HELP

Each $HELP page consists of up to six sections:

Sections are abbreviated or omitted where not appropriate.

1) Title and Reference
2) Purpose
3) Syntax
4) Comments (Usage rules)
5) Examples
6) Other references

TITLE and REFERENCE	The Title is the primary key used to access the page. The Reference is a manual page number or description right justified within parentheses on the top line which indicates where further information on the keyword can be obtained. References take the following forms:
(number)	Page number, 1979 Wang BASIC-2 Language Reference Manual
(Disk number)	Page number, 1981 Wang Basic-2 Disk Reference Manual
(Wang SB 2.x)	Wang BASIC-2 Software Bulletin Release 2.x
(SDS SB 2.7)	SDS-Extended BASIC-2 Software Bulletin Release 2.7
(SDS SB 2.8)	SDS-Extended BASIC-2 Software Bulletin Release 2.8
PURPOSE	This is a brief description of the main purpose(s) of the BASIC-2 keyword.
SYNTAX	General syntax is specified using a meta-language to differentiate between exact and programmer-supplied entries, and to show alternate and optional parameters and data.
COMMENTS	This section includes an explanation of all non-standard abbreviations used in the syntax, plus variable dimensioning requirements, parameter tables, restrictions, and information on the statement.
EXAMPLES	Most pages include one or more BASIC-2 examples.
OTHER REFERENCES	Where appropriate, mention is made of other $HELP titles or manuals that contain more information, or which use the keyword in a different manner.

PURPOSE: A parameter.

REFER TO: $BREAK
SAVE DA
SAVE DC

PURPOSE: A device-table-slot designator, LIST parameter, and keyword character.

SYNTAX ... #nn [...]

COMMENTS: nn = A numeric-expression such that 0 <= nn <= 15.

Where allowed in BASIC-2 syntax, #nn designates that the I/O defined by the accompanying BASIC-2 verb will be directed toward or received from the peripheral whose address has been entered into the specified device-table-slot by SELECT.

EXAMPLES:
```
10 SELECT #1 D10
20 DATASAVE DC #15, X$()
30 $GIO #X (A000) A$()
```

REFER TO: SELECT

LIST #

#ID
#ID'
#PART
#PI
#TERM

#ID

PURPOSE: A built-in numeric-function that returns the CPU identification.

SYNTAX ... #ID [...]

COMMENTS: #ID returns a value imbedded in the CPU bootstrap PROMs such that 0<= #ID <= 65535.

The #ID value is assigned randomly during CPU manufacturing, and it is unlikely that two CPUs will have the same #ID.

#ID is a relatively recent implementation. Older CPUs using older PROMs return a value of zero, signifying that #ID is not implemented.

EXAMPLES:
```
10 PRINT #ID
20 X = #ID
```

#ID'

PURPOSE: A built-in numeric-function that returns the SDS-Extended BASIC-2 operating system registration number.

SYNTAX ... #ID' [...]

COMMENTS: #ID' returns a value imbedded in the OS machine language code such that 0<= #ID' <= 999999.

The #ID' value is assigned during OS generation and it is unique to each registered licensee of SDS-Extended BASIC-2.

EXAMPLES:
```
10 PRINT #ID'
20 X = #ID'
30 IF #ID' <> 121617 THEN STOP

40 IF #ID' + #ID 568821 THEN 50
 : PRINT "SYSTEM NOT LICENSED FOR USE IN THIS CONFIGURATION"
 : PRINT "PLEASE CALL SOUTHERN DATA SYSTEMS, INC. FOR ASSISTANCE"
```

```
50 PRINT "Your CPU number is ";#ID
 : PRINT "Your OS Registration number is ";#ID'
```

#PART

PURPOSE: A built-in numeric-function that returns the originating partition number.

SYNTAX ... #PART [...]

COMMENTS: #PART returns a value from 1 to 16 corresponding to the number of the originating partition which is executing the program.

EXAMPLES:
```
10 PRINT #PART
20 X = #PART
```

#PI

PURPOSE: A built-in numeric-function that returns the value of pi.

SYNTAX ... #PI [...]

COMMENTS: The value of #PI = 3.14159265359.

EXAMPLE:
```
10 X = 2*#PI
```

#TERM

PURPOSE: A built-in numeric-function that returns the terminal number assigned to the orginating partition.

SYNTAX ... #TERM [...]

COMMENTS: #TERM returns a value from 0 to 16 corresponding to the number of the terminal (port) assigned to the originating partition. If the value is zero, no terminal is assigned.

EXAMPLES:
```
10 PRINT #TERM
20 X = #TERM
```

$

PURPOSE: An alpha-variable name designator, and a keyword character specifying a special class of BASIC-2 keywords.

REFER TO: Chapter 5 of the BASIC-2 Language Reference Manual

$ALERT
$BREAK
$CLOSE
$DISCONNECT
$FORMAT

$FORMAT DISK
$GIO
$HELP
$ID
$IF OFF
$IF ON
$INIT
$MSG
$OPEN
$PACK
$PSTAT
$RELEASE PART
$RELEASE TERMINAL
$TRAN
$UNPACK

$ALERT

PURPOSE: Generate an interrupt to a specified partition.

SYNTAX $ALERT partition

COMMENTS: partition = A numeric-expression such that
1 <= partition <= maximum partitions in system.

$ALERT signals that the specified partition is to execute an interrupt subroutine at the completion of its currently executing statement.

$ALERT interrupt capability must have been enabled in the partition with a SELECT ON ALERT GOSUB statement. Otherwise the interrupt is ignored.

If the partition is currently executing an interrupt subroutine, the interrupt is not fielded until the subroutine completes.

$ALERT can be used to "wake up" a partition disabled by $BREAK!.

Execution of a LOAD, RUN, or CLEAR in the partition clears pending interrupts.

EXAMPLES: 10 $ALERT 4
20 $ALERT X(Y)

REFER TO: SELECT

$BREAK

PURPOSE: Give up one or more partition time-slices.

