameters. An improper parameter list for a machine-resident function.

985 Invalid quoted string.

987 Invalid line number: must be a whole number 1 thru 32 766.

Notes

Table of Contents

Appendix F: Keyword Summary

F

Keyword Summary

The following sections summarize the BASIC keywords by categories.

Booting the System

LIST BIN	Lists binaries in the system.
LOAD BIN	Loads a BIN-type file into memory (BASIC Workstation only).
SYSBOOT	Returns system control to the boot ROM (BASIC Workstation only).
rmb	(HP-UX command) enters BASIC/UX from HP-UX.
QUIT	Exits BASIC/UX and returns to HP-UX.

Program Entry/Editing

CHANGE	Performs search and replace operations on the program in memory.
COPYLINES	Copies program lines from one position to another.
EDIT	Accesses a program using edit mode to enter new program lines or modify existing ones. Also used with typing-aid softkeys.
FIND	Searches for a character sequence in a program.
DEL	Deletes specified program lines from memory.
INDENT	Indents a program to reflect its structure.
LIST	Lists program lines or typing-aid softkeys.
MOVELINES	Moves program lines from one position to another.
REM and !	Allows comments on program lines.
REN	Renumbers programs.
SECURE	Makes program lines unlistable.

Program Debugging and Error Handling

CAUSE ERROR Simulates the occurrence of the BASIC error of the specified number.

CLEAR ERROR Resets most error indicators (ERRN, ERRLN, ERRM$, and ERRL) to their power-up state.

ERRDS Returns the device selector involved in the last I/O error.

ERRL Indicates whether an error occurred during execution of a specified line.

ERRLN Returns the program-line number of the most recent error.

ERRM$ Returns the text of the last error message.

ERRN Returns the most recent program execution error.

ERROR RETURN Returns program control to the line following the line which caused the most recent GOSUB. Used with ON ERROR GOSUB to avoid retrying the line that caused the error (use RETURN to return control to the line which caused the error).

ERROR SUBEXIT Returns program control to the line following the line which caused the most recent CALL. Used with ON ERROR CALL to avoid retrying the line that caused the error (use SUBEXIT to return control to the line which caused the error).

TRACE ALL Allows tracing of program flow and variable assignments during program execution.

TRACE PAUSE Causes program execution to pause at a specified line.

TRACE OFF Disables TRACE ALL and TRACE PAUSE.

XREF Provides a cross-reference to all identifiers used in a program.

Memory Allocation and Management

ALLOCATE	Dimensions and allocates memory for arrays or string variables during program execution.
COM	Dimensions and reserves memory for variables in a common area for access by more than one context.
COMPLEX	Dimensions and reserves memory for complex variables and arrays.
DEALLOCATE	Reclaims memory previously allocated.
DELSUB	Deletes specified subprograms from memory.
DIM	Dimensions and reserves memory for REAL numeric arrays and strings.
INITIALIZE	Creates and deletes RAM mass storage volumes. (See also under "Mass Storage.")
INTEGER	Dimensions and reserves memory for INTEGER variables and arrays.
LOADSUB	Loads BASIC subprograms from a PROG-type file into memory.
OPTION BASE	Specifies the default lower bound for arrays.
REAL	Dimensions and reserves memory for full-precision (REAL) variables and arrays.
SCRATCH	Erases selected portions of memory.

Comparison Operators

=	Equality.
< >	Inequality.
<	Less than.
<=	Less than or equal to.
>	Greater than.
>=	Greater than or equal to.

General Math

+	Addition operator.
–	Subtraction operator.
*	Multiplication operator.
/	Division operator.
^	Exponentiation operator.
ABS	Returns an argument's absolute value.
DIV	Divides one argument by another and returns the integer portion of the quotient.
DROUND	Returns the value of an expression, rounded to a specified number of digits.
EXP	Raises the base e to a specified power.
FRACT	Returns the fractional portion of an expression.
INT	Returns the integer portion of an expression.
LET	Assigns values to variables.
LGT	Returns the log (base 10) of an argument.
LOG	Returns the natural logarithm (base e) of an argument.
MAX	Returns the largest value in a list of arguments.
MAXREAL	Returns the largest number available.
MIN	Returns the smallest value in a list of arguments.
MINREAL	Returns the smallest number available.
MOD	Returns the remainder of integer division.
MODULO	Return the modulo of division.
PI	Returns an approximation of π.
PROUND	Returns the value of an expression, rounded to the specified power of ten.
RANDOMIZE	Modifies the seed used by the RND function.
RES	Returns last live keyboard numeric result.
RND	Returns a pseudo-random number.

