HEWLETT PACKARD

Part No. 09826-90050
E0682
Microfiche No. 09826-99050

Printed in U.S.A.
First Edition June 1982

HP Computer Systems

BASIC 2.0 Condensed Reference

HEWLETT PACKARD

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

BASIC 2.0 Condensed Reference

Manual Part No. 09826-90050

Hewlett-Packard Calculator Products Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

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

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

June 1982...First Edition

Table of Contents

Data Types

I/O path A combination of firmware and hardware that can be used for the transfer of data to and from a BASIC program. Associated with an I/O path is a unique data type that describes the I/O path. This association table uses about 200 bytes and is referenced when the I/O path name is used.

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.

REAL A numeric data type that is stored internally in eight bytes using sign-and-magnitude representation. One bit is used for the number's sign, 11 bits for a biased exponent (bias = 1023), and 52 bits for a mantissa. There is an implied "1." preceding the mantissa (this can be thought of as the 53rd bit). Approximated to four digits, the range of REAL numbers is:

−1.798 E+308 thru −2.225 E−308, 0, and
+2.225 E−308 thru +1.798 E+308.

If a numeric variable is not explicitly declared as an INTEGER, it is a REAL.

string A data type comprised of a contiguous series of characters. Each character in the string is stored in one byte 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.

Expression Evaluation

Operators

Dyadic Operator	Operation
+	REAL or INTEGER addition
-	REAL or INTEGER subtraction
*	REAL or INTEGER multiplication
/	REAL division
^	Exponentiation
&	String concatenation
DIV	Gives the integer quotient of a division
MOD	Gives the integer remainder (modulus) of a division
=	Comparison for equality
<>	Comparison for inequality
<	Comparison for less than
>	Comparison for greater than
<=	Comparison for less than or equal to
>=	Comparison for greater than or equal to
AND	Logical AND
OR	Logical inclusive OR
EXOR	Logical exclusive OR

Monadic Operator	Operation
-	Reverses the sign of an expression
+	Identity operator
NOT	Logical complement

Math Hierarchy

Operators of equal precedence are evaluated left-to-right.

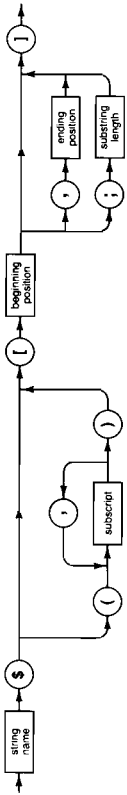
Precedence	Operator
High	Parentheses: (may be used to force any order of operations)
	Functions, user-defined and machine-resident
	Exponentiation
	Multiplication and division
	Plus and minus
	Relational operators
	NOT
	AND
Low	OR EXOR

String Hierarchy

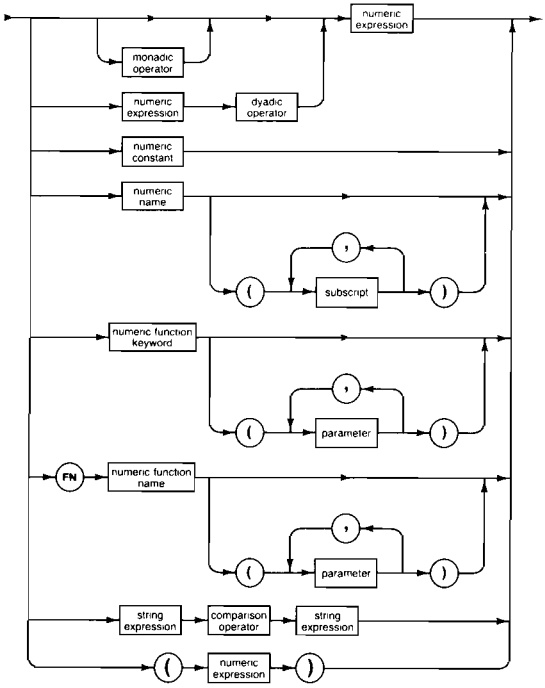
Precedence	Operator
High	Parentheses
	Substrings and functions
Low	Concatenation

Substrings

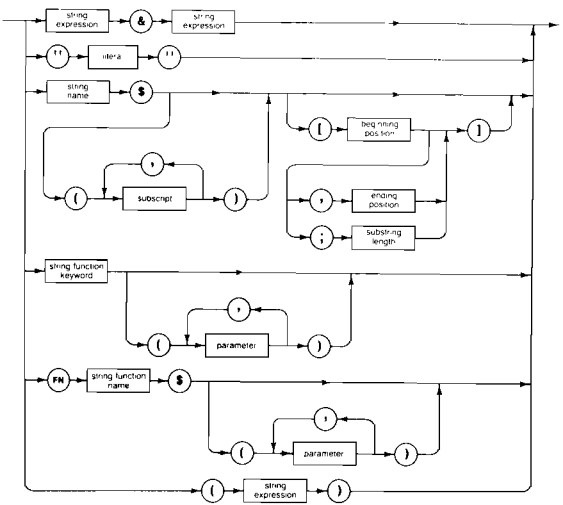
The beginning position must be at least one and no greater than the current length plus one. The ending position must be no less than the beginning position minus one and no greater than the dimensioned length of the string. The maximum substring length must be at least zero and no greater than one plus the dimensioned length of the string minus the beginning position. (See drawing.)



Numeric Expressions



String Expressions

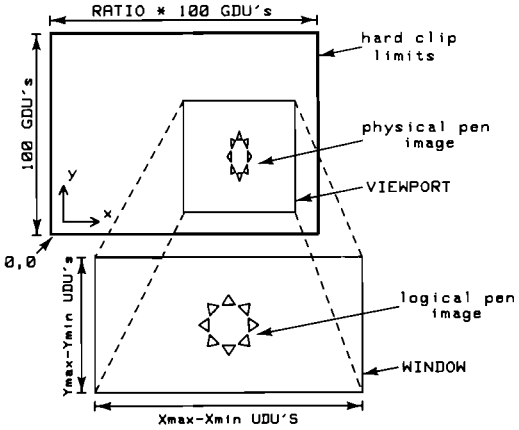


Simplified Graphics Mapping

VIEWPORT, specified in Graphics Display Units (GDU), defines the area of the screen into which a coordinate system defined by WINDOW or SHOW is mapped.

WINDOW specifies a coordinate system, measured in User Defined Units (UDU), that is mapped into the area defined by VIEWPORT. The x and y bounds specified in a WINDOW statement are placed exactly on the VIEWPORT bounds, and the aspect ratio of the co-ordinate system is adjusted accordingly (non-isotropic view).

SHOW is similar to WINDOW, except the range of the specified coordinate system is adjusted by the computer to maintain equal UDU on both axes (isotropic view).



Glossary

array A structured data type that can be of type REAL, INTEGER, or string. Arrays are created with the DIM, REAL, INTEGER, ALLOCATE, or COM statements. Arrays have 1 to 6 dimensions; each dimension is allowed 32 767 elements. The lower and upper bounds for each dimension must fall in the range −32 767 (−32 768 for ALLOCATE) thru +32 767, and the lower bound must not exceed the upper bound. The default lower bound is the OPTION BASE value; the OPTION BASE statement can be used to specify 0 or 1 as the default lower bound. The default OPTION BASE at power-on or SCRATCH A 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 lower bound or greater than the 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.

command A statement that is executed from the keyboard input line. (Also see "statement".)

context An instance of an environment. A context consists of a specific instance of all data types which may be accessed by a program at a specific point in its execution.

device selector A numeric expression used to specify the source or destination of an I/O operation. A device selector can be either an interface select code or a combination of an interface select code and a primary address. To construct a device selector with a primary address, multiply the interface select code by 100 and add the primary address.

