HP Series 200 Computers

BASIC 2.0
Condensed
Reference
with
Extensions 2.1

**HEWLETT
PACKARD**

**HEWLETT
PACKARD**

BASIC 2.0
Condensed
Reference
with
Extensions 2.1

Manual Part No. 09826-90051

Printing History

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. Minor corrections which are made during normal reprints do not cause the date to change. New editions will incorporate all material updated since the previous edition. An updated page will have a revision date at the bottom of the page and a vertical bar in the margin opposite the change. The manual part number changes when extensive tecnical changes are incorporated.

Table of Contents

What Version?

As of December 1982, the compatibility of system software is indicated by the revision number included in the software's name. The revision number has a digit to the left and right of a decimal point. The digit to the left of the decimal point indicates the "level" of the system. System software files are compatible with one another if they have matching "level" numbers. For example, a binary program named "ABC2.3" would be compatible with BASIC 2.0 because they both have "2" as the leading digit. The digit to the right of the decimal point indicates the revision number of that particular piece of code. Thus, "XYZ2.1" would be an updated version of "XYZ2.0" and replaces the old version.

The language's version is displayed on the CRT at power-up. BASIC 2.0 gives the following message:

    BASIC Ready 2.0

If the numeric version code (2.0 in the example) is not shown, then BASIC 1.0 is resident. Loading a binary extension with LOAD BIN produces a message similar to the following, where AP2.0 is the binary program that is loaded:

    BASIC Ext. AP2.0 copyright HP 1982

Because LIF conventions do not allow periods in file names, an underscore character is used when naming these files. Therefore, the "AP2.0" module is stored in a file named "AP2_0".

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 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 zero, 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 remainder 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.)
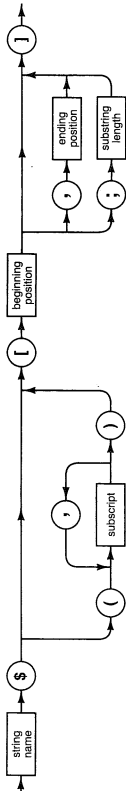
Numeric Expressions

4

5
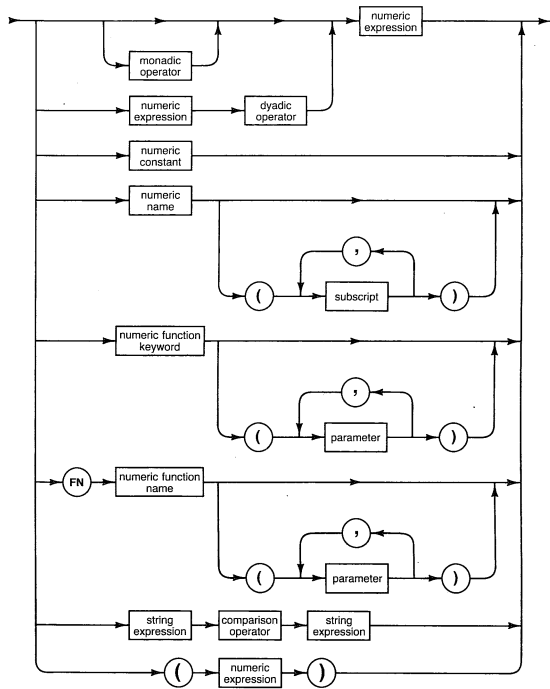
String Expressions
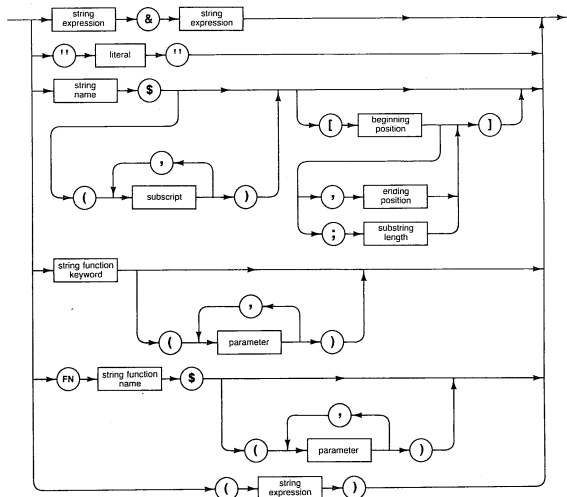
Simplified Graphics Mapping

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 coordinate system is adjusted accordingly (anisotropic 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).
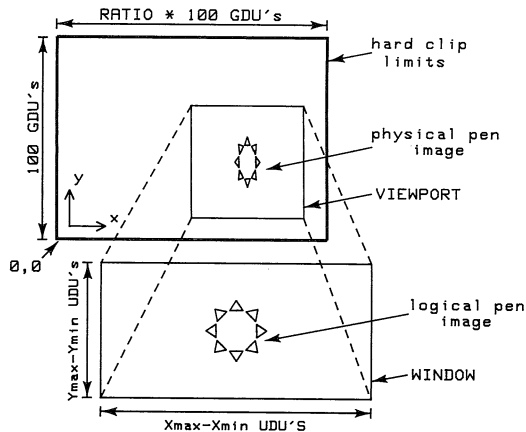
Simplified Color Model

PLOTTER IS...;COLOR MAP enables the color map mode. The color map must be enabled to use the SET PEN statement.

PEN selects a color to be used for subsequent lines, characters, and edges.

SET PEN defines the color displayed on the CRT when a given pen selector is used. Secondary keywords COLOR and INTENSITY select the color model to be used (see below).

AREA defines and/or selects the color to be used for filling polygons, array plots, and rectangles. COLOR and INTENSITY select the color model to be used (see below). Secondary keyword PEN selects a pen to be used, instead of dithered colors.

...COLOR selects the HSL color model. The HSL models uses coordinates in a cylindrical color space, with the axes being *Hue*, *Saturation*, and *Luminosity*. See the drawing on the next page.

...INTENSITY selects the RGB color model. The RGB model uses coordinates in a cartesian color space, with the axes being *Red*, *Green*, and *Blue*.

HSL Color Space
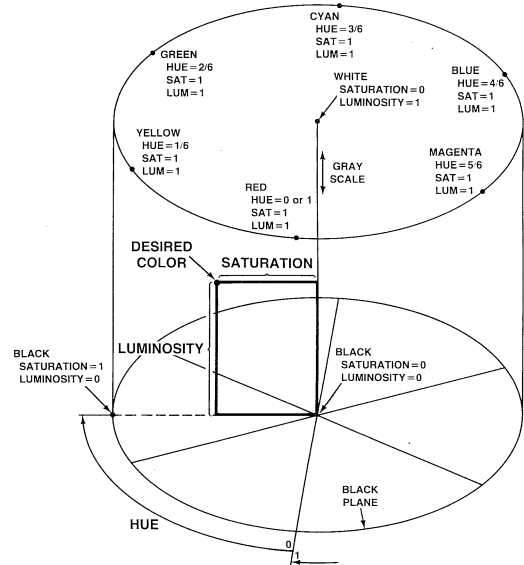
Glossary

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 (MAT parameters are a special case). To specify a single element of an array, subscripts are placed in parentheses after the array name. Each subscript must be not 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 expension 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, a CSUB, 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. (Also see the CSUB keyword.)