SGN	Returns the sign of an argument.
SQRT	Returns the square root of an argument (same as SQR).
SQR	Returns the square root of an argument (same as SQRT).

Complex Math

ARG	Returns the argument (or the angle in polar coordinates) of a COMPLEX value.
CMPLX	Creates a COMPLEX value, given a real and an imaginary part.
CONJG	Returns the conjugate of a COMPLEX value (negates imaginary part).
IMAG	Returns the imaginary part of a COMPLEX value.
REAL	Returns the real part of a COMPLEX value.

Binary Functions

BINAND	Returns the bit-by-bit logical-and of two arguments.
BINCMP	Returns the bit-by-bit complement of an argument.
BINEOR	Returns the bit-by-bit exclusive-or of two arguments.
BINIOR	Returns the bit-by-bit inclusive-or of two arguments.
BIT	Returns the state of a specified bit of an argument.
ROTATE	Returns a value obtained by shifting an argument's binary representation a number of bit positions, with wrap-around.
SHIFT	Returns a value obtained by shifting an argument's binary representation a number of bit positions, without wrap-around.

Trigonometric Operations

ACS	Returns the arccosine of an argument.
ASN	Returns the arcsine of an argument.
ATN	Returns the arctangent of an argument.
COS	Returns the cosine of an angle.
DEG	Sets the degrees mode.
RAD	Sets the radians mode.
SIN	Returns the sine of an angle.
TAN	Returns the tangent of an angle.

Hyperbolic Operations

ACSH	Returns the hyperbolic arc cosine of a numeric expression.
ASNH	Returns the hyperbolic arcsine of a numeric expression.
ATNH	Returns the hyperbolic arctangent of a numeric expression.
COSH	Returns the hyperbolic cosine of a numeric expression.
SINH	Returns the hyperbolic sine of a numeric expression.
TANH	Returns the hyperbolic tangent of a numeric expression.

String Operations

&	Concatenates two string expressions.
CHR$	Converts a numeric value into an ASCII character.
DVAL	Converts an alternate-base representation into a numeric value.
DVAL$	Converts a numeric value into an alternate-base representation.
IVAL	Converts an alternate-base representation into an INTEGER number.
IVAL$	Converts an INTEGER into an alternate-base representation.
LEN	Returns the number of characters in a string expression.
LEXICAL ORDER IS	Determines the collating sequence used in string comparisons.
LWC$	Returns the lowercase value of a string expression.

MAXLEN	Returns the maximum (dimensioned) length of a string variable.
NUM	Returns the decimal value of the first character in a string.
POS	Returns the position of a string within a string expression.
REV$	Reverses the order of the characters in a string expression.
RPT$	Repeats the characters in a string expression a specified number of times.
TRIM$	Removes the leading and trailing blanks from a string expression.
UPC$	Returns the uppercase value of a string expression.
VAL	Converts a string of numerals into a numeric value.
VAL$	Returns a string expression representing a specified numeric value.

Logical Operators

AND	Returns 1 or 0 based on the logical AND of two arguments.
EXOR	Returns 1 or 0 based on the logical exclusive-or of two arguments.
NOT	Returns 1 or 0 based on the logical complement of an argument.
OR	Returns 1 or 0 based on the logical inclusive-or of two arguments.