SYNTAX $BREAK [n]
[!]

COMMENTS: n = A numeric-expression specifying the number of time-slices to give up such that 0 <= n <= 255.

! = A parameter specifying that the partition is to be

permanently bypassed by the operating system until a RESET command is received or a $ALERT is executed from another partition.

If neither n nor ! is specified, a $BREAK 1 is assumed.

EXAMPLES:
```
10 $BREAK
20 $BREAK MIN(255,5*X)
30 $BREAK !
```

$CLOSE

PURPOSE: Release control of one or more peripheral devices.

SYNTAX
```
$CLOSE [file#  ] [, file#  ] ...
       [address] [, address]
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

address = An explicit address of the form /taa where t is the device-type and aa is the unit-address.

If neither file# nor address is specified, all devices are $CLOSEd.

EXAMPLES:
```
10 $CLOSE
20 $CLOSE /01C,#4
```

REFER TO: $OPEN

$DISCONNECT

PURPOSE: Provide a means of detecting or forcing terminal disconnection.

SYNTAX
```
$DISCONNECT ON  [time]
$DISCONNECT OFF
```

COMMENTS:

ON = A parameter which enables $DISCONNECT.

OFF = A parameter which disables $DISCONNECT.

time = A numeric-expression such that 0 <= time <= 65534 which specifies the number of seconds before the operating system forces terminal disconnection. If omitted, a time of zero is assumed. If zero, forced disconnection is disabled.

If $DISCONNECT is enabled, and a forced or other terminal disconnection occurs, the operating system automatically LOAD RUNs a user-written program named "@DISCNCT" in all partitions controlled by the terminal.

An MXE terminal controller is required for use of this statement.

$DISCONNECT is a command to the terminal port and operates independently of the partition(s) executing the command. The last $DISCONNECT issued to a particular port by any partition attached to that port determines the current $DISCONNECT mode.

```
EXAMPLES: 10 $DISCONNECT ON
          20 $DISCONNECT OFF
          30 $DISCONNECT ON 60: REM Kill terminal in one minute
```

$FORMAT

PURPOSE: Create a format specification for the field form of $PACK/$UNPACK.

SYNTAX $FORMAT alpha-variable = field-spec [,field-spec] ...

COMMENTS:

field-spec = A mnemonic specifying a hexadecimal sequence to be stored in the alpha-variable for later use by the field form of $PACK or $UNPACK.

Field-Spec	Definition
SKIPxxx	Skip over a field
Fxxx	ASCII free format
Ixxx[.dd]	ASCII integer format
Dxxx[.dd]	IBM display format
Uxxx[.dd]	IBM USASCII-8 format
P+xxx[.dd]	IBM packed decimal format
Pxxx[.dd]	Unsigned packed decimal format
Axxx	Alphanumeric format

xxx = Field width such that 1 <= xxx <= 255

dd = Implied decimal position such that 0 <= dd <= 15

```
EXAMPLES: 10 $FORMAT F$ = A16,SKIP96,P+5.2
          20 $UNPACK(F=F$)Q$() TO A$,B()
```

REFER TO: $PACK
$UNPACK

$FORMAT DISK

PURPOSE: Format a disk platter.

SYNTAX $FORMAT DISK pd [file# [,]]
[address[,]]

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

$FORMAT DISK provides a means of formatting disk platters under program control. The formatting procedures and the time required to complete the operation is device dependent.

The referenced disk must be software formattable.

All information on the platter is erased.

```
EXAMPLES: 10 $FORMAT DISK F
          20 $FORMAT DISK T#1
          30 $FORMAT DISK T/D15,
```

$GIO

PURPOSE: Provides a generalized means of controlling I/O devices not directly supported by BASIC-2 I/O statements.

SYNTAX `$GIO [comment] [file#  [,]] (arg-1 [,arg-2]) [arg-3] [; arg-3] ... [address[,]]`

COMMENTS:

- comment = An optional character string used to describe the operation of the $GIO sequence. Only uppercase letters, digits, and spaces are allowed.
- file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.
- address = An explicit address of the form /taa where t is the device-type and aa is the unit-address.
- arg-1 = A microcommand sequence consisting of a literal-string, alpha-variable, or string of hex digits.
- arg-2 = An alpha-variable at least 10 bytes in length used as storage, error, and status registers by $GIO.
- arg-3 = An alpha-variable which serves as a data buffer for multi-character I/O operations. It is of the form:
  alpha-variable [<[s][,n]>]
  where s is a numeric-expression specifying the start-byte, and n is a numeric-expression specifying the number of bytes involved in the operation.

If neither file# nor address is specified, the TAPE device is assumed.

```
EXAMPLES: 10 $GIO (A000) X$()
          20 $GIO PRINT TOP OF FORM /215 (400C)
          30 $GIO SEND DATA #1 (Q$(),R$) A$();B$<S,X+Y>
```

$HELP

PURPOSE: Retrieve and display (or print) text from a $HELP disk file.

SYNTAX `$HELP [S] [pd [file#  ]] [<filename>] [keyword] [;] [TO alpha-var] [address]`

COMMENTS:

- S = A parameter indicating that the text output is to be scrolled one screen at a time with the number of lines displayed per screen determined by the current value of SELECT LINE. This parameter is executed only in immediate mode.
- pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

keyword = An alpha-variable or literal-string specifying the key to the page to be retrieved. If omitted, all page keys are returned. If the specified key does not exist, an error X78 is signaled.

filename = An alpha-variable or literal-string specifying the name of a disk file containing text in $HELP format. If omitted, the name "HELP" is assumed.

; = A parameter indicating that text is to be retrieved continuously starting with the page specified by the key.

alpha-var = An alpha-variable, which, if included, receives the $HELP text. If ommitted, the output is sent to the LIST device.

If the platter-designator is omitted, "T" is assumed.

If file# and address are omitted, the disk unit specifified by SELECT DISK is assumed.

Lines of text stored in an alpha-variable are separated by carriage returns (HEX(0D)). The first line on a page is preceeded by a HEX(0E) to highlight the title. Output ceases when either all requested text has been retrieved, or the alpha-variable is full.

If the page keys are to be transferred to a variable, the keyword is omitted, and the semi-colon is required.

EXAMPLES:
```
$HELP
$HELP S "PRINT"
10 X$() = $HELP T#3, <"SCREENS"> K$(3)
20 SELECT LIST 215: $HELP T/D12, "MAT";
30 $HELP <F$> K$ TO T$()
40 $HELP ; TO K$()
```

$ID

PURPOSE: Retrieve the name of the registered licensee of the SDS-Extended BASIC-2 Operating System.

SYNTAX alpha-varable = $ID

COMMENTS: The name of the registered licensee is imbedded in the Operating System code during system generation. The string retrieved by the $ID function is the same as that displayed on the system Pre-loader screen displayed during system initialization.

EXAMPLES:

Assume the registered licensee is SDS Distributing Company, Inc., of Raleigh, NC. Then the name line listed on the system Pre-loader screen displayed during system initialization would read:

"SDS Distributing Company, Inc., -- Raleigh, NC"

The following statements would give the results indicated.