Secondary addresses may be appended after a primary address by multiplying the device selector by 100 and adding the address. This may be repeated up to 6 times, adding a new secondary address each time. A device selector, once rounded, can contain a maximum of 15 digits.

When a device selector contains an odd number of digits, the leftmost digit is the interface select code. For an even number of digits, the leftmost two digits are the interface select code. For example, 70502 selects interface 7, primary address 05, and secondary address 02. Device selector 1516 selects interface 15 and primary address 16.

file name A file name consists of one to ten characters. Valid file names can contain uppercase letters, lowercase letters, numerals, the underbar (_), and CHR$(161) thru CHR$(254). LIF-compatible file names can contain only uppercase letters and numerals. The first character in a LIF-compatible file name must be a letter.

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. A function is considered numeric if it returns a numeric quantity.

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 interfaces. (Also see "device selector".)

msus This is the acronym for "mass storage unit specifier". It is a string expression that specifies a device to be used for mass storage operations.

name A name consists of one to fifteen characters. The first character must be an uppercase ASCII letter or one of the characters from CHR$(161) thru CHR$(254). The remaining characters, if any, can be lowercase ASCII letters, numerals, the underbar (_), or CHR$(161) thru CHR$(254). Names may be typed using any combination of uppercase and lowercase letters, unless the name uses the same letters as a keyword. Conflicts with keywords can be resolved by mixing the lettercase in the name. (Also see "file name".)

primary address A numeric expession in the range of 0 thru 31 that specifies an individual device on an interface which is capable of servicing more than one device. The HP-IB interface can service multiple devices. (Also see "device selector".)

program line A statement that is preceded by a line number (and an optional line label) and stored with the ENTER key into a program. (Also see "statement".)

recursive See the *BASIC Language Reference*.

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".)

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 executed from the keyboard input line, it is called a command.

subprogram Can be either 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 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. Either type of subprogram may alter the values of parameters passed by reference or variables in COM.

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.

Keywords

A

ABORT

This statement ceases HP-IB activity. When system controller but not active controller, ABORT causes the computer to assume active control.

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

ABS

This function returns the absolute value of its argument. The result will be of the same type (REAL or INTEGER) as the argument.

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

ACS

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

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

ALLOCATE

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

```
ALLOCATE Temp(Low:High)
ALLOCATE INTEGER Array(Index,2,8)
ALLOCATE R$[LEN(A$)+1]
ALLOCATE Text$(Lines)[80]
```

ALPHA

This statement turns the alphanumeric display on or off.

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


AND

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

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

ASN

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

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

ASSIGN

This statement assigns an I/O path name and attributes to a device, or group of devices, or a mass storage file.

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

ATN

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

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

AXES

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

```
AXES 10,10
AXES X,Y,Midx,Midy,Maxx/10,Maxy/10
AXES Xspace,Yspace,Xlocy,Ylocx,Xmajor,
 Ymajor,Majorsize
```

B

BEEP

This statement produces one of 63 audible tones.

```
BEEP
BEEP Freq,Duration
BEEP 81.38*Tone,Seconds
```

BINAND

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

```
Low_4_bits=BINAND(Byte,15)
IF BINAND(Stat,3) THEN Bit_set
```

BINCMP

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

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

BINEOR

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

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

BINIOR

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

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

BIT

This function returns a 1 or 0 representing the value of the specified bit of its argument. The least-significant-bit is bit 0.

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


C

CALL

This statement transfers program execution to the specified SUB subprogram and may pass parameters to the subprogram.

```
CALL Process(Reference,(Value),@Path)
CALL Transform(Array(*))
Parse(String$(Line)[8],Pointer+Offset)
ON END @File CALL Sortnumbers
IF Flag THEN CALL Special
```

CAT

This statement lists the contents of the mass storage media's directory.

```
CAT
CAT TO #701
CAT ":INTERNAL"
CAT ":INTERNAL,4,1" TO #12
```

CHR$

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

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

CLEAR

This statement allows the active controller to put HP-IB devices into a device-dependent state.

```
CLEAR 7
CLEAR Voltmeter
CLEAR @Source
```

CLIP

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

```
CLIP Left,Right,Bottom,Top
CLIP ON
CLIP OFF
```

COM

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

```
COM X,Y,Z
COM /Block/ Text$,@Path,INTEGER Points(*)
COM INTEGER I,J,REAL Array(-128:127)
```

CONT

This command resumes execution of a paused program at the specified line. If no line is specified, execution resumes at the line pointed to by the program counter.

```
CONT
CONT 550
CONT Sort
```

CONTROL

This statement sends control information to an interface or to the internal table associated with an I/O path name.

```
CONTROL @Rand_file,7;File_length
CONTROL 1;Row,Column
CONTROL Interface,Register;Value
```

COPY

This statement allows copying of individual files or entire discs.

```
COPY Old_file$ TO New_file$
COPY "MY_FILE" TO "TEMP<pc>:INTERNAL,4,1"
COPY ":INTERNAL" TO ":INTERNAL,4,1"
COPY Int_disc$ TO Ext_disc$
```


COS

This function returns the cosine of the argument. The range of the returned REAL value is −1 thru +1.

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

CREATE ASCII

This statement creates an ASCII file of specified length on the mass storage media.

```
CREATE ASCII "TEXT",100
CREATE ASCII Name$&":INTERNAL,4,1",Length
```

CREATE BDAT

This statement creates a BDAT file of specified length on the mass storage media.

```
CREATE BDAT "George",48
CREATE BDAT "Abc<PC>",Records,Record_len
CREATE BDAT Name$&Msus$,Bytes,1
```

CSIZE

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

```
CSIZE 10
CSIZE 5,0.6
CSIZE Size,Width/Height
```

D

DATA

This statement contains data which can be read by READ statements.

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

DEALLOCATE

This statement deallocates memory space reserved by the ALLOCATE statement.

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

DEF FN

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

```
980  END
990  !
1000 DEF FNNew$(String$)
.
.
.
1120  RETURN R$
1130  FNEND

2000 DEF FNXform(@Ptr,INTEGER Array(*),
 OPTIONAL Text$)
.
.
.
```

DEG

This statement selects degrees as the unit of measure for expressing angles. Radians are assumed unless a DEG statement is executed.

DEL

This command deletes program lines. If only one line identifier is specified, only that line is deleted.

```
DEL 15
DEL Start,Last
DEL Sort,32000
```


DELSUB

This statement deletes one or more SUB subprograms or user-defined function subprograms from memory. Comments are associated with the subprogram above them.

```
DELSUB FNTrim$
DELSUB Process TO END
DELSUB Special1,Special3
```

DIM

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

```
DIM String$[100],Name$(12)[32]
DIM Param(48,8,8,2,2,2)
DIM Array(-128:127,16)
```

DISABLE

This statement disables event-initiated branches which were defined by ON KBD, ON KEY, ON KNOB, and ON INTR statements.

DISABLE INTR

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

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

DISP

This statement causes the display items to be sent to the display line on the CRT. See IMAGE for specifier descriptions.

```
DISP Prompt$;
DISP TAB(5),First,TAB(20),Second
DISP
DISP Name$,Id;Code
DISP USING Form3;Item(1),Item(2)
DISP USING "5Z.DD";Money
```

DIV

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

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

DRAW

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

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

DROUND

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

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

DUMP

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

```
DUMP ALPHA
DUMP GRAPHICS #702
```

DUMP DEVICE IS

This statement specifies which device receives the data when either DUMP ALPHA or DUMP GRAPHICS is executed without a device selector. Specifying EXPANDED results in dumps that are 2X on each axes and rotated 90°. Device 701 is assumed unless a DUMP DEVICE IS statement is executed.