Mass Storage

ASSIGN	Assigns an I/O path name and attributes to a file.
CAT	Lists the contents of the mass storage media's directory.
CHECKREAD	Enables or disables read-after-write verification of mass storage operations.
CHGRP	Changes the group id of an HFS file or directory.
CHOWN	Changes the ownership of an HFS file or directory.
COPY	Provides a method of copying mass storage files and volumes.
CREATE	Creates an HP-UX-type file on a mass storage media.
CREATE ASCII	Creates an ASCII-type file on a mass storage media.
CREATE BDAT	Creates a BDAT-type file on a mass storage media.
CREATE DIR	Creates a directory on a mass storage media.

GET	Reads an ASCII or HP-UX file into memory as a program.
INITIALIZE	Formats a mass storage media for use with BASIC and places a LIF directory on the media.
LINK	Allows the linking of two file names to the same file.
LOAD	Loads a PROG-type file into memory.
LOAD KEY	Loads typing-aid softkey definitions.
LOADSUB	Loads BASIC subprograms from a PROG-type file into memory.
LOCK	Prevents other SRM workstation computers from accessing the file to which the specified I/O path is currently assigned.
MASS STORAGE IS or MSI	Specifies the default mass storage device.
PERMIT	Changes the access permission bits on an HFS file or directory.
PRINT LABEL	Writes a string expression to the label of a media.
PROTECT	Specifies a LIF protect code or a password for an SRM file or directory.
PURGE	Deletes a file or directory.
READ LABEL	Reads the label of a media to a string variable.
RENAME	Changes a directory's name or file's name and/or path.
SAVE and RE-SAVE	Create an ASCII file and write BASIC program lines as strings into the file. RE-SAVE can write to an existing HP-UX file.
STORE and RE-STORE	Create a PROG file and write a BASIC program from memory into the file in an internal format.
STORE KEY and RE-STORE KEY	Create a BDAT file and store the typing-aid softkey definitions in the file.
STORE SYSTEM	Stores BASIC and all binaries currently in memory in a SYSTM file on LIF and SRM. On HFS, it is an HP-UX file.
UNLOCK	Removes exclusive access to an SRM file set by the LOCK statement.

Program Control

CALL	Transfers program execution to a specified subprogram and passes parameters.
CONT	Resumes execution of a paused program.
DEF FN	Defines the beginning of a function subprogram.
FNEND	Defines the bounds of a user-defined function subprogram.
END	Terminates program execution and marks the end of the main program segment.
FN	Invokes a user-defined function.
FOR...NEXT	Defines a loop which is repeated a specified number of times.
GOTO	Transfers program execution to a specified line.
GOSUB	Transfers program execution to a specified subroutine.
IF...THEN	Provides conditional branching.
ELSE	Provides a conditional execution of a program segment.
LOOP	Defines a loop which is repeated until the expression in an EXIT IF statement is evaluated as true.
EXIT IF	Provides looping with conditional exit.
NPAR	Returns the number of parameters passed to the current subprogram.
ON	expression Transfers program execution to one of several locations based on the value of an expression.
PAUSE	Suspends program execution.
REPEAT...UNTIL	Allows execution of a program segment until the specified condition is true.
RETURN	Transfers program execution from a subroutine to the line following the invoking GOSUB.
RETURN	expression Transfers program execution from a user-defined function by returning a value to the calling context.
RUN	Starts program execution.
SELECT...CASE	Allows execution of one program segment of several.

STOP Terminates execution of the program.

SUB Defines the beginning of a SUB subprogram and specifies its formal parameters.

SUBEND Defines the bounds of a subprogram.

SUBEXIT Transfers control from within a subprogram to the calling context.

SUSPEND/ RESUME INTERACTIVE Allows suspending and resuming interactive keyboard operation while a program is running.

SYSTEM$ Returns selected system status and configuration information.

WAIT Causes program execution to wait a specified number of seconds.

WAIT FOR EOR Causes program execution to wait for an end-of-record during a TRANSFER.

WAIT FOR EOT Causes program execution to wait for an end-of-transfer.

WHILE Allows execution of a program segment while the specified condition is true.

Event-Initiated Branching

CDIAL Returns information about “control dial” devices.

DISABLE Disables event-initiated branching (except for ON END, ON ERROR, and ON TIMEOUT).