```
DIM A$80
A$=$ID
PRINT A$
SDS Distributing Company, Inc., -- Raleigh, NC

10 DIM A$80
20 A$=$ID
30 PRINT "This software is licensed to ";A$
 : PRINT "under operating system registration number ";#ID'
 : PRINT "for operation on CPU number ";#ID

70 REM Get Client Name
 : A$=$ID
 : N$=STR(A$,1,POS(A$="-")-1)
```

$IF OFF

PURPOSE: Test if a peripheral device is not ready.

SYNTAX
```
$IF OFF [file#,  ] line-number
        [address,]
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

address = An explicit address of the form /taa where t is the device-type and aa is the unit-address.

$IF OFF tests the specified device for a ready condition. If the device is not ready, a branch is made to the specified line-number.

If a device-address or file-number is not specified, $IF OFF tests the TAPE device.

Consult the BASIC-2 or peripheral manual for information on specific devices.

EXAMPLES:
```
10 $IF OFF 100
20 $IF OFF /001, 250
30 $IF OFF #1, 300
```

$IF ON

PURPOSE: Test if a peripheral device is ready.

SYNTAX
```
$IF ON [file#,  ] line-number
       [address,]
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

address = An explicit address of the form /taa where t is the device-type and aa is the unit-address.

$IF ON tests the specified device for a ready condition. If the device is ready, a branch is made to the specified line-number.

If a device-address or file-number is not specified, $IF ON tests the TAPE device.

Consult the BASIC-2 or peripheral manual for information on specific devices.

EXAMPLES:
```
10 $IF ON 100
20 $IF ON /001, 250
30 $IF ON #1, 300
```

$INIT

PURPOSE: Configure the system, or pass system control to the bootstrap.

SYNTAX
Command:
```
$INIT "password"
```

Statement:
```
$INIT (arg-1, arg-2, arg-3, arg-4, arg-5 [, arg-6])
```

COMMENTS:

password = A literal-string with length of one to eight characters.

arg-1 = An alpha-variable specifying partition sizes.

arg-2 = An alpha-variable specifying the terminal for each partition.

arg-3 = An alpha-variable specifying programability of each partition.

arg-4 = An alpha-variable specifying the automatically booted program name for each partition.

arg-5 = An alpha-variable specifying allowable devices.

arg-6 = An alpha-variable specifying the reconfiguration password required by the $INIT command. If omitted, "SYSTEM" is assumed.

The immediate mode command is used by terminal #1 to transfer CPU control to the bootstrap prom, at which point the system can be reconfigured. The proper password is required.

The statement form of $INIT requires extensive system knowledge for proper usage. Consult the BASIC-2 manual for further information.

EXAMPLE:
```
$INIT "SYSTEM"
10 $INIT (X1$(),X2$(),X3$(),X4$(),X5$(),X6$())
```

$MSG

PURPOSE: Define or retrieve the system message.

SYNTAX
Assignment:
```
$MSG = alpha-expression
```

$MSG = alpha-expression

Retrieval:
alpha-variable = $MSG

COMMENTS: The contents of $MSG are automatically displayed on line 0 of the CRT following a CLEAR, RESET, or system initialization.

The system message may be retrieved by a program using a LET.

Only terminal #1 may modify the message, which can be up to 80 bytes long.

EXAMPLE:
```
10 $MSG = "Good Morning!"
20 M$ = $MSG
```

$OPEN

PURPOSE: Obtain exclusive access to a peripheral device for a partition.

SYNTAX
```
$OPEN [line-number,] [file#  ] [, file#  ] ...
                     [address] [, address]
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

address = An explicit address of the form /taa where t is the device-type and aa is the unit-address.

line-number = The program line-number to branch to if any of the specified devices is not available.

If multiple devices are specified, exclusive access is not granted to any device unless all specified devices are available.

EXAMPLES:
```
10 $OPEN #12, /07B
20 $OPEN 1000, #4
```

REFER TO: $CLOSE

$PACK

PURPOSE: Pack an alpha-variable from a list of variables.

SYNTAX
```
$PACK [(type = spec)] var [<[s][,n]>] FROM list-item [,list-item]...
```

COMMENTS:

type = An F to denote the field form of $PACK, or a D to denote the delimiter form of $PACK. If not specified, Wang disk record format is assumed.

spec = An alpha-variable or literal-string containing the $PACK specifications to be used. A type F specification can be constructed using $FORMAT.

var = Any alpha-variable large enough to contain the packed data.

s = A numeric-expression specifying the first byte of

var to receive packed data. If omitted, byte 1 is assumed.

n = A numeric-expression specifying the number of bytes in var to receive packed data. If omitted, the remainder of the variable is assumed.

list-item = A literal-string, a numeric-expression, a variable, or an array-designator.

List values are extracted sequentially from left to right (or element by element if the list-item is an array) and packed into alpha-var.