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
```

ABORTIO

Minimum requirement AP2.0. This statement terminates a TRANSFER which is taking place through an I/O path. The I/O path named in the statement must be assigned to a file or device, not a buffer.

```
ABORTIO @Isc
IF Stop_flag THEN ABORTIO @Device
```

ABS

This function returns the absolute value of its 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.
See DEALLOCATE.

```
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. The statement has no effect on the contents of the alpha memory, it just controls whether it is displayed or not.

```
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
```

AREA

Minimum requirement GRAPH2.1. This statement defines or selects an area fill color. See SET PEN for a color table.

```
AREA COLOR Hue,Saturation,Luminosity
AREA INTENSITY Red(I),Green(I),Blue(I)
AREA PEN 3
```

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, group of devices, a mass storage file, or a buffer. Without AP2.0, the only attribute allowed is FORMAT.

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

Minimum requirement AP2.0:

```
ASSIGN @Buffer TO BUFFER String$
ASSIGN @Buf TO BUFFER Array(*)
ASSIGN @Buff2 TO BUFFER [48000]
ASSIGN @Path TO File$;RETURN Outcome
ASSIGN @Port TO Gpio;WORD
ASSIGN @Dest TO Rs_232;EOL My$ DELAY .1,
  CONVERT OUT BY INDEX Out$,PARITY ODD
```

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

BASE

Minimum requirement AP2.0. This function returns the lower subscript bound of a dimension of an array.

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

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"
```

BREAK

Minimum requirement AP2.0. This statement directs a serial interface to send a break sequence.

```
BREAK 9
BREAK @Datacomm
```

C

CALL

This statement transfers program execution to the specified SUB (or CSUB) 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
```

18

CAT

This statement lists the contents of the mass storage media's directory. With AP2.0, portions of the directory may be selected, or information about a PROG file may be requested.

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

Minimum requirement AP2.0:

```
CAT TO String$;NO HEADER
CAT Prog_file$
CAT;SELECT "C",SKIP First,COUNT Retn_var
```

CHANGE

Minimum requirement AP2.0. This program editing command allows search-and-replace operations.

```
CHANGE "Old Text" TO "New Text"
CHANGE "Row" TO "Column" IN 2560,3310
CHANGE "November" TO "December";ALL
```

CHECKREAD

Minimum requirement AP2.0. This statement enables or disables optional read-after-write verification of data sent to mass storage media.

```
CHECKREAD ON
CHECKREAD OFF
```

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 is in the back of this book.

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

19

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)
```

Minimum requirement AP2.0:

```
COM Buffy$[1024] BUFFER
```

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 duplicating of discs.

```
COPY Old_file$ TO New_file$
COPY "MY_FILE" TO "TEMP<pc>:HP9895,700,1"
COPY ":INTERNAL" TO ":INTERNAL,4,1"
COPY Int_disc$ TO Ext_disc$
```

COPYLINES

Minimum requirement AP2.0. This program editing command copies contiguous program lines from one location to another. If only one line identifier is specified, only that line is copied. See MOVELINES.

```
COPYLINES 1200 TO 3255
COPYLINES 10,120 TO 500
COPYLINES Label1,Label2 TO Label3
```

COS

This function returns the cosine of the argument.

```
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$&Msus$,Sectors
```

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
```

CRT

Minimum requirement AP2.0. This function returns 1, the device selector of the CRT.

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

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
```

CSUB

CSUB statements are created in Pascal using a special CSUB preparation utility (09800-10640). When viewed in BASIC's edit mode, they look like SUB statements. CSUBs cannot be edited from BASIC.

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(,)"
```

22

DATE

Minimum requirement AP2.0. This function converts a formatted date string into a numeric value in seconds.

```
PRINT DATE("30 MAY 1983")
Days=(DATE(Day1$)-DATE(Day2$)) DIV 86400
```

DATE$

Minimum requirement AP2.0. This function formats a number of seconds into a string representing the formatted date (DD MMM YYYY).

```
PRINT DATE$(TIMEDATE)
Day1$=DATE$(Event1)
```

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

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

Minimum requirement AP2.0:

```
DEF FNExtract(Buf$ BUFFER,Pointer)
```

23

DEG

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

```
DEG
```

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 (or CSUB) 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
```

DET

Minimum requirement AP2.0. This function returns the determinant of a matrix.

```
Last_det=DET
PRINT DET(Matrix)
```

DIGITIZE

Minimum requirement GRAPH2.0. This statements inputs the X and Y coordinates of a digitized point from the locator specified by GRAPHICS INPUT IS.

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

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)
```

Minimum requirement AP2.0:

```
DIM Buff$[512] BUFFER,Array(500) BUFFER
```

DISABLE

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

```
DISABLE
```

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
```

DOT

Minimum requirement AP2.0. This function returns the inner (dot) product of two numeric vectors.

```
Res=DOT(Vec1,Vec2)
PRINT DOT(A,B)
```

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 a printing device. With GRAPH2.0, the source can be a 98627A.

```
DUMP ALPHA
DUMP ALPHA #701
DUMP GRAPHICS #Device
```

Minimum requirement GRAPH2.0:

```
DUMP GRAPHICS Color_source
DUMP GRAPHICS 28 TO #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
```

DVAL

Minimum requirement AP2.0. This function converts a binary, octal, decimal, or hexidecimal character representation into a REAL whole number.

```
Number=DVAL(String$,Radix)
PRINT DVAL("FF5900",16)
```

DVAL$

Minimum requirement AP2.0. This function converts a numeric quantity into a character string representing a binary, octal, decimal, or hexidecimal base.

```
String$=DVAL$(Number,Radix)
PRINT DVAL$(Count MOD 256,2)
```

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*. With AP2.0, typing-aid keys can be edited.

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

Minimum requirement AP2.0:

```
EDIT KEY 5
```

ENABLE

This statement re-enables all event-initiated branches which were suspended by DISABLE.

```
ENABLE
```

ENABLE INTR

This statement enables the specified interface to generate an interrupt which can cause event-initiated 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.

```
END
```

ENTER

This statement is used to input data from a specified source 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]
```

ERRDS

Minimum requirement AP2.0. This function returns the device selector of the I/O resource involved in the most recent I/O error.