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


E

EDIT

This command allows you to enter a new program or edit a program already in memory. Refer to Chapter 2 of *BASIC Programming Techniques*.

```
EDIT
EDIT Label2
EDIT 1000,5
```

ENABLE

This statement re-enables all ON KBD, ON KEY, ON KNOB, and ON INTR branches which were suspended by DISABLE.

ENABLE INTR

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

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

END

This non-optional statement marks the end of the main program. Subprograms (if any) follow the END statement.

ENTER

This statement is used to input data from a device, file, or string and assign the values entered to variables. See IMAGE for specifier descriptions.

```
ENTER 705;Number,String$
ENTER Device;X;Y;Z
ENTER Command$;Parameter
ENTER @File;Array(*)
ENTER @Random,Record USING 20;Text$(Line)
ENTER @Source USING Fmt5;B(1),B(2),B(3)
ENTER 12 USING "#,6A";A$[2;6]
```

ERRL

This function returns a value of 1 if the most recent error occurred in the specified line. Otherwise, a value of 0 is returned. The specified line must be in same context as the ERRL function.

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

ERRN

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

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

EXOR

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

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

EXP

This function raises e to the power of the argument. Internally, Naperian $e \approx 2.718\,281\,828\,459\,05$.

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

F

FN

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

```
PRINT X;FNChange(X)
Final$=FNStrip$(First$)
Param=FNProcess(Reference,(Value),@Path)
R=FNTrans(Item(Start+Offset),LookUp(*))
```


FOR...NEXT

This construct defines a loop which is repeated until the loop counter passes a specified value.

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

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

FRAME

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

G

GCLEAR

This statement clears the graphics display or sends a command to an external plotter to advance the paper.

GET

This statement reads the specified ASCII file and attempts to store the strings into memory as program lines. If specified, program lines are renumbered to the first line identifier and run at the second line identifier. Also see LOAD.

```
GET "George"
GET Name$&Msus$,New_process
GET Next_prog$,180,10
```

GINIT

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

GLOAD

This statement loads the contents of an INTEGER array into the graphics display memory. Also see GSTORE.

```
GLOAD Picture(*)
IF Flag THEN GLOAD Array(*)
```

GOSUB

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

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

GOTO

This statement transfers program execution to the specified line. The specified line must be in the current context.

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

GRAPHICS

This statement turns the graphics display on or off. This statement has no effect on the contents of the graphics memory, it just controls whether it is displayed or not.

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

GRID

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

```
GRID 10,10
GRID Xspace,Yspace,Xlocy,Ylocx,Xcount,
 Ycount,Minor_size
```


GSTORE

This statement stores the contents of the graphics display memory into an INTEGER array. The size of the array must correspond exactly to the size of the graphics memory. Also see GLOAD.

```
GSTORE Picture(*)
IF Final THEN GSTORE A(*)
```

I

IDRAW

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

```
IDRAW X+50,0
IDRAW Delta_x,Delta_y
```

IF...THEN

This statement provides conditional branching.

```
150  IF Flag THEN Next_file
160  IF Pointer<1 THEN Pointer=1

580  IF First_pass THEN
590    Flag=0
600    INPUT "Command?",Cmd$
610    IF LEN(Cmd$) THEN GOSUB Parse
620  END IF

1000  IF X<0 THEN
1010    BEEP
1020    DISP "Improper Argument"
1030  ELSE
1040    Root=SQR(X)
1050  END IF
```

IMAGE

This statement provides image specifiers for the formatting of data which is entered or output with various statements. These specifiers can also be included after a USING clause in statements supporting that syntax.

```
IMAGE 4Z,DD,3X,K,/
IMAGE "Result = ",SDDDE,3(XX,ZZ)
IMAGE #,B
```

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

Specifier	Meaning
K	Freefield Entry. Numeric: Entered characters are sent to number builder. Leading non-numeric characters are ignored. All blanks are ignored. Trailing non-numeric characters and characters sent with EOI true are delimiters. Numeric characters include digits, decimal point, +, −, e, and E when their order is meaningful. String: Entered characters are placed in the string. A carriage-return not immediately followed by a line-feed is entered into the string. Entry to a string terminates on CR/LF, LF, a character received with EOI true, or when the dimensioned length of the string is reached.
−K	Like K except that LF is entered into a string, and thus CR/LF and LF do not terminate the entry.
S	Same action as D.
M	Same action as D.
D	Demands a character. Non-numerics are accepted to fill the character count. Blanks are ignored; other non-numerics are delimiters.

Specifier	Meaning
Z	Same action as D.
B	Demands one byte. The byte becomes a numeric quantity.
W	Demands two bytes. The two bytes are considered to be a 16-bit, two's-complement integer. The first byte entered on an 8-bit interface is the most significant byte. The word is entered in a single operation on a 16-bit interface.
A	Demands a string character. Any character received is placed in the string.
X	Skips a character.
.	Same action as D.
E	Same action as 4D.
ESZZ	
ESZ	Same action as 3D.
ESZZZ	Same action as 5D.
#	Statement is terminated when the last ENTER item is terminated. EOI and line-feed are item terminators.
%	Same as # except EOI is a statement terminator. Early termination is allowed if the current item is satisfied.
/	Demands a new field; skips all characters to the next line-feed. EOI is ignored.
L	Ignored for ENTER.
@	Ignored for ENTER.
literal	Skips one character for each character in the literal.

Effects of the image specifiers on a DISP, LABEL, OUTPUT, or PRINT statement are shown in the following table.

Specifier	Meaning
K −K	Compact field. Outputs a number or string in standard form with no leading or trailing blanks.
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.
B	Outputs the character represented by one byte of data. This is similar to the CHR$ function. The least significant eight bits of the number are sent.
W	Outputs two characters represented by the two bytes in a 16-bit word. On an 8-bit interface, the most significant byte is sent first. The word is sent in a single operation on a 16-bit interface.
A	Outputs a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the string. If the image specifier is exhausted before the string, the remaining characters are not sent.
X	Outputs a blank.
.	Outputs a decimal point radix indicator.
E ESZZ	Outputs an E, a sign, and a two digit exponent.
ESZ	Outputs an E, a sign, and a one digit exponent.

Specifier	Meaning
ESZZZ	Outputs an E, a sign, and a three digit exponent.
#	Suppresses automatic output of the EOL (End-Of-Line) sequence at the end of the output list.
%	Ignored in output images.
L	Outputs an EOL sequence.
@	Outputs a form-feed.
/	Outputs a carriage-return and a line-feed.
literal	Outputs the characters contained in the literal.

IMOVE

This statement updates the logical pen position, by adding the X and Y displacements to the current logical pen position. The X and Y increments are interpreted according to the current unit-of-measure.

```
IMOVE X+50,0
IMOVE Delta_x,Delta_y
```

INITIALIZE

This statement prepares mass storage media for use by the computer. When INITIALIZE is executed, **any data on the media is lost.**

```
INITIALIZE ":INTERNAL"
INITIALIZE Disc$,Interleave
```

INPUT

This statement is used to assign keyboard input to program variables.

```
INPUT "Name?",N$,"ID Number?",Id
INPUT "Enter 3 numbers",V(1),V(2),V(3)
INPUT "",String$[1;10]
INPUT Array(*)
```

INT

This function returns the greatest integer which is less than or equal to the expression. The result will be of the same type (REAL or INTEGER) as the argument.

```
Whole=INT(Number)
PRINT "Integer portion =";INT(X)
```

INTEGER

This statement declares INTEGER variables, dimensions INTEGER arrays, and reserves memory for them.

```
INTEGER I,J,K
INTEGER Array(-128:255,4)
```

IPLOT

This statement moves the pen from the current pen position to a position calculated by adding the specified X and Y displacements to the current pen position. Plotting action is determined by the current line type and the optional pen control parameter (see PLOT).

```
IPLOT 0,5
IPLOT Delta_x,Delta_y,Pen_control
```

K

KBD$

This function returns the contents of the keyboard buffer. Also see ON KBD.

```
PRINT KBD$;
Buffer$=Buffer$&KBD$
```

KNOBX

This function returns the net number of knob pulses counted since the last time the KNOBX counter was zeroed. Invoking the KNOBX function zeros the counter after it is read. Also see ON KNOB.

```
Position=KNOBX
IF KNOBX<0 THEN Backwards
```


L

LABEL

This statement sends text to graphic devices.

```
LABEL Number,String$
LABEL X(Offset)+K;A$[1,8];
LABEL Array(*)
LABEL USING 160;X,Y,Z
LABEL USING "5Z.DD";Money
```

LDIR

This statement defines the angle at which labels are drawn. The angle is interpreted as counter-clockwise, from horizontal. The current angle mode is used.

```
LDIR 90
LDIR ACS(Side)
```

LEN

This function returns the current number of characters in the argument. The length of the null string is 0.