In the following field specifications table, xx is the field width in binary where xx > 0, and d is the implied decimal position in binary.

Valid Field Specifications

Spec	Definition
00xx	Skip over a field
10xx	ASCII free format
2dxx	ASCII integer format
3dxx	IBM display format
4dxx	IBM USASCII-8 format
5dxx	IBM packed decimal format
6dxx	Unsigned packed decimal format
A0xx	Alphanumeric format

The first byte of a delimiter specification must fall in the range HEX(00) to HEX(03) and is ignored by $PACK. The second byte specifies the delimiter.

EXAMPLES:
```
$PACK S$() FROM S/2, T$(), "LAST ITEM"
$PACK (D=HEX(002D)) D$() FROM A$(),B$,STR(C$,40)
$PACK (F=F$) Q$ FROM 1,2,B$,Z()
```

REFER TO: $FORMAT

$PSTAT

PURPOSE: Retrieve or modify partition status.

SYNTAX
Assignment:
```
$PSTAT = alpha-expression
```

Retrieval:
```
alpha-variable = $PSTAT(n)
```

COMMENTS:

n = A numeric expression the integer value of which specifies a partition number such that 1 <= n <= number of partitions.

alpha-variable = A receiver containing 29 bytes of information output by $PSTAT such that:

Bytes	Description
1-8	User specified bytes;
9	Operating system type (M or V);
10	Packed (##) operating system release number;
11	Packed (##) memory bank number;
12-13	Packed (##.##) partition size;
14	Programability (P or space);
15	Packed (##) terminal number;

16 Terminal status (M, D, or W);
17-24 Partition global name;
25 Packed (##) ERR function value;
26 Packed (##) text partition number;
27 Packed (##) global partition number;
28 Packed (##) data partition number; and
29 Binary address of pending I/O device.

Only the user-specified portion of $PSTAT is modified in an assignment.

EXAMPLES:
```
10 P$ = $PSTAT (#PART)
20 $PSTAT = "Menu"&M$
```

$RELEASE PART

PURPOSE: Reassign a partition to terminal 0.

SYNTAX $RELEASE PART

COMMENTS: If executed in foreground, $RELEASE PART detaches the partition from the terminal. The terminal is automatically reattached to any background partition assigned to it. If no such background partition exists, the terminal is effectively removed from the system until execution of a RESET.

EXAMPLE :
```
10 $RELEASE PART
```

$RELEASE TERMINAL

PURPOSE: Reassign a terminal to another partition.

SYNTAX
```
$RELEASE TERMINAL [TO partition-name [,STOP]]
                        expression
```

COMMENTS: partition-name = A literal-string or alpha-variable containing a global partition name of 1-8 bytes.

expression = A numeric-expression, the integer value of which specifies the partition number for reattachment such that 1 <= expression <= number of partitions.

$RELEASE TERMINAL is ignored if the terminal is not attached to the partition when the statement is executed.

The TO parameter releases the terminal to a specified partition.

The STOP parameter causes a HALT to be executed upon attachment to the new partition. (This allows control to be regained in a partition that is continually looping through $RELEASE-type statements.)

EXAMPLE:
```
10 $RELEASE TERMINAL
20 $RELEASE TERMINAL TO P
30 $RELEASE TERMINAL TO "GLOBAL"
40 $RELEASE TERMINAL TO 1, STOP
```

$TRAN

PURPOSE: Translate characters in an alpha-variable.

SYNTAX $TRAN (arg-1 [<[s][,n]>], arg-2 [<[s][,n]>]) [mask] [R]

COMMENTS:

arg-1 = An alpha-variable. Each byte in arg-1 is replaced by a byte determined by arg-2.

arg-2 = An alpha-variable or literal-string which serves as a table to determine which bytes should replace those in arg-1.

mask = Two hexadecimal digits that are logically ANDed to each character of arg-1 before translation.

R = A parameter specifying replacement mode. If omitted, table offset mode is assumed.

s = A numeric-expression specifying the starting byte of the argument to use. If omitted, the first byte is assumed.

n = A numeric-expression specifying the number of bytes in the argument to use. If omitted, the remainder of the argument is assumed.

In replacement mode, arg-2 contains pairs of characters. As $TRAN scans and masks each arg-1 character, a match is sought as the second character of each pair. If found, the arg-1 character is replaced by the first character of the pair.

In table offset mode, the masked binary value of each arg-1 character is treated as an offset from the beginning of the arg-2 translation table. If available, the corresponding character at that position is substituted for the arg-1 character.

EXAMPLES:
```
10 $TRAN (A$(),B$) 7F R
20 $TRAN (C$(5,2),"0123456789ABCDEF")
```

$UNPACK

PURPOSE: Unpack an alpha-variable to a list of variables.

SYNTAX $UNPACK [(type = spec)] var [<[s][,n]>] TO list-item [,list-item]...

COMMENTS:

type = An F to denote the field form of $UNPACK, or a D to denote the delimiter form of $UNPACK. If not specified, Wang disk record format is assumed.

spec = An alpha-variable or literal-string containing the $UNPACK specifications to be used. A type F specification can be constructed using $FORMAT.

var = An alpha-variable containing the packed data.

s = A numeric-expression specifying the first byte of var to be unpacked. If omitted, byte 1 is assumed.

n = A numeric expression specifying the number of bytes in var to be unpacked. If omitted, the remainder of the variable is assumed.

list-item = A variable or array-designator.

List-items are filled sequentially from left to right (or element by element if the list-item is an array).

In the following field specifications table, xx is the field width in binary where xx > 0, and d is the implied decimal position in binary.

Valid Field Specifications

Spec	Definition
00xx	Skip over a field
10xx	ASCII free format
2dxx	ASCII integer format
3dxx	IBM display format
4dxx	IBM USASCII-8 format
5dxx	IBM packed decimal format
6dxx	Unsigned packed decimal format
A0xx	Alphanumeric format

In the following delimiter specifications table, xx denotes the delimiter character. The Error column indicates if a system error is signaled if there is insufficient data for all variables in the list. The Skip column indicates if list-items are skipped or ignored if successive delimeters are encountered.