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

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(Label2) THEN Other
```

ERRM$

Minimum requirement AP2.0. This function returns the text of the error message associated with the most recent program execution error.

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

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

FIND

Minimum requirement AP2.0. This program editing command allows searching for specified text.

```
FIND "SUB Printer"
FIND "Cost=" IN 1550
FIND "Target" IN Label1,Label2
```

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
```

FRACT

Minimum requirement AP2.0. This function returns the "fractional part" of its argument. For all X, $X = INT(X) + FRACT(X)$.

```
PRINT FRACT(17/3)
Right_part=FRACT(Both_parts)
```

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.

```
FRAME
```

G

GCLEAR

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

```
GCLEAR
```

GESCAPE

Minimum requirement GRAPH2.1. This statement is used for communicating device-dependent graphics information. The type, size, and shape of the arrays must be appropriate for the requested operation.

```
GESCAPE Device,Operation;Return_array(*)
GESCAPE 28,5
GESCAPE 3,2;Color_map(*)
GESCAPE Ds,Op,Send_array(*);Receive(*)
```

Operation Selector	Summary of Request
1	Number of color-map entries
2	Color map values
3	Hard clip values
4	Normal drawing mode
5	Alternate drawing mode

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_pros$,180,10
```

GINIT

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

```
GINIT
```

GLOAD

This statement loads the contents of an INTEGER array into the graphics frame buffer. Also see GSTORE.

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

Minimum requirement GRAPH2.0:

```
GLOAD 28,Hp98627(*)
```

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
```

32

GRAPHICS

This statement turns the graphics display on or off. The 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
```

GRAPHICS INPUT IS

Minimum requirement GRAPH2.0. This statement defines the device to be used for subsequent DIGITIZE and READ LOCATOR statements.

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

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 frame buffer into an INTEGER array. The size of the array depends upon the version of BASIC loaded and the size of the frame buffer. See GLOAD.

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

Minimum requirement GRAPH2.0:

```
GSTORE 28,Hp98627(*)
```

33

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 = ",SDDDDE,3(XX,ZZ)
IMAGE #,B
```

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

34

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.
H	Like K, except that the European number format is used. Thus, comma is the radix indicator and period is a numeric item terminator. (Minimum requirement AP2.0)
$-$H	Same as $-$K for strings; same as H for numbers. (Minimum requirement AP2.0)
S	Same action as D.
M	Same action as D.
D	Demands a character. Non-numerics are accepted to fill the character count. Blanks are ignored; other non-numerics are delimiters.
Z	Same action as D.
*	Same action as D. (Minimum requirement AP2.0)

35

Specifier	Meaning
,	Same action as D.
R	Like D, R demands a character. When R is used in a numeric image, the number builder uses the European number format. See the H specifier. (Minimum requirement AP2.0)
E	Same action as 4D.
ESZ	Same action as 3D.
ESZZ	Same action as 4D.
ESZZZ	Same action as 5D.
A	Demands a string character. Any character received is placed in the string.
X	Skips a character.
literal	Skips one character for each character in the literal.
B	Demands one byte. The byte becomes a numeric quantity.
W	Demands one word, which is interpreted as a 16-bit, two's-complement integer. The first byte entered on an 8-bit interface is the most-significant byte. The word is always entered in a single operation on a 16-bit interface. Files, string variables, and buffers are treated as 8-bit sources. Pad bytes are entered if necessary to achieve word-boundary alignment.
Y	Like W, except that pad bytes are not entered. Does not override a BYTE attribute used with a 16-bit interface. (Minimum requirement AP2.0)
#	Statement is terminated when the last ENTER item is terminated. EOI and line-feed are item terminators, and early termination is not allowed.
%	Same as # except EOI is a statement terminator. Early termination is allowed if the current item is satisfied.

Specifier	Meaning
+	Specifies that an END indication is required with the last character of the last item to terminate the ENTER statement. Line-feeds are not statement terminators. (Minimum requirement AP2.0)
−	Specifies that a line-feed terminator is required as the last character of the last item to terminate the ENTER statement. EOI is ignored, and other END indications cause an error. (Minimum requirement AP2.0)
/	Demands a new field; skips all characters to the next line-feed. EOI is ignored.
L	Ignored for ENTER.
@	Ignored for ENTER.

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

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. (Minimum requirement AP2.0)
−H	Same as H. (Minimum requirement AP2.0)
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.

Specifier	Meaning
Z	Same as D, except that leading zeros are output.
*	Like D, except that asterisks are output instead of leading zeros. (Minimum requirement AP2.0)
,	Outputs a decimal point radix indicator.
R	Outputs a comma radix indicator. (Minimum requirement AP2.0)
E	Same as ESZZ.
ESZ	Outputs an E, a sign, and a one-digit exponent.
ESZZ	Outputs an E, a sign, and a two-digit exponent.
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 string. If the image specifier is exhausted before the string, the remaining characters are not sent.
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.
W	Outputs a 16-bit word as a two's-complement integer. On an 8-bit interface, the most-significant byte is sent first. The word is always sent in a single operation on a 16-bit interface. Files, string variables, and buffers are treated as 8-bit destinations. Pad bytes are sent if necessary to achieve word-boundary alignment.
Y	Like W, except that pad bytes are not sent. Does not override a BYTE attribute used with a 16-bit interface. (Minimum requirement AP2.0)

Specifier	Meaning
#	Suppresses automatic output of the EOL (End-Of-Line) sequence following the last output item.
%	Ignored in output images.
+	Changes the automatic EOL sequence to a single carriage-return. (Minimum requirement AP2.0)
−	Changes the automatic EOL sequence to a single line-feed. (Minimum requirement AP2.0)
/	Outputs a carriage-return and a line-feed.
L	Outputs the current EOL sequence. The default is CR/LF. With AP2.0, the sequence may be redefined with ASSIGN or PRINTER IS.
@	Outputs a form-feed.

IMOVE

This statement updates the logical pen position, by adding the X and Y displacements to the current logical pen position.

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

INDENT

Minimum requirement AP2.0. This program editing command indents a program to reflect its structure and nesting.

```
INDENT
INDENT 8,4
```

INITIALIZE

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

```
INITIALIZE ":INTERNAL"
INITIALIZE ":HP9895,700,1"
INITIALIZE Disc$,Interleave
```

With AP2.1:

```
INITIALIZE ":BUBBLE,30"
INITIALIZE ":MEMORY,0,2",Sectors
```

INPUT

This statement is used to assign keyboard input to program variables. Also see LINPUT.

```
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. For all X, $X = INT(X) + FRACT(X)$.