DISABLE EXT SIGNAL Disable BASIC/UX handling of HP-UX signals.

DISABLE INTR Disables interrupts defined by the ON INTR statement.

ENABLE Re-enables all event-initiated branches previously suspended by DISABLE.

ENABLE EXT SIGNAL Enable BASIC/UX handling of HP-UX signals.

ENABLE INTR Enables the specified interface to generate an interrupt which can cause event-initialted branches.

EXECUTE Execute an HP-UX command from BASIC/UX.

HILBUF$ Returns data sent by an HP-HIL device.

KBD$ Returns the contents of the ON KBD buffer.

KNOBX Returns the number of horizontal knob pulses.

KNOBY Returns the number of vertical knob pulses.

ON CDIAL Sets up and enables a branch to be taken upon sensing rotation of one of the dials on a "control dial" device.

OFF CDIAL Disables any ON CDIAL branching currently set up.

ON CYCLE Defines and enables an event-initiated branch to be taken each time the specified number of seconds has elapsed.

OFF CYCLE Cancels any event-initiated branches previously defined and enabled by an ON CYCLE statement.

ON DELAY Defines an enables an event-initiated branch to be taken after the specified number of seconds has elapsed.

OFF DELAY Cancels any event-initiated branches previously defined and enabled by an ON DELAY statement.

ON END Defines and enables an event-initiated branch to be taken when end-of-file is reached on the mass storage file associated with the specified I/O path.

OFF END Cancels any event-initiated branches previously defined and enabled by an ON END statement.

ON EOR Defines and enables an event-initiated branch to be taken when an end-of-record is encountered during a TRANSFER.

OFF EOR Cancels any event-initiated branches previously defined and enabled by an ON EOR statement.

ON EOT Defines and enables an event-initiated branch to be taken when the last byte is tranferred by a TRANSFER statement.

OFF EOT Cancels any event-initiated branches previously defined and enabled by an ON EOT statement.

ON ERROR Defines and enables an event-initiated branch which results from a trappable error.

OFF ERROR Cancels any event-initiated branches previously defined and enabled by an ON ERROR statement. Further errors are reported to the user in the usual fashion.

ON EXT SIGNAL Defines an event-initiated branch to be taken when a system generated signal is received (BASIC/UX only).

OFF EXT SIGNAL Cancels event-initiated branches previously defined by an ON EXT SIGNAL statement (BASIC/UX only).

ON HIL EXT Enables an end-of-line interrupt in response to receiving data from HIL devices whose poll records are not otherwise being processed by the BASIC system.

OFF HIL EXT Cancels any event-initiated branches previously defined and enabled by an ON HIL EXT statement.

ON INTR Defines an event-initiated branch to be taken when an interface card generates an interrupt.

OFF INTR Cancels any event-initiated branches previously defined and enabled by an ON INTR statement.

ON KBD Defines an event-initiated branch to be taken when a key is pressed.

OFF KBD Cancels any event-initiated branches previously defined and enabled by an ON KBD statement.

ON KEY...LABEL Defines and enables an event-initiated branch to be taken when a softkey is pressed.

OFF KEY Cancels any event-initiated branches previously defined and enabled by an ON KEY statement.

ON KNOB Defines an event-initiated branch to be taken when the knob is turned.

OFF KNOB Cancels any event-initiated branches previously defined and enabled by an ON KNOB statement. Any pending ON KNOB branches are lost. Further use of the knob will result in normal scrolling or cursor movement.

ON SIGNAL Defines an event-initiated branch to be taken when a SIGNAL statement is executed using the same signal selector.

OFF SIGNAL Cancels the ON SIGNAL definition with the same signal selector. If no signal selector is provided, all ON SIGNAL definitions are cancelled. OFF SIGNAL only applies to the current context.

ON TIME	Defines an event-initiated branch to be taken when the clock reaches a specified time.
OFF TIME	Cancels any event-initiated branches previously defined and enabled by an ON TIME statement.
ON TIMEOUT	Defines an event-initiated branch to be taken when an I/O timeout occurs on the specified interface.
OFF TIMEOUT	Cancels any event-initiated branches previously defined and enabled by an ON TIMEOUT statement.
SET HIL MASK	Select HIL devices to be used by BASIC/UX processes.
SIGNAL	Generates a software interrupt.
SYSTEM PRIORITY	Sets a minimum level of system priority for event-initiated branches.