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

LET

This is the assignment statement, which is used to assign values to variables.

```
LET Number=33
Array(I+1)=Array(I)/2
String$="Hello Scott"
A$(7)[1;2]=CHR$(27)&"Z"
```

LGT

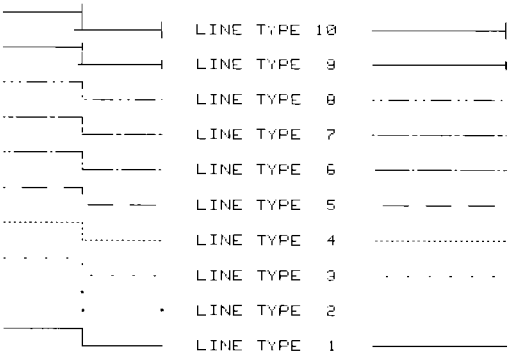
This function returns the logarithm (base 10) of its argument.

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

LINE TYPE

This statement selects a line type and repeat length for lines, labels, frames, axes and grids. CRT line types are shown below.

```
LINE TYPE 1
LINE TYPE Style,Repeat_len
```



LINPUT

This statement accepts alphanumeric input from the keyboard for assignment to a string variable. The LINPUT statement allows commas or quotation marks to be included in the value of the string, and leading or trailing blanks are not deleted.

```
LINPUT "Next Command?",Response$
LINPUT Array$(I)[3]
```

LIST

This statement lists the program currently in memory to the selected device. Beginning and ending line labels or numbers may be specified to list parts of the program.

```
LIST #701
LIST #Device;Label1,Label2
LIST 110,250
```


LOAD

This statement loads PROG files into memory. PROG files are created by the STORE statement. A line identifier may be included to specify the line where execution will begin after the LOAD. Also see GET.

```
LOAD "George"
LOAD "TEMP:INTERNAL,4,1",Label1
LOAD Next_file$,500
```

LOAD BIN

This command loads a BIN file into memory. BIN files are created with the STORE BIN statement.

```
LOAD BIN "BEB"
LOAD BIN "HP98627A:INTERNAL,4,1"
LOAD BIN File_name$
```

LOADSUB

This statement loads BASIC subprograms from a file of type PROG into memory. PROG files are created by the STORE statement.

```
LOADSUB ALL FROM "George"
LOADSUB ALL FROM "TEMP:INTERNAL,4,1"
LOADSUB ALL FROM Name$&Msus$
```

LOCAL

This statement returns all specified devices to their local state.

```
LOCAL @Dvm
LOCAL 728
LOCAL Ics
```

LOCAL LOCKOUT

This HP-IB statement sends the LLO (local lockout) message, preventing an operator from returning the specified device to local (front panel) control.

```
LOCAL LOCKOUT 7
LOCAL LOCKOUT Isc
LOCAL LOCKOUT @Hpib
```

LOG

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

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

LOOP

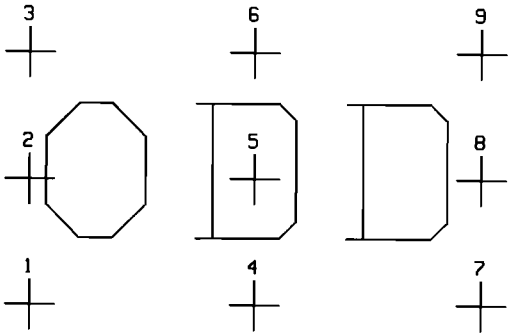
This construct defines a loop which is repeated until the expression in an EXIT IF statement is evaluated as true (non-zero). There may be any number of EXIT IF statements, but they must be at the same nesting level as the LOOP statement.

```
550  LOOP
560    Test=RND-.5
570  EXIT IF Test>.4
580    CALL Simulate
590  END LOOP
```

LORG

This statement specifies the relative origin of labels with respect to the current pen position.

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



M

MASS STORAGE IS

This statement specifies the system mass storage device.

```
MASS STORAGE IS ":INTERNAL,4,1"
MASS STORAGE IS Msus$
```

MOD

This operator returns the remainder of an integer division.

```
Remainder=Dividend MOD Divisor
PRINT "Seconds =";Time MOD 60
```

MOVE

This statement updates the logical pen position.

```
MOVE 10,75
MOVE Next_x,Next_y
```

N

NOT

This operator returns 1 if its argument equals 0. Otherwise, 0 is returned.

```
Invert_flag=NOT Std_device
IF NOT Pointer THEN Next_op
```

NPAR

This function returns the number of parameters passed to the current subprogram.

```
IF NPAR>3 THEN Extra
Factors=NPAR-2
```

NUM

This function returns the decimal value of the ASCII code of the first character in the argument. The range of returned values is 0 thru 255.

```
Ascii_val=NUM(String$)
A$[I;1]=CHR$(NUM(A$[I])+32)
```

O

OFF END

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

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

OFF ERROR

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

OFF INTR

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

```
OFF INTR
OFF INTR 12
OFF INTR Hpib
```

OFF KBD

This statement cancels event-initiated branches previously defined and enabled by an ON KBD statement. Further keypresses are sent to the operating system in the normal manner.

OFF KEY

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

```
OFF KEY
OFF KEY 4
OFF KEY Current_Key
```

OFF KNOB

This statement cancels event-initiated branches previously defined and enabled by the ON KNOB statement. Further use of the knob results in normal scrolling or cursor movement.

OFF TIMEOUT

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

```
OFF TIMEOUT
OFF TIMEOUT 12
OFF TIMEOUT Hpib
```

ON

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

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

ON END

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.

```
ON END @Source GOTO 500
ON END @Device GOSUB Select
ON END @File RECOVER Label1
ON END @Dest CALL Expand
```

ON ERROR

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.

```
ON ERROR GOTO 1200
ON ERROR GOSUB Fix
ON ERROR RECOVER Label1
ON ERROR CALL Report
```

ON INTR

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.

```
ON INTR 7 GOTO 500
ON INTR Hpib,Priority GOSUB Service
ON INTR 12 RECOVER Label1
ON INTR Isc,4 CALL Service
```

ON KBD

This statement defines and enables an event-initiated branch which occurs when a key is pressed.

```
ON KBD GOTO 150
ON KBD,Priority GOSUB Label1
ON KBD ALL RECOVER 880
ON KBD ALL,3 CALL Keypress
```

ON KEY

This statement defines and enables an event-initiated branch which occurs when a softkey is pressed.

```
ON KEY 0 GOTO 150
ON KEY Number,Priority GOSUB Label1
ON KEY 1 LABEL A$ RECOVER 770
ON KEY 5 LABEL "Print",3 CALL Report
```

ON KNOB

This statement defines and enables event-initiated branches which result from turning the knob.

```
ON KNOB .1 GOTO 250
ON KNOB Seconds,Priority GOSUB Service
ON KNOB 1/2 RECOVER Label1
ON KNOB .333,4 CALL Pulses
```


ON TIMEOUT

This statement defines and enables an event-initiated branch which results from an I/O timeout on the specified interface.

```
ON TIMEOUT 7,4 GOTO 770
ON TIMEOUT Isc,Seconds GOSUB Abort
ON TIMEOUT 12,1/2 RECOVER Label1
ON TIMEOUT Printer,Time CALL Message
```

OPTION BASE

This statement specifies the default lower bound of arrays. Zero is the assumed lower bound unless an OPTION BASE statement is executed.