Valid Delimiter Specifications

Spec	Error?	Skip?
00xx	Yes	Yes
01xx	No	Yes
02xx	Yes	No
03xx	No	No

EXAMPLES:
```
$UNPACK S$() TO X,Y$,STR(Z$,4)
$UNPACK (D=HEX(002D)) D$() TO A$(),B$,C()
$UNPACK (F=F$) Q$ TO G,G$(G)
```

REFER TO: $FORMAT

%

PURPOSE: Define an output image to be used with a PRINTUSING statement.

SYNTAX
```
% [characters ] ...
  [format-spec]
```

COMMENTS:
```
characters     = Any characters except the pound-sign (#) and the
                 carriage return (HEX(0D)).

format-spec    = [+] [$] [#[,]...] [.] [#...] [↑↑↑↑] [+]
                 [-]                                 [-]
                                                     [++]
                                                     [--]
```

The % identifies the statement as an image statement. It must be the only statement on the line. One # is required in the format-spec. A leading or trailing sign can be present, but not both.

EXAMPLE:
```
10 %-#.#######↑↑↑↑
20 %NET INCOME = $##,###,##0.
```

```
30 %              REPORT TITLE
```

REFER TO: PRINTUSING
PRINTUSING TO

&

PURPOSE: An operator specifying string concatenation.

SYNTAX alpha-variable = [...] alpha-spec & alpha-spec [...]

COMMENTS: alpha-spec = An alpha-variable, alpha-literal, alpha-function, or string function.

The & operator can only be used in assignment (LET) statements.

Trailing spaces are truncated in variables used in alpha-spec unless the string function is used.

```
EXAMPLES: 10 X$ = DATE & TIME
          20 Y$ = "ADDRESS: " & S$ & " " & C$ & ", " & CO$ & "  " & Z$
          30 Q$ = STR(A$) & B$: REM CONSTRUCT KEY TO FILE
```

'

PURPOSE: The apostrophe is a BASIC-2 text parameter.

REFER TO: DEFFN'
GOSUB'
SCRATCH DISK

*

PURPOSE: Math operator for multiplication.

SYNTAX ... arg-1 * arg-2 [...]

COMMENTS: arg-1,arg-2 = Numeric-expressions.

A * (asterisk or star) denotes that arg-1 is multiplied by arg-2.

Consult the BASIC-2 manual for proper use in numeric expressions and the hierarchy of evaluation.

EXAMPLE: 10 A = B * C

+

PURPOSE: A positive sign or a math operator for addition.

SYNTAX ... [arg-1] + arg-2 [...]

COMMENTS: arg-1,arg-2 = Numeric-expressions.

A + (plus-sign) when used as an operator denotes that items on either side are to be added together. If arg-1 is omitted, + denotes the sign of arg-2 as positive. BASIC-2 assumes arg-2 is positive if the sign is omitted.

Consult the BASIC-2 manual for proper use in numeric expressions and the hierarchy of evaluation.

EXAMPLES: 10 A = B + C
X = + 12

—

PURPOSE: Math operator for subtraction and negation.

SYNTAX ... [arg-1] - arg-2 [...]

COMMENTS: arg-1,arg-2 = Numeric-expressions.

A - (minus-sign) when used with two arguments denotes that arg-2 is subtracted from arg-1. If arg-1 is omitted, - denotes the sign of a numeric constant or negates (changes the sign) of arg-2. BASIC-2 assumes arg-2 is positive if the sign is omitted.

Consult the BASIC-2 manual for proper use in numeric expressions and the hierarchy of evaluation.

EXAMPLES: 10 A = B - C
20 X = - 12

/

PURPOSE: Math operator for division.

SYNTAX ... arg-1 / arg-2 [...]

COMMENTS: arg-1,arg-2 = Numeric-expressions.

A / (slash, stroke or virgule) denotes that arg-1 is divided by arg-2.

Consult the BASIC-2 manual for proper use in numeric expressions and the hierarchy of evaluation.

EXAMPLE: 10 A = B / C

↑

PURPOSE: Math operator for exponentiation.

SYNTAX ... arg-1 ↑ arg-2 [...]

COMMENTS: arg-1,arg-2 = Numeric-expressions.

A ↑ (up-arrow) denotes that arg-1 is raised to the power of arg-2.

If arg-2 is an integer, BASIC-2 performs the exponentiation as a series of multiplications to preserve accuracy and reduce execution time. Otherwise, BASIC-2 uses logarithmns to calculate the result.

If arg-1 is negative, arg-2 must be an integer.

If arg-1 is zero, arg-2 must be positive.

Consult the BASIC-2 manual for proper use in numeric expressions and the hierarchy of evaluation.

EXAMPLE: 10 A = B ↑ C

:

PURPOSE: The colon is used as a statement separator on multi-statement lines.

SYNTAX line-number statement [: statement] ...

EXAMPLE: 10 X = Y: IF X > 0 THEN 20: Y = 0

;

PURPOSE: The semi-colon is a BASIC-2 text parameter. The semi-colon is also used to indicate when more display information is available following a LIST S or $HELP S immediate mode command.

REFER TO: PRINT
PRINTUSING
PRINTUSING TO
SELECT
ON SELECT
$GIO
$HELP

<

PURPOSE: The left-angle-bracket is a logical operator denoting "less than", and a BASIC-2 text character.

REFER TO: IF
MAT SEARCH
POS

$GIO
$HELP
$PACK
$TRAN
$UNPACK
ON SELECT
PLOT
SAVE DA
SAVE DC
SELECT

<=

PURPOSE: A logical operator denoting "less than or equal".

REFER TO: IF
MAT SEARCH
POS

<>

PURPOSE: A logical operator denoting "not equal".

REFER TO: IF
MAT SEARCH
POS

=

PURPOSE: The equal-sign is an assignment operator and a logical operator denoting equality.