```
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)
```

Minimum requirement AP2.0:

```
INTEGER Buf(2000) BUFFER
```

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
```

Minimum requirement GRAPH2.1:

```
IPLOT Array(*)
IPLOT Shape(*),FILL,EDGE
```

IVAL

Minimum requirement AP2.0. This function converts a binary, octal, decimal, or hexidecimal character representation into an INTEGER.

```
Number=IVAL(String$,Radix)
PRINT IVAL("FE56",16)
```

IVAL$

Minimum requirement AP2.0. This function converts an INTEGER into a binary, octal, decimal, or hexidecimal string representation.

```
String$=IVAL$(Number,Radix)
PRINT IVAL$(Count MOD 256,2)
```

K

KBD

This function returns 2, the device selector of the keyboard.

```
STATUS KBD;Kbd_status
OUTPUT KBD;Clear$;
```

KBD$

This function returns the contents of the keyboard buffer and then clears it. 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. See IMAGE for specifier descriptions.

```
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
```

42

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"
```

LEXICAL ORDER IS

Minimum requirement AP2.0. This statement defines the collating sequence for all string operations.

```
LEXICAL ORDER IS FRENCH
LEXICAL ORDER IS Lex_table(*)
```

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 all subsequent lines. CRT line types are shown below.

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


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

43
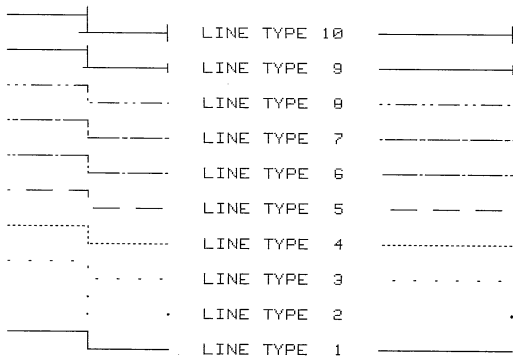
LINPUT

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. With AP2.0, the typing-aid definitions can be listed.

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

Minimum requirement AP2.0:

```
LIST KEY #Printer
```

LOAD

This statement loads programs, binary programs, or typing-aid definitions into memory. Also see GET.

```
LOAD File_name$
LOAD "TEMP:INTERNAL,4,1",Run_line
LOAD BIN "AP2_0"&Msus$
```

Minimum requirement AP2.0:

```
LOAD KEY "AIDS"
```

LOADSUB

This statement loads BASIC subprograms from a file of type PROG into memory. See STORE.

```
LOADSUB ALL FROM "George"
LOADSUB ALL FROM Name$&Msus$
```

Minimum requirement AP2.0:

```
LOADSUB FROM "My_subs:HP9895,700,1"
LOADSUB Fred FROM Name$
LOADSUB FNConvert$ FROM "STRFUNCTS"
```

LOCAL

This statement returns all specified devices to their local state.

```
LOCAL @Dvm
LOCAL 728
LOCAL Isc
```

LOCAL LOCKOUT

This HP-IB statement sends the LLO (local lockout) message, which prevents local (front panel) control of devices in the remote state.

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

LOG

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

```
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 (nonzero). 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
```
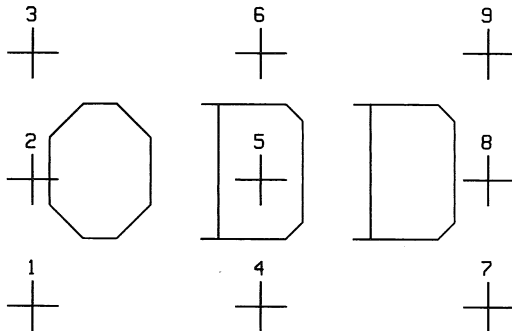

LWC$

Minimum requirement AP2.0. This function returns a string formed by replacing any lowercase characters with their corresponding uppercase character.

```
Lower$=LWC$(Mixed$)
IF LWC$(Answer$)="y" THEN True_test
```

M

MASS STORAGE IS

This statement specifies the system mass storage device. It may be abbreviated as MSI when typed from the keyboard. Note that some *device type* specifiers work for more than one model of disc drive.

```
MASS STORAGE IS ":INTERNAL"
MASS STORAGE IS ":HP9895,700,1"
MASS STORAGE IS Msus$
```

Minimum requirement AP2.1:

```
MASS STORAGE IS ":BUBBLE,30"
MASS STORAGE IS ":MEMORY,0,2"
```

Minimum Requirement	Device Types Added
BASIC 1.0	INTERNAL (no device selector)
BASIC 2.0	INTERNAL (with device selector) HP9895 HP82901 HP82902 HP8290X
AP2.0	CS80 HP9121 HP9133 HP9134 HP9135 HP913X HP9885
SRM2.0	REMOTE
AP2.1	MEMORY BUBBLE EPROM

MAT

Minimum requirement AP2.0. This statement can be use to initialize string and numeric arrays, copy string and numeric arrays, and perform operations on numeric arrays.

```
MAT Array= A*(Ref+1/3)
MAT String$= (RPT$(" ",80))
MAT Clone= Parent
MAT A= Array1<>Array2
MAT Vector= CSUM(Matrix)
MAT Identity= IDN
MAT Inverse= INV(Matrix)
```

MAT REORDER

Minimum requirement AP2.0. This statement reorders elements in an array according to the subscript list in a vector.

```
MAT REORDER Array BY Vector,Dimension
MAT REORDER Lines$ BY New_order
```

MAT SORT

Minimum requirement AP2.0. This statement sorts an array along one dimension according to lexical or numerical order.

```
MAT SORT Array(Tag,*)
MAT SORT Vals(1,*,3),(2,*,5) DES
MAT SORT Start_vec(*) TO End_vec
MAT SORT String$(*,2)[1;3] TO Order
```

MAX

Minimum requirement AP2.0. This function returns a value equal to the greatest value in the list of arguments. Each element of an array is considered a separate value.

```
Biggest=MAX(Elements(*))
PRINT MAX(Item1,17,Total/3)
Result=MAX(Floor,MIN(Ceiling,Argument))
```

MIN

Minimum requirement AP2.0. This function returns a value equal to the least value in the list of arguments. Each element of an array is considered a separate value.

```
Smallest=MIN(Elements(*))
PRINT MIN(Item1,17,Total/3)
```

MOD

This operator returns the remainder of a 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
```

MOVELINES

Minimum requirement AP2.0. This program editing statement moves contiguous program lines from one location to another. If only one line identifier is specified, only that line is moved. See COPYLINES.

```
MOVELINES 1200 TO 3250
MOVELINES 10,440 TO 540
MOVELINES Label1,Label2 TO Label3
```

N

NOT

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

```
Invert_flag=NOT Std_device
IF NOT My_job THEN Sleep
```

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 CYCLE

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON CYCLE statement.

```
OFF CYCLE
```

OFF DELAY

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON DELAY statement.

```
OFF DELAY
```

OFF END

This statement cancels event-initiated branches previously defined and enabled by an ON END statement. Subsequent EOR and EOF conditions generate an error.