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

OR

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

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

OUTPUT

This statement outputs items to a specified destination. See IMAGE for specifier descriptions.

```
OUTPUT 701;Number,String$;
OUTPUT @File;Array(*),END
OUTPUT @Random,Record USING Fmt1;Item(5)
OUTPUT 12 USING "#,6A";B$[2;6]
OUTPUT Dest$ USING 110;A/1000,VAL$(Res),
OUTPUT @Printer;Rank;Id;Name$
```

P

PAUSE

This statement suspends program execution.

PEN

This statement selects a pen on the current plotting device.

```
PEN -1
PEN Select
```

CRT Pen	Action
1	Draw
0	Complement
-1	Erase

PENUP

This statement lifts the pen on an external plotter.

PI

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

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

PIVOT

This statement specifies a rotation of coordinates which is applied to all drawn lines, but not to labels or axes.

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

PLOT

This statement moves the pen from the current pen position to the specified X and Y coordinates. Plotting action is determined by the current line type and the optional pen control parameter.

```
PLOT 20,90
PLOT Next_x,Next_y,Pen_control
```


Pen Control	Pen Action
Negative Even	Up before move
Negative Odd	Down before move
Positive Even	Up after move
Positive Odd	Down after move
Default	Down after move

PLOTTER IS

This statement selects a plotting device. Device 3,"INTERNAL" (the CRT) is assumed unless a PLOTTER IS statement is executed.

POS

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

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

PPOLL

This function conducts a Parallel Poll and returns a byte representing eight status-bit messages of the devices on an HP-IB.

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

PPOLL CONFIGURE

This statement programs the logical sense and data bus line on which a specified device responds to a parallel poll. Response line is specified by bits 0 thru 2 and logic sense is specified by bit 3.

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

PPOLL UNCONFIGURE

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

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

PRINT

This statement sends items to the PRINTER IS device. See IMAGE for specifier descriptions.

```
PRINT "LINE";Number;
PRINT Array(*),
PRINT TABXY(1,1),Head$,TABXY(Col,3),Msg$
PRINT String$[1,8],TAB(12),Result
PRINT USING 125;X,Y,Z
PRINT USING "5Z.DD";Money
PRINT USING Fmt3;Id,Item$,Kilograms/2.2
```

PRINTALL IS

This statement assigns a logging device for recording operator interaction and troubleshooting messages.

```
PRINTALL IS 701
PRINTALL IS Gpio
```

PRINTER IS

This statement specifies the system printing device for all PRINT, CAT and LIST statements which do not specify a destination. The PRINTER IS device is 1 (the CRT) at power-on and after SCRATCH A.

```
PRINTER IS 701
PRINTER IS Gpio
```

PROTECT

This statement specifies the protect code used on PROG, BDAT, and BIN files.

```
PROTECT Name$,Pc$
PROTECT "George<xx>:INTERNAL,4,1","NEW"
```

PURGE

This statement deletes a file entry from the directory of the mass storage media.

```
PURGE Name$&Msus
PURGE "TEMP:INTERNAL,4,1"
PURGE "George<PC>"
```


R

RAD

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

RANDOMIZE

This statement selects a seed for the RND function.

```
RANDOMIZE
RANDOMIZE Old_seed*PI
```

RATIO

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

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

READ

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

```
READ Number,String$
READ Array(*)
READ Field$[5,15]
READ Item(1,1),Item(2,1),Item(3,1)
```

READIO

This function reads the contents of the specified hardware register on the specified interface.

```
Upper_byte=READIO(Gpio,4)
PRINT "Register";I;"=";READIO(7,I)
```

REAL

This statement reserves storage for floating point variables and arrays.

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

REM

This statement allows comments in a program.

```
100  REM   Program Title
190  !
200  Info=0 ! Clear flag byte
```

REMOTE

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

```
REMOTE 712
REMOTE Device
REMOTE @Hpib
```

REN

This command renumbers the lines in a program. An increment of 10 is assumed unless otherwise specified by the second parameter.

```
REN 1000
REN 100,2
```

RENAME

This statement changes a file's name in the mass storage media's directory.

```
RENAME "TEMP<pc>" TO "FINAL"
RENAME "OLD:INTERNAL,4,1" TO "NEW"
RENAME Name$&Msus$ TO Temp$
```

REPEAT...UNTIL

This construct defines a loop which is repeated until the expression in the UNTIL statement is evaluated as true (non-zero).

```
770  REPEAT
780    CALL Process(Param)
790    Param=Param*Scaling
800  UNTIL Param>Maximum
```


RE-SAVE

This statement creates an ASCII file and copies program lines as strings into that file. If the specified file already exists, the old entry is purged after the new file is successfully saved.

```
RE-SAVE "George"
RE-SAVE "TEMP:INTERNAL,4,1",Label1
RE-SAVE Name$&Msus$,1,Sort
```

RESTORE

RESTORE specifies which DATA statement will be used by the next READ operation.

```
RESTORE
RESTORE Third_array
```

RE-STORE

This statement creates a PROG file and stores an internal form of the BASIC program and all normal binary programs into that file. If the specified file already exists, the old entry is purged after the new file is successfully stored.

```
RE-STORE "MYPROG<pc>"
RE-STORE "TEMP:INTERNAL,4,1"
RE-STORE Name$&Msus$
```

RE-STORE BIN

This statement creates a BIN file and stores all normal binary programs into that file.

```
RE-STORE BIN "BINPROG<pc>"
RE-STORE BIN "BIN2:INTERNAL,4,1"
RE-STORE BIN Name$&Msus$
```

RESUME INTERACTIVE

This statement restores the normal functions of any program control keys previously deactivated by SUSPEND INTERACTIVE.

RETURN

This statement returns program execution to the line following the invoking GOSUB. The keyword RETURN is also used in user-defined functions (DEF FN).

RND

This function returns a pseudo-random number greater than 0 and less than 1.

```
Percent=RND*100
IF RND<.5 THEN Case1
```

ROTATE

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified. The shift is performed with wraparound. Also see SHIFT.

```
New_word=ROTATE(Old_word,2)
Q=ROTATE(Q,Places)
```

RPLOT

This statement moves the pen from the current pen position to the specified relative X and Y position. The relative origin is set by line drawing statements other than RPLOT. The plotting action is determined by the current line type and the optional pen control parameter (see PLOT).

```
RPLOT 10,12
RPLOT Rel_x,Rel_y,Pen_control
```

RUN

This command starts program execution at the specified line.

```
RUN 10
RUN Part2
```


S

SAVE

This statement creates an ASCII file and copies program lines as strings into that file.

```
SAVE "WHALES"
SAVE "TEMP:INTERNAL,4,1",Label1
SAVE Name$&Msus$,1,Sort
```

SCRATCH

This command erases all or selected portions of memory.

```
SCRATCH
SCRATCH A
```

`SCRATCH` clears the BASIC program from memory. All variables not in COM are also cleared.

`SCRATCH C` clears all variables, including those in COM. The program is left intact.