REFER TO: LET

IF
MAT SEARCH
POS

=SELECT
SCRATCH DISK

=SELECT

PURPOSE: Retrieve device table parameters and store them in a variable.

SYNTAX alpha-variable = SELECT file#
class
ALL

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

class = An I/O class such as PRINT, LIST, or DISK

ALL = A parameter which specifies that retrives the address and status of all peripherals declared in the master device table.

The following tables describe the format of the data returned by =SELECT. Each symbol in the format represents a hex digit.

Parameter	Length	Format
file#	8	f t aa ssss cccc eeee
CI	2	0 t aa
INPUT	2	0 t aa
PLOT	2	0 t aa

```
          TAPE            2      0 t aa
          CO              4      0 t aa ww 00
          PRINT           4      0 t aa ww 00
          LIST            4      0 t aa ww 00
          ALL            64      aa up [aa up] ...

          Format Symbol    Description

          t                Device type
          aa               Device address
          ww               Line width
          f                File status
                                 0 = Not open
                                 1 = Open on "F" drive
                                 2 = Open on "R" drive
                                 3 = Open on "T" drive
          ssss             Start sector address of file
          cccc             Current sector address of file
          eeee             Ending sector address of file
          p                A partition number where p = actual-partition - 1
          u                Device status
                                Bit   Meaning if Set (Equal to 1)
                                 1    Device is a disk drive
                                 2    Device is exclusively assigned to p
                                 3    Device is currently in use by p
                                 4    Device is currently hogged by p
          0                A hex digit used as a place holder

EXAMPLES: 10 X$ = SELECT #1
          20 Y$ = SELECT TAPE
          30 Z$ = SELECT ALL

REFER TO: SELECT
```

>

```
PURPOSE:  The right-angle-bracket is a logical operator denoting "greater
          than", and a BASIC-2 text character.

REFER TO: IF
          MAT SEARCH
          POS

          $GIO
          $HELP
          $PACK
          $TRAN
          $UNPACK
          ON SELECT
          PLOT
          SAVE DA
          SAVE DC
          SELECT
```

>=

```
PURPOSE:  A logical operator denoting "greater than or equal".

REFER TO: IF
          MAT SEARCH
          POS
```

?

PURPOSE: A LINPUT parameter.

REFER TO: LINPUT

@

PURPOSE: A variable-name character used to denote a global variable, or a character used in global partition reference keywords.

REFER TO: SELECT @PART
DEFFN @PART

@PART

PURPOSE: A BASIC-2 keyword which refers to a global partition.

REFER TO: DEFFN @PART
SELECT @PART

ABS

PURPOSE: Compute the absolute value of a numeric expression.

SYNTAX ... ABS(expression) [...]

COMMENTS: If the value of the expression is negative, ABS changes the sign to positive.

EXAMPLE: 10 X = ABS(Y)

ADD

PURPOSE: Add binary numbers in alpha-variables with optional carry.

SYNTAX receiver = [...] ADD[C] operand [...]

COMMENTS: receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

The sum of the binary values of the operands is stored in the receiver. If an alpha-variable does not immediately follow the =, the receiver is considered an operand in its calculation.

If C is omitted, the characters are added from right to left with no bit carries between characters. When C is included, the value of the operand is considered to be a single, multi-byte binary number and carry propogates between bytes.

Trailing spaces are included as part of an alpha-variable value.

EXAMPLES: 10 A$ = ADD ALL(20)
20 A$ = ADDC HEX(008000)
30 STR(A$,,2)= B$ ADDC C$

ALERT

REFER TO: $ALERT
SELECT

ALL

PURPOSE: Generate an alpha-expression argument consisting of an unlimited number of identical characters; a keyword used in other statements.

SYNTAX receiver = [...] ALL (character) [...]

COMMENTS: receiver = alpha-variable [,alpha-variable] ...

character = A pair of hexadecimal digits, an alpha-variable, or a literal string.

If character is specifed as an alpha-variable or literal-string, only the first byte is replicated by ALL.

EXAMPLE: 10 A$ = ALL(FF)
20 R$ = ALL("0") ADD N$
30 X$ = Y$ OR ALL(Z$)

REFER TO: INIT

DATASAVE DC CLOSE
=SELECT
RETURN CLEAR

AND

PURPOSE: Perform a logical AND function on alpha-operand bits; a logical operator.

SYNTAX receiver = [...] AND operand [...]

COMMENTS: receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

If an alpha-variable does not immediately follow the =, the receiver is considered an operand in the logical AND evaluation.

The corresponding bits of each operand are compared and the result is stored in the receiver. If both are 1, the result is 1. Otherwise the result is 0.

The AND function should not be confused with the logical operator AND used in IF statements.

EXAMPLE: 10 A$ = AND B$
20 C1$,C2$ = STR(D$,5,4) AND ALL(FF)

REFER TO: IF

ARC

PURPOSE: A keyword used to denote an inverse trigonometric function.

REFER TO: ARCCOS
ARCSIN
ARCTAN

ARCCOS

PURPOSE: Calculate the arc cosine of a numeric-expression.

SYNTAX ... ARCCOS(expression) [...]

COMMENTS: The absolute value of the expression must be <= 1. The result can represent degrees, radians, or gradians depending on the current SELECT parameter.

EXAMPLE: 10 X = ARCCOS(Y)

REFER TO: SELECT

ARCSIN

PURPOSE: Calculate the arc sine of a numeric-expression.

SYNTAX ... ARCSIN(expression) [...]

COMMENTS: The absolute value of the expression must be <= 1. The result can represent degrees, radians, or gradians depending on the current SELECT parameter.

EXAMPLE: 10 X = ARCSIN(Y)

REFER TO: SELECT

ARCTAN

PURPOSE: Calculate the arc tangent of a numeric-expression.

SYNTAX ... ARCTAN(expression) [...]

COMMENTS: The absolute value of the expression must be <= 1. The result can represent degrees, radians, or gradians depending on the current SELECT parameter.