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

OFF EOR

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON EOR statement.

```
OFF EOR @Device
OFF EOR @File
```

OFF EOT

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON EOT statement.

```
OFF EOT @Device
OFF EOT @File
```

OFF ERROR

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

```
OFF ERROR
```

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. Subsequent keypresses are sent to the operating system in the normal manner.

```
OFF KBD
```

OFF KEY

This statement cancels event-initiated branches previously defined and enabled by an ON KEY statement. Without AP, subsequent softkey presses cause beeps. With an AP, the action of subsequent softkey presses depends upon the typing-aid definitions.

```
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. Subsequent use of the knob results in normal scrolling or cursor movement.

```
OFF KNOB
```

OFF SIGNAL

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON SIGNAL statement.

```
OFF SIGNAL
OFF SIGNAL Sig_number
```

OFF TIME

Minimum requirement AP2.0. This statement cancels event-initiated branches previously defined and enabled by an ON TIME statement.

```
OFF TIME
```

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 CYCLE

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken each time the specified number of seconds has elapsed.

```
ON CYCLE Seconds,Priority CALL Sub_name
ON CYCLE Max_time RECOVER Backup
ON CYCLE 3600,3 GOTO 1200
```

ON DELAY

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken after the specified number of seconds has elapsed.

```
ON DELAY Seconds,Priority CALL Sub_name
ON DELAY 3 GOTO 5710
ON DELAY Max_time,4 GOSUB No_operator
```

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 @File RECOVER Next_file
ON END @Path CALL Sub_name
```

ON EOR

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken when a end-of-record is encountered during a TRANSFER.

```
ON EOR @Path,Priority CALL Sub_name
ON EOR @Device GOTO 1500
ON EOR @File,2 GOSUB Label1
```

ON EOT

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken when the last byte is transferred by a TRANSFER statement.

```
ON EOT @Path,Priority CALL Sub_name
ON EOT @Device GOTO 1500
ON EOT @File,2 GOSUB Label1
```

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 RECOVER Crash
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,4 GOSUB Service
ON INTR Isc,Priority CALL Sub_name
```

ON KBD

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

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

ON KEY

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

```
ON KEY 0 GOTO 150
ON KEY Number,Priority GOSUB Label1
ON KEY 10 LABEL A$ RECOVER 770
ON KEY 5 LABEL "Print",3 CALL Report
```

ON KNOB

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

```
ON KNOB ,1 GOTO 250
ON KNOB Seconds,Priority CALL Sub_name
ON KNOB 1/2,4 GOSUB Label1
```

ON SIGNAL

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken when a SIGNAL statement is executed using the same signal selector.

```
ON SIGNAL Selector,Priority CALL Sub_name
ON SIGNAL RECOVER Trap
ON SIGNAL 8 GOTO 770
```

ON TIME

Minimum requirement AP2.0. This statement defines and enables an event-initiated branch to be taken when the clock reaches a specified time.

```
ON TIME Seconds,Priority CALL Sub_name
ON TIME TIMEDATE+3600 GOTO 1440
ON TIME Alarm,15 GOSUB Label3
```

ON TIMEOUT

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

```
ON TIMEOUT 7,4 GOTO 770
ON TIMEOUT Isc,Seconds CALL Sub_name
ON TIMEOUT 12,1/2 RECOVER Label1
```

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

PASS CONTROL

Minimum requirement AP2.0. This statement passes Active Controller capability to a specified HP-IB device.

```
PASS CONTROL 719
PASS CONTROL @Device
```

PAUSE

This statement suspends program execution. See CONT.

```
PAUSE
```

PEN

This statement selects a pen on the current plotting device.

```
PEN -1
PEN Select
```

For monochrome CRTs:

> PEN>0 draws
> PEN = 0 complements
> PEN<0 erases

For Model 36C with GRAPH2.1, the default color-mapped pen colors are:

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

For non-color-mapped mode, only pens −7 thru +7 are used, and PEN 0 complements.

PENUP

This statement lifts the pen on the current plotting device.

```
PENUP
```

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. The current angle mode is used.

```
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

Minimum requirement GRAPH2.1:

```
PLOT Array(*)
PLOT Shape(*),FILL,EDGE
```

The plotting array must be 2-column or 3-column. 2-column arrays contain X and Y coordinates and assume a pen control of +1. The interpretation of a 3-column array is shown in the following table.

Col 1	Col 2	Col 3	Meaning
X	Y	<-2	Like −1 or −2
X	Y	−2	Pen up before move
X	Y	−1	Pen down before move
X	Y	0	Pen up after move
X	Y	1	Pen down after move
X	Y	2	Pen up after move
Pen #	-	3	Select pen
Line Type	Repeat	4	Select line type
Color	-	5	Color value for FILL
-	-	6	Start polygon with FILL
-	-	7	End polygon
-	-	8	End of array data
-	-	9	No-op
-	-	10	Start polygon with EDGE
-	-	11	Start polygon w/ FILL & EDGE
-	-	12	FRAME
Pen #	-	13	Area pen select
Red	Green	14	Color value for FILL
Blue	-	15	Color value for FILL
-	-	>15	Ignored

PLOTTER IS

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

```
PLOTTER IS Device,Id$
PLOTTER IS 705,"HPGL"
```

Minimum requirement GRAPH2.0:

```
PLOTTER IS 28,"98627A;US TV"
PLOTTER IS Ds,"98627A;EURO STD"
```

Minimum requirement GRAPH2.1:

```
PLOTTER IS CRT,"INTERNAL";COLOR MAP
```

POLYGON

Minimum requirement GRAPH2.1. This statement draws all or part of a closed regular polygon.

```
POLYGON Radius,Total_sides,Drawn_sides
POLYGON -Size,5,FILL,EDGE
```

POLYLINE

Minimum requirement GRAPH2.1. This statement draws all or part of an open regular polygon.

```
POLYLINE Radius,Total_sides,Drawn_sides
POLYLINE -Size,5
```

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 RESPONSE

Minimum requirement AP2.0. This statement defines the computer's response to a Parallel Poll on an HP-IB interface.

```
PPOLL RESPONSE Isc;Service
PPOLL RESPONSE @Hpib;1
```

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. The PRINTALL device is 1 (the CRT) at power-on and after SCRATCH A.

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

Minimum requirement AP2.0:

```
PRINTALL IS 707;EOL My$ DELAY .5
PRINTALL IS Device;WIDTH 120,EOL My$ END
```

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
```

Minimum requirement AP2.0:

```
PRINTER IS 707;EOL My$ DELAY .5
PRINTER IS Device;WIDTH 120,EOL My$ END
```

PROTECT

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

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

PROUND

Minimum requirement AP2.0. This function returns the value of the argument rounded to a specified power-of-ten.

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

PRT

Minimum requirement AP2.0. This function returns 701, the factory-set device selector for an external printer.