`SCRATCH A` clears the BASIC program memory, all normal binary programs, and all variables, including those in COM. Most internal parameters in the computer are reset by this command.

SELECT...CASE

This construct provides conditional execution of one program segment chosen from several.

```
600  SELECT String$
610    CASE "0" TO "9"
620      GOSUB Digits
630    CASE ";"
640      GOSUB Delimiter
650    CASE <CHR$(32),>CHR$(126)
660      GOSUB Control_chr
670    CASE ELSE
680      GOSUB Ignore
690  END SELECT
```

SEND

This statement sends messages to an HP-IB.

```
SEND 7;UNL MTA LISTEN 1 DATA "HELLO" END
SEND @Hpib;UNL MLA TALK Pa CMD 24+128
```

CMD — The following expressions are sent as bytes with ATN true.

DATA — The following expressions are sent as bytes with ATN false. If END is added, EOI is set on the last byte.

TALK — Following expression sent as talk address (ATN true).

UNT — Sends untalk command (ATN true).

LISTEN — Following expression(s) sent as listen address (ATN true).

UNL — Sends unlisten command (ATN true).

SEC — Following expression(s) sent as secondary address (ATN true).

MTA — Sends my talk address (ATN true).

MLA — Sends my listen address (ATN true).

SET TIME

This statement resets the time-of-day given by the real-time clock.

```
SET TIME 0
SET TIME Hours*3600+Minutes*60
```

SET TIMEDATE

This statement resets the absolute seconds (time and day) given by the real-time clock.

```
SET TIMEDATE TIMEDATE+86400
SET TIMEDATE Strange_number
```


SGN

This function returns 1 if the argument is positive, 0 if it equals zero, and −1 if it is negative.

```
Root=SGN(X)*SQR(ABS(X))
Z=2*PI*SGN(Y)
```

SHIFT

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified, without wraparound. Also see ROTATE.

```
New_word=SHIFT(Old_word,-2)
Mask=SHIFT(1,Position)
```

SHOW

This statement is used to define an isotropic current unit-of-measure for graphics operations.

```
SHOW -5,5,0,100
SHOW Left,Right,Bottom,Top
```

SIN

This function returns the sine of the angle represented by the argument.

```
Sine=SIN(Angle)
PRINT "Sine of";Theta;"=";SIN(Theta)
```

SPOLL

This function returns an integer containing the serial poll response from the addressed device.

```
Stat=SPOLL(707)
IF SPOLL(@Device) THEN Respond
```

SQR

This function returns the square root of the argument.

```
Amps=SQR(Watts/Ohms)
PRINT "Square root of";X;"=";SQR(X)
```

STATUS

This statement returns the contents of I O path or interface status registers.

```
STATUS 1;Xpos,Ypos
STATUS Isc,Register;Val1,Val2,Val3
STATUS 1,9;Screenwidth
STATUS @File,5;Record
```

STOP

This statement terminates execution of the program.

STORE

This statement creates a PROG file and stores an internal form of the BASIC program and all normal binary programs into the file.

```
STORE "PROG2<pc>"
STORE "TEMP:INTERNAL,4,1"
STORE Name$&Msus$
```

STORE BIN

This command creates a BIN file and stores all normal binary programs into the file.

```
STORE BIN Name$
STORE BIN "FRED<pc>:INTERNAL,4,1"
```

SUB

This is the first statement in a SUB subprogram and specifies the subprogram's formal parameters. SUB subprograms must follow the main program's END statement and must be terminated by a SUBEND statement. See CALL.

```
SUB Parse(String$)
SUB Process
SUB Transform(@Printer,INTEGER Array(*),
 OPTIONAL Text$)
```

SUBEXIT

This statement may be used to return from a SUB subprogram at some point other than the SUBEND statement.

SUSPEND INTERACTIVE

This statement deactivates the program control keys (such as STEP and PAUSE).

```
SUSPEND INTERACTIVE
SUSPEND INTERACTIVE,RESET
```

T

TAN

This function returns the tangent of the angle represented by the argument.

```
Tangent=TAN(Angle)
PRINT "Tangent of";Z;"=";TAN(Z)
```

TIMEDATE

This function returns the current value of the real-time clock.

```
Elapsed=TIMEDATE-T0
DISP TIMEDATE MOD 86400
```

TRACE ALL

This statement allows tracing program flow and variable assignments during program execution. Trace output is sent to the system message line and (if PRINTALL is on) to the PRINTALL device.

```
TRACE ALL Sort
TRACE ALL Label1,Label2
TRACE ALL 1500,2450
```

TRACE OFF

This statement turns off all tracing activity.

TRACE PAUSE

This statement causes program execution to pause before executing the specified line, and displays the next line to be executed on the CRT. Tracing slows program execution.

```
TRACE PAUSE
TRACE PAUSE 530
TRACE PAUSE Loop_end
```

TRIGGER

This statement sends a trigger message to a selected device, or to all devices addressed to listen, on the HP-IB.

```
TRIGGER 712
TRIGGER Device
TRIGGER @Hpib
```

V

VAL

This function converts a string expression into a numeric value.

```
Day=VAL(Date$)
IF VAL(Response$)<0 THEN Negative
```

VAL$

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.

```
PRINT Esc$;VAL$(Cursor-1)
Special$=Text$&VAL$(Number)
```

VIEWPORT

This statement defines an area (in GDUs) onto which WINDOW and SHOW statements are mapped. It also sets the soft clip limits to the boundaries it defines.

```
VIEWPORT 0,35,50,80
VIEWPORT Left,Right,Bottom,Top
```


W

WHILE

This construct defines a loop which is repeated until the expression in the WHILE statement evaluates to false (zero).

```
330  WHILE Size>=Minimum
340    GOSUB Process
350    Size=Size/Scaling
360  END WHILE
```

WAIT

This statement will cause the computer to wait approximately the number of seconds specified before executing the next statement.

```
WAIT 3
WAIT Seconds/2
```

WINDOW

This statement is used to define the current unit-of-measure for graphics operations.

```
WINDOW -5,5,0,100
WINDOW Left,Right,Bottom,Top
```

WRITEIO

This statement writes an integer representation of the register-data to the specified hardware register on the specified interface.

```
WRITEIO 12,0;Set_pctl
WRITEIO Isc,Register;Reg_data
```

I/O Path Status and Control Registers

Example Statements

```
STATUS @Path,Register;Variable
CONTROL @Path,Register;Value
```

Status Register 0 0 = Invalid I/O path name
1 = I/O path assigned to a device
2 = I/O path assigned to a data file

If Assigned to a Device:

Status Register 1 Interface select code

Status Register 2 Number of devices

Status Register 3 1st device selector
If assigned to more than one device, the other device selectors are available starting in Status Register 4.