HP-HIL Device Support

HIL SEND	Sends HP-HIL commands to HP-HIL devices.

See also ON/OFF CDIAL, CDIAL, ON/OFF HIL EXT, HILBUF$, ON/OFF KNOB, KNOBX, KNOBY, in the preceding "Event-Initiated Branching" section.

Graphics Control

ALPHA ON/OFF	Turns the alpha planes on or off.
AREA	Selects an area fill color.
CLIP	Redefines a soft-clip area.
DIGITIZE	Inputs the coordinates of a digitized point.
DUMP GRAPHICS	Copies the contents of the graphics display to a printing device.
DUMP DEVICE IS	Specifies the device for DUMP operations.
GCLEAR	Clears the graphics area.
GESCAPE	ends and returns device-dependent graphics information.
GINIT	Resets graphics parameters to power-on values.
GLOAD	Loads the graphics display from an INTEGER array.
GRAPHICS ON/OFF	Turns the graphics planes on or off.

GRAPHICS INPUT IS	Specifies the device for digitizing operations.
GSEND	Sends an HPGL command to the current PLOTTER IS device or file.
GSTORE	Copies the contents of the graphics display to an INTEGER array.
PLOTTER IS	Specifies the default plotting device or file.
RATIO	Returns the physical aspect ratio of the plotter's hard-clip limits.
READ LOCATOR	Samples the locator device, without waiting for a digitize signal.
SET ECHO	Specifies the coordinates of an echo on the current plotting device.
SET LOCATOR	Sets the locator position on the input device.
SET PEN	Defines the color of entries in the color map.
SHOW	Defines plotting units that will appear in the VIEWPORT area.
TRACK...ON/OFF	Enables and disables locator tracking on the current display device.
VIEWPORT	Specifies an area in which WINDOW and SHOW statements are mapped.
WHERE	Returns the current logical position of the pen.
WINDOW	Specifies the min and max values for the plotting area specified by VIEWPORT.

Graphics Plotting

DRAW	Draws a line to a specified point.
IDRAW	Draws a line incrementally to a specified point.
IMOVE	Moves the pen incrementally to a specified point.
IPLOT	Draws a line incrementally to the specified point with optional pen control.
LINE TYPE	Selects a plotting line type.
MOVE	Moves the pen to a specified point.
PDIR	Specifies rotation for IPLOT, RPLOT, RECTANGLE, POLYGON and POLYLINE.
PEN	Selects a plotter pen.
PENUP	Lifts the pen from the plotting surface.
PIVOT	Specifies rotation for lines made with moves, draws, plots, polygons, or rectangles.
PLOT	Draws a line to the specified point with optional pen control.
POLYGON	Draws all or part of a closed polygon.
POLYLINE	Draws all or part of an open polygon.
RECTANGLE	Draws a rectangle that can be filled and edged.
RPLOT	Draws a line relative to a movable origin with optional pen control.

Graphic Axes and Labeling

AXES	Draws axes with optional tick marks.
CSIZE	Sets the size and aspect ratio for labeled characters.
FRAME	Draws a frame around the current clipping area.
GRID	Draws a full grid pattern for axes.
LABEL	Draws alphanumeric labels.
LDIR	Defines the angle for drawing labels.
LORG	Specifies a labeling location relative to the pen location.
SYMBOL	Allows labeling with user-defined symbols.

HP-IB Control

ABORT	Terminates bus activity and asserts IFC.
CLEAR	Places specified devices in a device-dependent state.
LOCAL	Returns specified devices to their local state.
LOCAL LOCKOUT	Sends the LLO message, disabling all device's front-panel controls.
PASS CONTROL	Passes Active Controller capability to another device.
PPOLL	Returns a parallel poll byte from the bus.
PPOLL CONFIGURE	Programs a parallel poll bit for a specified device.
PPOLL RESPONSE	Defines the computers response to a parallel poll.
PPOLL UNCONFIGURE	Disables parallel poll for specified devices.
REMOTE	Sets specified devices to their remote state.
REQUEST	Sends a service request to the Active Controller.
SEND	Sends explicit command and data messages on the bus.
SPOLL	Returns a serial poll byte from a specified device.
TRIGGER	Sends the trigger message to specified devices.