```
PRINTER IS PRT
OUTPUT PRT;Text$
```

PURGE

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

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

62

R

RAD

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

```
RAD
```

RANDOMIZE

This statement selects a seed for the RND function.

```
RANDOMIZE
RANDOMIZE Old_seed*PI
```

RANK

Minimum requirement AP2.0. This function returns the number of dimensions in an array.

```
Dimensions=RANK(Array$)
IF RANK(A)=2 THEN PRINT "A is a matrix"
```

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)
```

READ LOCATOR

Minimum requirement GRAPH2.0. This statement samples the locator device without waiting for a digitize operation. See GRAPHICS INPUT IS.

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

63

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)
```

Minimum Requirement AP2.0:

```
REAL Buf(100) BUFFER
```

RECTANGLE

Minimum requirement GRAPH2.1. This statement draws a rectangle.

```
RECTANGLE Width,Height
RECTANGLE 4,-6,FILL,EDGE
```

REDIM

Minimum requirement AP2.0. This statement changes the subscript range of previously dimensioned arrays.

```
REDIM Array(New_lower:New_upper)
REDIM String$(A,B,C)
```

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. The default for both parameters is 10.

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

Minimum requirement AP2.0:

```
REN 261,1 IN 260,Label2
```

RENAME

This statement changes a file's name in a 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
```

REQUEST

Minimum requirement AP2.0. This statement is used to send a Service Request (SRQ) on an HP-IB when the computer is non-active controller. To request service, the response must have bit 6 set.

```
REQUEST Isc;Response
REQUEST @Hpib;Bit_6+Bit_0
```

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

RESET

Minimum requirement AP2.0. This statement resets an interface or the pointers of either a mass storage file or a buffer.

```
RESET 20
RESET Hpib
RESET @Buffer
```

RESTORE

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

```
RESTORE
RESTORE Third_array
```

RE-STORE

This statement creates a file and stores either an internal form of the BASIC program, normal binary programs, or typing-aid key definitions. If the specified file already exists, the old entry is purged after the new file is successfully stored. (The only HP-supplied binary program which can be stored is PHYREC.)

```
RE-STORE Name$&Msus$
RE-STORE "TEMP:INTERNAL,4,1"
RE-STORE BIN "PHYREC<Pc>"
```

Minimum requirement AP2.0:

```
RE-STORE KEY "AIDS"
```

RESUME INTERACTIVE

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

```
RESUME INTERACTIVE
```

RETURN

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).

```
RETURN
```

REV$

Minimum requirement AP2.0. This function returns a string formed by reversing the sequence of characters in the argument.

```
Reverse$=REV$(Forward$)
Last_blank=LEN(A$)-POS(REV$(A$)," ")
```

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. A negative position parameter causes a left shift. 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
```

Minimum requirement GRAPH2.1:

```
RPLOT Array(*)
RPLOT Shape(*),FILL,EDGE
```

RPT$

Minimum requirement AP2.0. This function returns a string formed by repeating the argument a specified number of times.

```
PRINT RPT$("*",80)
Center$=RPT$(" ",(Right-Left-Length)/2)
```

RUN

This command starts program execution at the specified line. If no parameter is specified, the program starts at the beginning.

```
RUN
RUN 10
RUN Part2
```

S

SAVE

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

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

SC

Minimum requirement AP2.0. This function returns the interface select code associated with an I/O path name.

```
Interface=SC(@Device)
Drive_isc=SC(@File)
```

SCRATCH

This command erases all or selected portions of memory.

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 typing-aid key definitions, and all variables, including those in COM. Most internal parameters in the computer are reset to power-on values by this command.

Minimum requirement AP2.0:

```
SCRATCH KEY
SCRATCH KEY 2
```

This clears the typing-aid definition of all softkeys or the selected softkey.

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 Prime 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 ECHO

Minimum requirement GRAPH2.0. This statement sets an echo to the specified location on the current PLOTTER IS device.

```
SET ECHO X_location,Y_location
SET ECHO 1000,2200
```

SET PEN

Minimum requirement GRAPH2.1. This statement defines the color of one or more entries in the color map. Any pixels already drawn with the specified pen are changed to the new color. The following table shows HSL values for the default color map entries.

```
SET PEN P_num COLOR Hue,Saturate,Luminous
SET PEN Selector INTENSITY Red,Blue,Green
SET PEN Start_pen COLOR Hsl_array(*)
SET PEN 2 INTENSITY 4/15,1/15,9/15
```

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
14	Orange	.08	1	1
15	Brown	.08	.70	.85

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. A negative position parameter causes a left shift. 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
```

SIGNAL

Minimum requirement AP2.0. This statement generates a software interrupt. See ON SIGNAL.

```
SIGNAL Selector
SIGNAL 3
```

SIN

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

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

SIZE

Minimum requirement AP2.0. This function returns the number of elements in a dimension of an array.

```
Total_words=SIZE(Words$,1)
Upperbound(2)=BASE(A1,2)+SIZE(A1,2)-1
```

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.

```
STOP
```

STORE

This statement creates a file and stores either an internal form of the BASIC program, normal binary programs, or typing-aid definitions into the file. (The only HP-suppied binary program which can be stored is PHYREC.)

```
STORE Name$&Msus$
STORE "TEMP:INTERNAL,4,1"
STORE BIN "PHYREC<Pc>"
```

Minimum requirement AP2.0:

```
STORE KEY "AIDS"
```

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

Minimum requirement AP2.0:

```
SUB Extract(Buff$ BUFFER,Pointer)
```

SUBEXIT

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

```
SUBEXIT
IF Done THEN SUBEXIT
```

SUM

Minimum requirement AP2.0. This function returns the sum of all the elements in a numeric array.

```
Total=SUM(Array)
PRINT SUM(Squares)
```

SUSPEND INTERACTIVE

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

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

SYMBOL

Minimum requirement GRAPH2.1. This statement allows labeling with user-defined symbols.

```
SYMBOL My_char(*)
SYMBOL Logo(*),FILL,EDGE
```

SYSTEM PRIORITY

Minimum requirement AP2.0. This statement sets the system priority to a specified level.

```
SYSTEM PRIORITY Level
SYSTEM PRIORITY 15
```

SYSTEM$

Minimum requirement AP2.0. This function returns a string containing system status and configuration information.

```
Memory=VAL(SYSTEM$("AVAILABLE MEMORY"))
IF SYSTEM$("TRIG MODE")="RAD" THEN Radian
```

The following requests are allowed.