If 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 physical records

Status Register 4 Bytes per record = 256

Status Register 5 Current physical record

Status Register 6 Current byte within physical record

If 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 (bytes)

Status Register 5 Current record

Control Register 5 Set current record

Status Register 6 Current byte within record

Control Register 6 Set current byte within record

Status Register 7 EOF record

Control Register 7 Set EOF record

Status Register 8 Byte within EOF record

Control Register 8 Set byte within EOF record

Keyboard Status and Control Registers

Example Statements

```
STATUS 2,Register,Variable
CONTROL 2,Register,Value
```

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	Undefined
Control Register 2	Undefined
Status Register 3	Undefined
Control Register 3	Set auto-repeat interval. If 1 thru 255, repeat rate in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at power-on or SCRATCH A is 80 ms.)
Status Register 4	Undefined
Control Register 4	Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at power-on or SCRATCH A is 700 ms.)
Status Register 5	Undefined
Control Register 5	Undefined
Status Register 6	Undefined
Control Register 6	Undefined

Status Register 7

Most Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	INITIALIZE Timeout Interrupt Disabled	Reserved For Future Use	Reserved For Future Use	RESET Key Interrupt Disabled	Keyboard and Knob Interrupt Disabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Interrupt Status

Least Significant Bit

Control Register 7 (Set bit to disable)

Most 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

Interrupt Disable Mask

Least Significant Bit

Status Register 8 Keyboard language jumper
0 = US ASCII
1 = French
2 = German
3 = Swedish/Finnish
4 = Spanish
5 = Katakana

Control Register 8 Undefined

Status Register 9 Keyboard configuration jumper (0 thru 8)

Control Register 9 Undefined

Control Register 10 Undefined

Status Register 11 Reserved for future use

Control Register 11 Undefined

Status Register 12 "Pseudo-EOI for CTRL-E" flag

Control Register 12 Enable pseudo-EOI for CTRL-E if non-0

Status Register 13 Katakana flag

Control Register 13 Set Katakana if non-0

State at Last Knob Interrupt

Least Significant Bit							
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
SHIFT Key Pressed	CTRL Key Pressed	0	0	0	0	0	0
Value = 1	Value = 2	Value = 4	Value = 8	Value = 16	Value = 32	Value = 64	Value = 128

Status Register 10

Most Significant Bit

CRT Status and Control Registers

Example Statements

```
STATUS 1,Register;Variable
CONTROL 1,Register;Value
```

Status Register 0	Current PRINT position (X)
Control Register 0	Set PRINT position (X)
Status Register 1	Current PRINT position (Y)
Control Register 1	Set PRINT position (Y)
Status Register 2	Insert character mode
Control Register 2	Set insert character mode if non-0
Status Register 3	Number of lines in offscreen memory above top of CRT.
Control Register 3	Undefined
Status Register 4	Display functions mode
Control Register 4	Set display functions mode if non-0
Status Register 5	Undefined
Control Register 5	Undefined
Status Register 6	ALPHA ON flag
Control Register 6	Undefined
Status Register 7	GRAPHICS ON flag
Control Register 7	Undefined
Status Register 8	Display line position (X)
Control Register 8	Set display line position (X)
Status Register 9	Screenwidth (number of characters)
Control Register 9	Undefined
Status Register 10	Cursor enable flag
Control Register 10	Cursor enable; 0 = cursor not visible non-0 = cursor visible
Status Register 11	CRT character mapping flag
Control Register 11	Disable CRT character mapping if non-0

HP-IB Status and Control Registers

Status Register 0 Card identification = 1

Control Register 0 Reset interface if non-zero

Interrupt and DMA Status

Least Significant Bit

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
DMA Channel 0 Enabled	DMA Channel 1 Enabled	0	0	Hardware Interrupt Level Switches		Interrupt Requested	Interrrupts Enabled
Value = 1	Value = 2	Value = 4	Value = 8	Value = 16	Value = 32	Value = 64	Value = 128

Status Register 1

Most Significant Bit

Serial Poll Response Byte

Least Significant Bit

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Device Dependent Status						SRQ 1 = I did it 0 = I didn't	Device Dependent Status
Value = 1	Value = 2	Value = 4	Value = 8	Value = 16	Value = 32	Value = 64	Value = 128

Control Register 1

Most Significant Bit

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	Reserved For Future Use	Handshake In Progress	Interrupts Enabled	Reserved For Future Use
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

62

Control Register 2

Parallel Poll Response Byte

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1 = True	DIO7 1 = True	DIO6 1 = True	DIO5 1 = True	DIO4 1 = True	DIO3 1 = True	DIO2 1 = True	DIO1 1 = True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3

Controller Status and Address

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

63

Control Register 3

Set My Addrress

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 4

Interrupt Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

64

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 4 Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

Status Register 5

Interrupt Enable Mask

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

65

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 6 — Interface Status

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*Least-significant bit of last address recognized

Status Register 7 — Bus Control and Data Lines

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*Only if addressed to TALK, else not valid.
**Only if Active Controller, else not valid.

Interrupt Enable Register (ENABLE INTR) — Same as Status Register 5

Second Byte of Non-ASCII Key Codes

Character	Value	Key
!	33	STOP
#	35	CLR LN
$	36	ANY CHAR
%	37	CLR-END
(	40	SHIFT - TAB
)	41	TAB
*	42	INS LN
+	43	INS CHR
-	45	DEL CHR
/	47	DEL LN
0	48	k0
1	49	k1
2	50	k2
3	51	k3
4	52	k4
5	53	k5
6	54	k6
7	55	k7
8	56	k8
9	57	k9
<	60	←
=	61	RESULT
>	62	→
?	63	RECALL
@	64	SHIFT - RECALL
A	65	PRT ALL
B	66	BACK SPACE
C	67	CONTINUE
D	68	EDIT
E	69	ENTER
F	70	DISPLAY FCTNS

Character	Value	Key
G	71	SHIFT - →
H	72	SHIFT - ←
I	73	CLR I/O
J	74	Katakana Mode
K	75	CLR SCR
L	76	GRAPHICS
M	77	ALPHA
N	78	DUMP GRAPHICS
O	79	DUMP ALPHA
P	80	PAUSE
R	82	RUN
S	83	STEP
T	84	SHIFT - ↓
U	85	CAPS LOCK
V	86	↓
W	87	SHIFT - ↑
X	88	EXECUTE
Y	89	Roman Mode
[	91	CLR TAB
]	93	SET TAB
^	94	↑
a	97	k10
b	98	k11
c	99	k12
d	100	k13
e	101	k14
f	102	k15
g	103	k16
h	104	k17
i	105	k18
j	106	k19

Error Messages

1 Missing ROM or configuration error.
2 Memory overflow.
3 Line not found in current context.
4 Improper RETURN.
5 Improper context terminator.
6 Improper FOR...NEXT matching.
7 Undefined function or subprogram.
8 Improper parameter matching.
9 Improper number of parameters.
10 String type required.
11 Numeric type required.
12 Attempt to redeclare variable.
13 Array dimensions not specified.
14 OPTION BASE not allowed here.
15 Invalid bounds.
16 Improper dimensions.
17 Subscript out of range.
18 String overflow or substring error.
19 Improper value or out of range.
20 INTEGER overflow.
22 REAL overflow.
24 Trigonometric argument too large.
25 Absolute value of ASN or ACS argument is greater than 1.
26 Zero to non-positive power.
27 Negative base to non-integer power.
28 LOG or LGT of a non-positive number.
29 Illegal floating point number.
30 SQR of a negative number.

31 Division (or MOD) by zero.
32 String does not represent a valid number.
33 Improper argument for NUM or RPT$.
34 Referenced line not an IMAGE statement.
35 Improper image.
36 Out of data in READ.
38 TAB or TABXY not allowed here.
40 Improper REN command.
41 First line number greater than second line number.
46 No binary for STORE BIN or no program for SAVE.
47 Improper COM declaration.
49 Branch destination not found.
51 File not currently assigned.
52 Improper mass storage unit specifier.
53 Improper file name.
54 Duplicate file name.
55 Directory overflow.
56 File name is undefined.
58 Improper file type.
59 End of file found.
60 End of record found in random mode.
62 Protect code violation.
64 Mass storage media overflow.
66 INITIALIZE failed.
67 Improper mass storage parameter.
68 Syntax error occurred during GET.
72 Disc controller not found or improper address.
73 Improper device type in mass storage unit specifier.
76 Incorrect unit code in msus.