Clock and Calendar

DATE	Converts a formatted date into a number of seconds.
DATE$	Converts a number of seconds into a formatted date.
SET TIME	Sets the time of day on the real-time clock.
SET TIMEDATE	Sets the time and date on the real-time clock.
TIME	Converts a formatted time of day into a number of seconds past midnight.
TIME$	Converts a number of seconds past midnight into a formatted time of day.
TIMEDATE	Returns the value of the real-time clock.
TIMEZONE	IS Specifies the clock offset from Greenwich Mean Time (GMT), which is used when sharing a disc with an HP-UX system.

General Device Input/Output

ABORTIO	Terminates an active TRANSFER.
ASSIGN	Associates an I/O path name and attributes with a device, group of devices, mass storage file, or buffer.
BEEP	Produces one of 63 audible tones.
BREAK	Sends a Break signal on a serial interface.
CONTROL	Sends control information to an interface or a table associated with an I/O path name.
CRT	Returns the device selector of the CRT.
DATA	Specifies data accessible via READ statements.
DISP	Outputs items to the CRT display line.
DUMP ALPHA	Transfers alpha contents of the CRT to a specified device.
DUMP DEVICE IS	Specifies a device for DUMP ALPHA and DUMP GRAPHICS operations.
ENTER	Inputs data from a device, file, string, or buffer to a list of variables.
IMAGE	Provides formats for use with ENTER, OUTPUT, DISP, LABEL and PRINT operations.
INPUT	Inputs data from the keyboard to a list of variables.
KBD	Returns the device selector of the keyboard.
LINPUT	Inputs literal data from the keyboard to a string variable.
OUTPUT	Outputs items to a specified device, file, string variable, or buffer.
PRINT	Outputs items to the current PRINTER IS device.
PRINTALL IS	Specifies a device for logging messages normally sent to the display.
PRINTER IS	Specifies a device for PRINT, CAT, and LIST statements.
PRT	Returns 701, usually the device selector of an external printer.
READ	Inputs data from DATA lists to variables.
READIO	Reads the contents of the specified hardware registers on the specified interface, or reads the contents of the specified memory address.

RESET Resets an interface or pointers of an I/O path.

RESTORE Causes a READ statement to access the specified DATA statement.

SC Returns the interface select code associated with an I/O path.

SOUND Produces a single tone or multiple tones on the sound generator of an HP-HIL interface.

STATUS Returns the value from a specified interface status register.

TAB Moves the print position ahead to a specified point; used within PRINT and DISP statements.

TABXY Specifies the print position on the internal CRT; used with PRINT statements.

TRANSFER Initiates unformatted I/O transfers.

WRITEIO Writes an integer representation of the register data to the specified hardware register on the specified interface or to the specified memory address.

Display and Keyboard Control

ALPHA HEIGHT Sets the number of display lines used for alpha output.

ALPHA PEN Selects the pen number to be used for displaying alpha.

CHRX Returns the number of pixel columns in an alpha character cell on a bit-mapped display.

CHRY Returns the number of pixel rows in an alpha character cell on a bit-mapped display.

CLEAR LINE Clears the keyboard input line of the display.

CLEAR SCREEN Clears the display screen.

CLEAR WINDOW Clear the contents of a window (BASIC/UX only).

CLS Clears the display screen.

CREATE WINDOW Create a window to be accessed by BASIC/UX.

CRT Returns 1, which is the select code of the CRT display.

DESTROY WINDOW Delete a window created by BASIC/UX.