AVAILABLE MEMORY
CRT ID
DUMP DEVICE IS
GRAPHICS INPUT IS
KEYBOARD LANGUAGE
LEXICAL ORDER IS
MASS MEMORY
MASS STORAGE IS
MSI

PLOTTER IS
PRINTALL IS
PRINTER IS
SERIAL NUMBER (need AP2.1)
SYSTEM ID
SYSTEM PRIORITY
TRIG MODE
VERSION: BASIC

T

TAN

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

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

TIME

Minimum requirement AP2.0. This function converts a formatted time-of-day string into a numeric value of seconds past midnight.

```
Seconds=TIME(T$)
SET TIME TIME("8:37:30")
```

TIME$

Minimum requirement AP2.0. This function converts the number of seconds past midnight into a string representing the formatted time of day (HH:MM:SS).

```
PRINT "It is ";TIME$(TIMEDATE)
IF VAL(TIME$(T1))>17 THEN Overtime
```

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 OFF
```

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
```

TRACK

Minimum requirement GRAPH2.0. This statement enables or disables tracking of the current locator position on the current display device. See GRAPHICS INPUT IS and PLOTTER IS.

```
TRACK Display IS ON
TRACK 709 IS OFF
```

TRANSFER

Minimum requirement AP2.0. This statement initiates unformatted I/O transfers, which can take place concurrently with continued program execution. Every TRANSFER needs a buffer as either its source or its destination (but not both).

```
TRANSFER @Device TO @Buff
TRANSFER @Buff TO @File;CONT
TRANSFER @Source TO @Buffy;DELIM "/",END
TRANSFER @F TO @B;RECORDS 12,EOR(COUNT 8)
```

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
```

TRIM$

Minimum requirement AP2.0. This function returns a string formed by stripping all leading and trailing blanks from the argument.

```
Left$=TRIM$("     center     ")
Clean$=TRIM$(User_input$)
```

U

UPC$

Minimum requirement AP2.0. This function returns a string formed by replacing any lowercase characters with the corresponding uppercase characters.

```
Capital$=UPC$(Mixed$)
IF UPC$(Yes$)="Y" THEN True_test
```

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

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
```

WAIT FOR EOR

Minimum requirement AP2.0. This statement waits until an end-of-record event occurs during a TRANSFER on the specified I/O path.

```
WAIT FOR EOR @File
WAIT FOR EOR @Device
```

WAIT FOR EOT

Minimum requirement AP2.0. This statement waits until the TRANSFER completes on the specified I/O path.

```
WAIT FOR EOT @File
WAIT FOR EOT @Device
```

WHERE

Minimum requirement GRAPH2.1. This statement returns the current logical position of the pen.

```
WHERE X_pos,Y_pos
WHERE X,Y,Status$
```

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
```

WINDOW

This statement is used to define an anisotropic 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
```

X

XREF

Requires AP2.0 or XREF2.1. This command produces a cross-referenced listing of the identifiers in a program or sub-program.

```
XREF
XREF #PRT
XREF #701;FNUser$
```

I/O Path Status and Control Registers

Example Statements

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

For All I/O Path Names:

Status Register 0

Returned Value	Meaning
0	Invalid I/O path name
1	I/O path name assigned to a device
2	I/O path name assigned to a data file
3	I/O path name assigned to a buffer

I/O Path Names Assigned to a Device:

Status Register 1	Interface select code
Status Register 2	Number of devices
Status Register 3	1st device selector

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

I/O Path Names Assigned to an ASCII File:

Status Register 1	File type = 3
Status Register 2	Device selector of mass storage device
Status Register 3	Number of records
Status Register 4	Bytes per record = 256
Status Register 5	Current record
Status Register 6	Current byte within record

I/O Path Names Assigned to a BDAT File:

Status Register 1	File type = 2
Status Register 2	Device selector of mass storage device
Status Register 3	Number of defined records
Status Register 4	Defined record length

Status Register 5	Current record
Control Register 5	Set current record
Status Register 6	Current byte within record
Control Register 6	Set current byte within record
Status Register 7	EOF record
Control Register 7	Set EOF record
Status Register 8	Byte within EOF record
Control Register 8	Set byte within EOF record

I/O Path Names Assigned to a Buffer:

Status Register 1	Buffer type (1 = named, 2 = unnamed)
Status Register 2	Buffer size in bytes
Status Register 3	Current fill pointer
Control Register 3	Set fill pointer
Status Register 4	Current number of bytes in buffer
Control Register 4	Set number of bytes
Status Register 5	Current empty pointer
Control Register 5	Set empty pointer
Status Register 6	Interface select code of inbound TRANSFER
Status Register 7	Interface select code of outbound TRANSFER
Status Register 8	If non-zero, inbound TRANSFER is continuous
Control Register 8	Cancel continuous mode inbound TRANSFER if zero
Status Register 9	If non-zero, outbound TRANSFER is continuous
Control Register 9	Cancel continuous mode outbound TRANSFER if zero
Status Register 12	Total number of bytes transferred by last inbound TRANSFER
Status Register 13	Total number of bytes transferred by last outbound TRANSFER

Status Register 10 — Inbound Termination

Most Significant Bit							Least Significant Bit
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANSFER Active	TRANSFER Aborted	TRANSFER Error	Device Termination	Byte Count	Record Count	Match Character
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 11 — Outbound Termination

Most Significant Bit							Least Significant Bit
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	TRANSFER Active	TRANSFER Aborted	TRANSFER Error	Device Termination	Byte Count	Record Count	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

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

Control Register 8 — Undefined

Interrupt Status
Least Significant Bit

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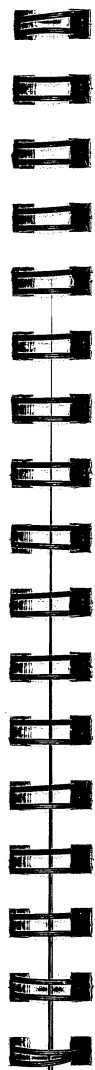
Status Register 7
Most Significant Bit

Interrupt Disable Mask
Least Significant Bit

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Keyboard and Knob	RESET Key	Reserved For Future Use	Reserved For Future Use	INITIALIZE Timeout	Not Used	Not Used	Not Used
Value = 1	Value = 2	Value = 4	Value = 8	Value = 16	Value = 32	Value = 64	Value = 128

Control Register 7 (Set bit to disable)
Most Significant Bit

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
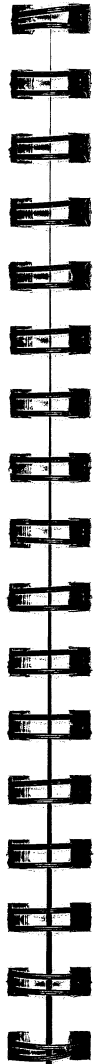
Status Register 10
Most Significant Bit

State at Last Knob Interrupt
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