77 Attempt to purge an open file.

78 Improper mass storage volume label.

79 File open on target device.

80 Media changed or not in drive.

81 Mass storage hardware failure (or disc not turning).

82 Mass storage unit not present.

83 Write protected.

84 Record not found.

85 Media not initialized.

87 Record address error.

88 Read data error.

90 Mass storage system error.

100 Numeric image specifier for string item.

101 String image specifier for numeric item.

102 Numeric field specifier is too large.

103 Item has no corresponding image specifier.

105 Numeric field specifier is too small.

106 Exponent field specifier is too small.

107 Sign specifier missing from image.

117 Too many nested structures.

118 Too many structures in context.

120 Not allowed while program running.

121 Line not in main program.

122 Program is not continuable.

125 Program is not running.

126 Quote mark in unquoted string.

127 Improper sequence of KNOB statements.

128 Line too long during GET.

131 Improper non-ASCII keycode.

132 Keycode buffer overflow.

133 DELSUB of non-existent or busy subprogram.

134 Improper SCRATCH statement.

136 REAL underflow.

140 Too many symbols in the program.

141 Variable cannot be allocated.

142 Variable not allocated.

143 Reference to missing OPTIONAL parameter.

145 May not build COM at this time.

146 Duplicate label in context.

150 Improper interface select code or device selector.

152 Parity error.

153 Insufficient data for ENTER.

154 String greater than 32 767 bytes in ENTER.

155 Improper interface register number.

156 Improper expression type in list.

157 No ENTER terminator found.

158 Improper image specifier.

159 Numeric data not received.

160 Too many digits in number.

163 Interface not present.

165 Image specifier greater than dimensioned string length.

167 Interface status error.

168 I/O timeout.

170 I/O operation not allowed.

171 Illegal I/O addressing sequence.

172 Peripheral (PSTS) error. I/O device failure.

173 Active or system controller required.

174 Nested I/O prohibited.

177 Unidentified I/O path name.

178 Trailing punctuation in ENTER.

306 Interface card failure.

313 USART receive buffer overflow.

314 Receive buffer overflow.

315 Missing data transmit clock.

316 CTS false too long.

317 Lost carrier disconnect. DSR and/or DCD inactive too long.

318 No activity disconnect.

319 Connection not established.

325 Illegal databits parity combination.

326 Register address out of range.

327 Register value out of range.

345 CASE expression type mismatch.

347 Structures improperly matched.

700 Improper plotter specifier.

702 CRT graphics hardware missing.

704 Upper bound less than lower bound.

705 VIEWPORT or CLIP beyond hard clip limits.

902 Must delete entire context.

903 No room to renumber.

906 SUB or DEF FN not allowed here.

909 May not replace SUB or DEF FN.

910 Identifier not found in this context.

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.

935 Identifier is too long: 15 characters max.

936 Unrecognized character.

937 Invalid OPTION BASE.

939 OPTIONAL appears twice.

940 Duplicate formal parameter name.

942 Invalid I/O path name.

943 Invalid function name.

946 Dimensions are inconsistent.

947 Invalid array bounds.

948 Multiple assignment prohibited.

949 This symbol not allowed here.

950 Must be a positive integer.

951 Incomplete statement.

961 CASE expression type mismatch.

962 Programmable only: cannot be executed from the keyboard.

963 Command only: cannot be stored as a program line.

977 Statement is too complex.

980 Too many symbols in this context.

982 Too many subscripts: 6 dimensions max.

983 Wrong type or number of parameters.

985 Invalid quoted string.

987 Invalid line number: only integers 1 thru 32 766 allowed.

ASCII TABLE

ASCII Char.	EQUIVALENT FORMS			
	Binary	Oct	Hex	Dec
NUL	00000000	000	00	0
SOH	00000001	001	01	1
STX	00000010	002	02	2
ETX	00000011	003	03	3
EOT	00000100	004	04	4
ENQ	00000101	005	05	5
ACK	00000110	006	06	6
BEL	00000111	007	07	7
BS	00001000	010	08	8
HT	00001001	011	09	9
LF	00001010	012	0A	10
VT	00001011	013	0B	11
FF	00001100	014	0C	12
CR	00001101	015	0D	13
SO	00001110	016	0E	14
SI	00001111	017	0F	15
DLE	00010000	020	10	16
DC1	00010001	021	11	17
DC2	00010010	022	12	18
DC3	00010011	023	13	19
DC4	00010100	024	14	20
NAK	00010101	025	15	21
SYN	00010110	026	16	22
ETB	00010111	027	17	23
CAN	00011000	030	18	24
EM	00011001	031	19	25
SUB	00011010	032	1A	26
ESC	00011011	033	1B	27
FS	00011100	034	1C	28
GS	00011101	035	1D	29
RS	00011110	036	1E	30
US	00011111	037	1F	31

ASCII Char.	EQUIVALENT FORMS			
	Binary	Oct	Hex	Dec
space	00100000	040	20	32
!	00100001	041	21	33
"	00100010	042	22	34
#	00100011	043	23	35
$	00100100	044	24	36
%	00100101	045	25	37
&	00100110	046	26	38
'	00100111	047	27	39
(	00101000	050	28	40
)	00101001	051	29	41
*	00101010	052	2A	42
+	00101011	053	2B	43
,	00101100	054	2C	44
-	00101101	055	2D	45
.	00101110	056	2E	46
/	00101111	057	2F	47
0	00110000	060	30	48
1	00110001	061	31	49
2	00110010	062	32	50
3	00110011	063	33	51
4	00110100	064	34	52
5	00110101	065	35	53
6	00110110	066	36	54
7	00110111	067	37	55
8	00111000	070	38	56
9	00111001	071	39	57
:	00111010	072	3A	58
;	00111011	073	3B	59
<	00111100	074	3C	60
=	00111101	075	3D	61
>	00111110	076	3E	62
?	00111111	077	3F	63

ASCII Char.	EQUIVALENT FORMS			
	Binary	Oct	Hex	Dec
@	01000000	100	40	64
A	01000001	101	41	65
B	01000010	102	42	66
C	01000011	103	43	67
D	01000100	104	44	68
E	01000101	105	45	69
F	01000110	106	46	70
G	01000111	107	47	71
H	01001000	110	48	72
I	01001001	111	49	73
J	01001010	112	4A	74
K	01001011	113	4B	75
L	01001100	114	4C	76
M	01001101	115	4D	77
N	01001110	116	4E	78
O	01001111	117	4F	79
P	01010000	120	50	80
Q	01010001	121	51	81
R	01010010	122	52	82
S	01010011	123	53	83
T	01010100	124	54	84
U	01010101	125	55	85
V	01010110	126	56	86
W	01010111	127	57	87
X	01011000	130	58	88
Y	01011001	131	59	89
Z	01011010	132	5A	90
[	01011011	133	5B	91
\	01011100	134	5C	92
]	01011101	135	5D	93
^	01011110	136	5E	94
_	01011111	137	5F	95

ASCII Char.	EQUIVALENT FORMS			
	Binary	Oct	Hex	Dec
`	01100000	140	60	96
a	01100001	141	61	97
b	01100010	142	62	98
c	01100011	143	63	99
d	01100100	144	64	100
e	01100101	145	65	101
f	01100110	146	66	102
g	01100111	147	67	103
h	01101000	150	68	104
i	01101001	151	69	105
j	01101010	152	6A	106
k	01101011	153	6B	107
l	01101100	154	6C	108
m	01101101	155	6D	109
n	01101110	156	6E	110
o	01101111	157	6F	111
p	01110000	160	70	112
q	01110001	161	71	113
r	01110010	162	72	114
s	01110011	163	73	115
t	01110100	164	74	116
u	01110101	165	75	117
v	01110110	166	76	118
w	01110111	167	77	119
x	01111000	170	78	120
y	01111001	171	79	121
z	01111010	172	7A	122
{	01111011	173	7B	123
|	01111100	174	7C	124
}	01111101	175	7D	125
~	01111110	176	7E	126
DEL	01111111	177	7F	127