DISPLAY FUNCTIONS ON/OFF	Enables and disables the "display functions" mode.
KBD	Returns 2, which is the select code of the keyboard.
KBD CMODE	Enables and disables the "98203 Keyboard Compatibility Mode."
KBD LINE PEN	Selects the pen number to be used for writing alpha characters on the "keyboard input line" and associated display areas.
KEY LABELS	Turns softkey labels on and off.
KEY LABELS PEN	Selects the pen number to be used for displaying softkey labels.
LIST WINDOW	List all active BASIC/UX windows and their attributes.
MERGE ALPHA	Joins the "simulated" separate alpha and graphics rasters set up by SEPARATE ALPHA FROM GRAPHICS.
MOVE WINDOW	Move a text or graphics window created by BASIC/UX.
PRINT PEN	Selects the pen number to be used for the output area and display line of the alpha display.
SCRATCH WINDOW	Delete all active BASIC/UX windows except the root BASIC/UX window.
SEPARATE ALPHA	Simulates the separate alpha and graphics rasters of Series 200 displays.
SET ALPHA MASK	Specifies which display planes can be modified by alpha display operations.
SET CHR	Re-defines the bit-pattern used by alpha character(s); only available on bit-mapped alpha displays.
SET DISPLAY MASK	Specifies which planes of the alpha display are to be displayed.
SET KEY	Sets the definition of one or more typing-aid softkeys.
SYSTEM KEYS	Sets the softkey definitions to the System menu (ITF keyboards only).
USER n KEYS	Sets the softkey definitions to the specified User menu (ITF keyboards only).

See also CONTROL, DISP, DUMP ALPHA, DUMP DEVICE IS, ENTER, IMAGE, INPUT, LINPUT, OUTPUT, PRINT, PRINTALL IS, PRINTER IS, STATUS, TAB, and TABXY in the preceding "I/O Operations" section.

Array Operations

BASE	Returns the lower bound of a dimension of an array.
DET	Returns the determinant of a matrix.
DOT	Returns the dot product of two vectors.
MAT	Performs various operations on numeric and string arrays.
MAT REORDER	Reorders the elements in an array according to the subscript list in a vector.
MAT SEARCH	Searches an array for user-defined conditions.
MAT SORT	Sorts an array along one dimension according to lexical or numeric order.
RANK	Returns the number of dimensions in an array.
REDIM	Changes the subscript range of an array.
SIZE	Returns the number of elements in a dimension of an array.
SUM	Returns the sum of all the elements in a numeric array.

Vocabulary

The following list contains all the words which are recognized by Series 200/300 computers with BASIC 5.0. Each individual word is some part of one or more valid statements or functions. These words cannot be used as variable names unless you mix their letter case.

ABORT
ABORTIO
ABS
ACK
ACS
ACSH
ALL
ALLOCATE
ALPHA
AND
AREA
ARG
ASCII
ASN
ASNH
ASSIGN
ATN
ATNH
AXES

BASE
BDAT
BEEP
BIN
BINAND
BINCMP
BINEOR
BINIOR
BIT
BREAK
BUFFER
BY
BYTE

CALL
CASE
CAT
CAUSE
CDIAL
CHANGE
CHECKREAD
CHGRP
CHOWN
CHR
CHR$
CHRX
CHRY
CLEAR
CLIP
CLS
CM
CMD
CMODE
CMPLEX
COLOR
COM
COMPLEX
CONDITIONAL
CONFIGURE
CONJG
CONT
CONTROL
CONVERT
COPY
COPYLINES
COS
COSH
COUNT
CREATE
CRT
CSIZE
CSUM
CYCLE

DATA
DATE
DATE$
DDC
DEALLOCATE
DEF
DEG
DEL
DELAY
DELETE
DELIM
DELSUB
DES
DESTROY
DET
DEVICE
DIGITIZE
DIM
DIR
DISABLE
DISP
DISPLAY
DIV
DKA
DOT
DRAW
DROUND
DUMP
DVAL
DVAL$

ECHO
EDGE
EDIT
ELSE
ENABLE
END
ENTER
EOL
EOR
EOT
ERRDS
ERRL
ERRLN
ERRM$
ERRN
ERROR
EVEN
EXD
EXEC
EXECUTE
EXIT
EXOR
EXP
EXPANDED
EXT
EXTEND