CRT Status and Control Registers

Example Statements

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

Status Register 0	Current print position (column)
Control Register 0	Set print position (column)

Status Register 1	Current print position (line)
Control Register 1	Set print position (line)

Status Register 2	Insert-character mode
Control Register 2	Set insert character mode if non-0

Status Register 3	Number of lines "above screen"
Control Register 3	Undefined

Status Register 4	Display functions mode
Control Register 4	Set display functions mode if non-0

Status Register 5	Undefined
Control Register 5	Set default alpha color:

Value	Result	Value	Result
< 128	Error	140	Cyan
128-135	Ignored	141	Blue
136	White	142	Magenta
137	Red	143	Black
138	Yellow	144-159	Ignored
139	Green	> 159	Error

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 (column)
Control Register 8	Set display line position (column)

Status Register 9	Screenwidth (number of characters)
Control Register 9	Undefined

Status Register 10	Cursor-enable flag
Control Register 10	Cursor-enable: 0 = cursor visible 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 mode
Control Register 12	Set key labels display mode: 0 = typing-aid key labels displayed unless program is in the RUN state 1 = key labels always off 2 = key labels displayed at all times

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

Status Register 2

Busy Bits

Bit 7 (Most Significant Bit)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (Least Significant Bit)
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 (Most Significant Bit)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (Least Significant Bit)
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

Control Register 3 — Set My Address

Most Significant Bit — Least Significant Bit

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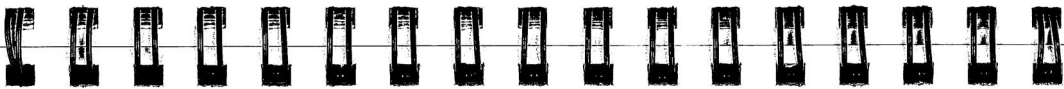
Status Register 4 — Interrupt Status

Most Significant Bit

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

Least Significant Bit

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

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

Status Register 5

Interrupt Enable Mask

Most Significant Bit

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

Least Significant Bit

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

Control Register 5

Parallel Poll Response Mask

Most Significant Bit

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Uncon-figure	Logic Sense	Data Bit Used For Response		
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

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

Non-ASCII keypresses can be simulated by outputting a 2-byte sequence to the keyboard. The decimal value of the first byte is 255. The interpretation of the second byte is shown in this table.

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	k_0
1	49	k_1
2	50	k_2
3	51	k_3
4	52	k_4
5	53	k_5
6	54	k_6
7	55	k_7
8	56	k_8
9	57	k_9
<	60	←
=	61	RESULT
>	62	→
?	63	RECALL
@	64	SHIFT - RECALL
A	65	PRT ALL

Character	Value	Key
B	66	BACK SPACE
C	67	CONTINUE
D	68	EDIT
E	69	ENTER
F	70	DISPLAY FCTNS
G	71	SHIFT - →
H	72	SHIFT - ←
I	73	CLR I/O
J	74	Katakana Mode
K	75	CLR SCR
L	76	GRAPHICS
M	77	ALPHA
N	78	DUMP GRAPHICS
O	79	DUMP ALPHA
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	k_{10}
b	98	k_{11}
c	99	k_{12}
d	100	k_{13}
e	101	k_{14}
f	102	k_{15}
g	103	k_{16}
h	104	k_{17}
i	105	k_{18}
j	106	k_{19}

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.
43 Matrix must be square.
44 Result matrix cannot be an operand.
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.
65 Incorrect data type for graphics operation.
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.
89	Checkread 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.
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.
135	READIO or WRITEIO to non-existent memory location.
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.
164	Illegal ASSIGN of WORD attribute.
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.
301	Cannot do while connected.
303	Cannot do while trace active.
304	Too many characters without a terminator.
306	Interface card failure.
308	Illegal character in data.
310	Not connected.
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.
324	Card trace buffer overflow.

325	Illegal databits/parity combination.
326	Register address out of range.
327	Register value out of range.
328	USART transmit underrun.
330	User-defined LEXICAL ORDER table size exceeds array size.
331	MAT REORDER vector has repeated subscripts.
332	Non-existent dimension given for MAT REORDER.
333	Improper subscript in MAT REORDER vector.
334	Vector size not equal to number of elements for MAT REORDER.
335	Pointer for MAT REORDER is not a vector.
337	Substring key out of range.
338	Key subscript out of range.
340	LEXICAL ORDER mode table is too long.
341	Improper LEXICAL ORDER mode indicator.
342	LEXICAL ORDER mode table is not an INTEGER vector.
343	LEXICAL ORDER mode pointer out of range.
344	LEXICAL ORDER 1-for-2 list is wrong size.
345	CASE expression type mismatch.
346	INDENT parameter out of range.
347	Structures improperly matched.
349	CSUB has been modified.
353	Data link failure.
370 thru 399	Run-time Pascal error during a CSUB. Subtract 400 from BASIC error number and refer to the *Pascal 2.0 User's Manual*.
401	Bad system-function argument.
403	COPYLINES or MOVELINES failed; program modification incomplete.
427	Priority may not be lowered.
450	Volume not found (SRM).
451	Volume labels do not match (SRM).
453	File in use (SRM).
454	Directory formats do not match (SRM).
455	Possibly corrupt file (SRM).

456	Unsupported directory operation (SRM).
457	Passwords not supported (SRM).
458	Unsupported directory format (SRM).
459	Specified file is not a directory (SRM).
460	Directory is not empty (SRM).
462	Invalid password (SRM).
465	Invalid rename across volumes (SRM).
471	TRANSFER not supported by the interface.
481	File locked or opened exclusively (SRM).
482	Cannot move a directory with a RENAME operation (SRM).
483	System down (SRM).
484	Password not found (SRM).
485	Invalid volume copy (SRM).
488	DMA hardware required.
511	Result array for MAT INV must be REAL.
600	BYTE/WORD attribute cannot be modified.
601	Improper CONVERT lifetime.
602	Improper BUFFER lifetime.
603	Variable was not declared as a BUFFER.
604	Improper source or destination for TRANSFER.
605	BDAT file type required.
606	Improper TRANSFER parameters.
607	Inconsistent attributes.
609	IVAL or DVAL result too large.
612	BUFFER pointers in use.
700	Improper plotter specifier.
702	CRT graphics hardware missing.
704	Upper bound not greater than lower bound. (P2≤P1 or VIEWPORT/CLIP conflict)
705	VIEWPORT or CLIP beyond hard clip limits.
708	Device not initialized.
713	Request not supported on specified graphics device.
733	GESCAPE operation not recognized.
900	Undefined typing-aid key.
901	Typing-aid memory overflow.

902	Must delete entire context.
903	No room to renumber.
904	Attempt to find a null string.
905	CHANGE would produce too long a line.
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.	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.	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.	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.	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