The ARCTAN and ATN functions are equivalent.

EXAMPLE: 10 X = ARCTAN(Y)

REFER TO: SELECT

AT

PURPOSE: Move the cursor to a location on the CRT, optionally erasing a number of characters starting at the specified location.

SYNTAX PRINT [...] AT (expression-1, expression-2 [,[expression-3]]) [...]

COMMENTS: expression-1 = The CRT row. Row numbers start at 0.

expression-2 = The CRT column. Columns numbers start at 0.

expression-3 = The number of CRT characters to be erased.

AT must be used with PRINT.

All characters on the CRT, starting at the specified row and column position, are erased if a comma follows expression-2 and expression-3 is omitted.

EXAMPLE:
```
10 PRINT AT(2,4);A
20 PRINT AT(FNX(X),FNY(Y),LEN(A$));A$
30 PRINT AT(5,0,);Q$()
```

REFER TO: PRINT

ATN

PURPOSE: Calculate the arc tangent of a numeric-expression.

SYNTAX ... ATN(expression) [...]

COMMENTS: The absolute value of the expression must be <= 1. The result can represent degrees, radians, or gradians depending on the current SELECT parameter.

The ARCTAN and ATN functions are equivalent.

EXAMPLE: 10 X = ATN(Y)

REFER TO: SELECT

BA

PURPOSE: A parameter denoting "block address" mode in disk operations.

REFER TO: DATALOAD BA
DATASAVE BA

PURPOSE: A Wang BASIC keyword used for tape cassette operations, which are not supported by BASIC-2.

REFER TO: DBACKSPACE

BEG

PURPOSE: A keyword used as a parameter in disk statements.

REFER TO: DBACKSPACE
LOAD DA
LOAD DC

BIN

PURPOSE: Convert the integer portion of an expression to its binary equivalent.

SYNTAX receiver = [...] BIN (expression [,n]) [...]

COMMENTS:

receiver = alpha-variable [,alpha-variable] ...

expression = A numeric expression such that:
0 <= expression <= 255, if n is ommitted; or
0 <= expression <= 2↑n-1, if n is included.

n = A numeric constant such that 1 <= n <= 6.

The n parameter specifies the number of binary bytes the expression will be converted to. If omitted, n is assumed to be 1.

BIN is the inverse of the VAL function.

EXAMPLE:
```
10 A$ = BIN(A)
20 STR(B$,,4) = STR(C$(),5,4) ADDC BIN(X*Y+Z,4)
```

REFER TO: VAL

BOOL

PURPOSE: Perform one of 16 possible logical operations on alpha-arguments.

SYNTAX receiver = [...] BOOLh operand [...]

COMMENTS:

receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

h = A hexadecimal digit (0-9, A-F) specifying the logical operation to be performed.

If an alpha-variable does not immediately follow the =, the receiver is considered an operand in the BOOL evaluation.

h	Logical Function Name	Binary Result of 1100 BOOLh 1010
0	Null	0000
1	Not-OR	0001
2	Operand does not imply receiver	0010
3	Complement of receiver	0011
4	Receiver does not imply operand	0100
5	Complement of operand	0101
6	Exclusive-OR	0110
7	Not-AND	0111
8	AND	1000
9	Equivalence	1001
A	Receiver equals operand	1010
B	Receiver implies operand	1011
C	Operand equals receiver	1100
D	Operand implies receiver	1101
E	OR	1110
F	Identity	1111

EXAMPLES:
```
10 A$ = BOOL9 B$
20 STR(C$,,4) = BIN(2↑4-1-Y,4) BOOL5 ALL(FF)
```

BOX

PURPOSE: Draw or erase a box on the CRT.

SYNTAX PRINT [...] BOX(height, width) [...]

COMMENTS:

height = The height of the box in CRT lines.

width = The width of the box in CRT columns.

The cursor marks the upper left corner of the box. It is not possible to change the current cursor posistion with the BOX function.

When both height and width are positive values, a box is drawn. When both height and width are negative values, a box is erased. The signs of height and width cannot be different.

If the height is zero, and a value for width is specified, a horizontal line is drawn or erased. If the width is zero, and height is specified, a vertical line is drawn or erased.

EXAMPLE:
```
10 PRINT BOX(2,5)
20 PRINT BOX(-5,0)
```

REFER TO: PRINT

BREAK

REFER TO: $BREAK

BT

PURPOSE: An I/O parameter specifying "block tape" operations.

REFER TO: DATALOAD BT

C

PURPOSE: A parameter denoting carry in binary operations.

REFER TO: ADD
SUB

CI

PURPOSE: A parameter which stands for "console input", a class of I/O.

REFER TO: SELECT

CLEAR

PURPOSE: Clear all or a portion of user-memory and optionally reset system functions.

SYNTAX
```
CLEAR [P [line-number] [,[line-number]]]
      [V                               ]
      [N                               ]
```

COMMENTS: CLEAR with no parameters:

1) Clears all of user-memory;
2) Sets the PRINT and LIST devices to the current CO device;
3) Sets the INPUT device to the CI device;
4) Turns off PAUSE and TRACE modes;
5) Flushes the system stacks;
6) Zeroes slots #1 - #15 in the device table;
7) Clears the interrupt table;
8) Clears the CRT and displays the READY and $MSG messages;
9) Closes all $OPENed devices; and
10) Reseeds the random number generator.

If the P parameter is specified, only program text is cleared. If either one or both line-numbers are specified, only a portion of program text is cleared.

If the N parameter is specified, only non-common variables are cleared.

If the V parameter is specified, all variables are cleared.

EXAMPLES:
```
CLEAR
CLEAR P
CLEAR P 100
CLEAR P 100,500
CLEAR P ,500
CLEAR N
CLEAR V
```

CLOSE

PURPOSE: A keyword used to specify the closing of a file.

REFER TO: DATASAVE DC CLOSE

$CLOSE

CO

PURPOSE: A parameter which stands for "console output", a class of I/O.