FILL
FIND
FN
FNEND
FOR
FORMAT
FRACT
FRAME
FRENCH
FROM
FUNCTIONS

GCLEAR
GERMAN
GESCAPE
GET
GINIT
GLOAD
GO
GOSUB
GOTO
GRAPHICS
GRID
GROUP
GSEND
GSTORE

HEADER
HEIGHT
HIL
HILBUF

IDD
IDN

IDRAW
IF
IMAG
IMAGE
IMOVE
IN
INDENT
INDEX
INITIALIZE
INPUT
INT
INTEGER
INTENSITY
INTERACTIVE
INTR
INV
IO
IPLOT
IS
IVAL
IVAL$

KBD
KBD$
KEY
KEYS
KNOB
KNOBX
KNOBY

LABEL
LABELS
LDIR
LEN
LET
LEXICAL
LGT
LINE
LINK
LINPUT
LIST
LISTEN
LL
LN
LOAD
LOADSUB
LOC
LOCAL
LOCATE
LOCATOR
LOCK
LOCKOUT
LOG
LOOP
LORG
LWC$

MAIN
MANAGER
MAP
MASK
MASS
MAT
MAX
MAXLEN
MAXREAL
MERGE
MIN
MINREAL
MLA
MOD
MODE
MODULO
MOVE
MOVELINES
MSI
MTA

NAMES
NEXT
NF
NO
NOT
NPAR
NUM
NV

ODD
OFF
ON
ONE
OPTION
OPTIONAL
OR
ORDER
OTHER
OUT
OUTPUT
OWNER

PAIRS
PARITY
PASS
PAUSE
PDIR
PEN
PENUP
PERMIT
PI
PIVOT
PLOT
PLOTTER
POLYGON
POLYLINE
POS
PPOLL
PRINT
PRINTALL
PRINTER
PRIORITY
PRM
PROTECT
PROUND
PRT
PURGE

QUIT

RAD
RANDOMIZE
RANK
RATIO
RE
READ
READIO
REAL
RECALL
RECORDS
RECOVER
RECTANGLE
REDIM
REM
REMOTE
REN
RENAME
REORDER
REPEAT
REQUEST
RES
RESET
RESPONSE
RESTORE
RE-STORE
RESUME
RETAIN
RETURN
REV$
RND
RNM
ROTATE
RPLOT
RPT$
RRG
RSC
RST
RSUM
RUN

SAVE
SB
SC
SCALE
SCRATCH
SCREEN
SEARCH
SEC
SECURE
SELECT
SEND
SEPARATE
SET
SF
SGN
SHIFT
SHOW
SIGNAL
SIN
SINH
SIZE
SKIP
SORT
SOUND
SPANISH
SPOLL
SQR
SQRT
STANDARD
STATUS
STEP
STOP
STORAGE
STORE
SUB
SUBEND
SUBEXIT

SUM
SUSPEND
SV
SWEDISH
SYMBOL
SYSBOOT
SYSTEM
SYSTEM$

TAB
TABXY
TALK
TAN
TANH
THEN
TIME
TIME$
TIMEDATE
TIMEOUT
TIMEZONE
TO
TRACE
TRACK
TRANSFER
TRIGGER
TRIM$
TRN
TYPE

UN
UNCONFIGURE
UNL
UNLOCK
UNT
UNTIL
UPC$
USER
USING

VAL
VAL$
VIEWPORT

WAIT
WHERE
WHILE
WIDTH
WINDOW
WORD
WRG
WRITE
WRITEIO

XREF

ZERO

Note 1: Although LOCATE and SCALE are recognized as reserved words when entered, they are stored and listed back as VIEWPORT and WINDOW, respectively.

Note 2: Although CSUB can appear as a reserved word in a program listing, it is not recognized as a reserved word when entered from the keyboard.

Notes

HEWLETT
PACKARD

HP Part Number
98613-90052

Microfiche No. 98613-99052
Printed in U.S.A.   E0888

98613-90642
For Internal Use Only