REFER TO: SELECT

COM

PURPOSE: Define and allocate storage for common variables.

SYNTAX COM com-element [,com-element] ...

COMMENTS:

```
com-element = numeric-scaler-variable
              numeric-array-name (dim1 [,dim2])
              alpha-array-name   (dim1 [,dim2]) [length]
              alpha-scaler-variable [length]

dim1, dim2 = an integer or numeric-scaler-variable such that:
             1 <= dim1 <= 65535 for one-dimensional arrays
             1 <= dim1, dim2 <= 255 for two-dimensional arrays

    length = an integer or numeric-scaler-variable such that:
             1 <= length <= 124
```

Common variables are preserved in memory until: execution of CLEAR, CLEAR V, and LOAD RUN commands; execution of a LOAD RUN statement; or master initialization.

EXAMPLE: 10 COM N,N(10),A$20,A$(X,Y)L

REFER TO: DIM

COM CLEAR

PURPOSE: Move the common-variable-pointer.

SYNTAX
```
COM CLEAR [scalar-variable ]
          [array-designator]
```

COMMENTS: Variables are added to the variable-table in the order that they are defined. The common-variable-pointer separates common and non-common variables in the table such that variables occurring before the pointer are considered common, those occurring after are non-common.

If a variable is specified, the common-variable-pointer is moved to the location in the table just prior to the specified variable. The specified variable and all variables occuring after it are redefined non-common. Variables occuring before it are redefined common.

When a variable is not specified, all currently defined variables are redefined as non-common.

EXAMPLES: 10 COM CLEAR
20 COM CLEAR A

REFER TO: COM

CON

PURPOSE: A keyword used in the matrix constant statement.

REFER TO: MAT CON

CONTINUE

PURPOSE: Continue program execution after a STOP statement or HALT command.

SYNTAX CONTINUE

COMMENTS: Execution begins at the statement following the last-executed statement.

CONTINUE must be the last command on a line when multiple statements and commands appear on an immediate mode line.

CONTINUE may not be used if:
1) Execution was halted by an untrapped error;
2) A new variable has been defined in immediate mode;
3) Program text has been renumbered;
4) A CLEAR command or RESET has been executed; or
5) Program text has not been resolved with RUN.

EXAMPLE: CONTINUE
$RELEASE PART: CONTINUE

CONVERT

PURPOSE: Convert numeric values between ASCII and internal numeric format.

SYNTAX Form 1: CONVERT alpha-variable TO numeric-variable

Form 2: CONVERT expression TO alpha-variable, (format)

COMMENTS: format = An image specification expressed as a literal without quotes, or contained in alpha variable. Formats must correspond to the following format:

```
[+][$][#[,]] ... [.][#] ... [↑↑↑↑][+ ]
[-]                          [- ]
                             [++]
                             [--]
```

At least one # is required. The number of characters cannot exceed 254 and imbedded spaces are ignored. Only one sign designation can occur. If omitted, the absolute value is assumed.

Form 1 requires that the number in the alpha-variable be represented in legal BASIC-2 format. Numbers converted using Form 2 cannot

necessarily be reconverted using FORM 1.

Form 2 Sign Conversion Table

Format Sign	Value is Positive or Zero	Value is Negative
+	+	-
-	space	-
++	two spaces	CR
--	two spaces	DB

EXAMPLES: Form 1:

```
10 CONVERT X$ TO X
20 CONVERT STR(X$(2),4) TO Y(3)
```

Form 2:

```
10 CONVERT X TO X$, (###)
20 CONVERT G(1) TO G$(1), ($#,###,###.##--)
30 CONVERT ROUND((A+B)/C,2) TO D$, (F$)
```

COPY

PURPOSE: Copy all or a portion of a disk platter to another platter.

SYNTAX

```
COPY pd [file#,  ] [(start, end)] TO pd [file#,  ] [(destination)]
        [address,]                      [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

start = A numeric-expression indicating the first sector to be copied.

end = A numeric-expression indicating the last sector to be copied.

destination = A numeric-expression indicating the start sector of the destination platter to which data will be copied.

If file# and address are omitted, the current DISK device is assumed.

If start and end are omitted, start is assumed to be 0, and end is the number of the last sector used in the disk catalogue area. Start and end are required on non-catalogued disks.

If destination is omitted, it is assumed to be the same as start.

COPY requires at least 800 bytes of unoccupied partition memory for buffering. Otherwise an error A03 results.

The source-platter and destination-platter may be the same.

Data transferred to the destination-platter may be validated with VERIFY.

EXAMPLES: 10 COPY R TO F

```
          20 COPY #1, TO T
          30 COPY T/D11, (S,E) TO T#X, (D)

REFER TO: VERIFY
          MOVE
```

COS

```
PURPOSE:  Calculate the cosine of a numeric-expression.

SYNTAX    ... COS(expression) [...]

COMMENTS: The absolute value of the expression must be less than 1E10 and can
          represent degrees, radians, or gradians depending on the current
          SELECT parameter.

EXAMPLE:  10 X = COS(Y)

REFER TO: SELECT
```

D

```
PURPOSE:  A LIST and SELECT parameter.

REFER TO: LIST
          SELECT
```

DA

```
PURPOSE:  A parameter denoting "direct address" mode in disk operations.

REFER TO: DATALOAD DA
          DATASAVE DA
          LOAD DA
          SAVE DA
```

DAC

```
PURPOSE:  Add unsigned packed decimal numbers with carry.

SYNTAX    receiver = [...] DAC operand [...]

COMMENTS:       receiver = alpha-variable [, alpha-variable] ...

                operand = An alpha-variable, literal-string, ALL or BIN.

          The sum of the packed decimal values of the operands is stored in the
          receiver.  If an alpha-variable does not immediately follow the =,
          the receiver is considered an operand in its calculation.

          The packed decimal format of the operands is not verified.  If they
          do not contain valid numbers, the results are undefined.

          Addition occurs from right to left and carry propagates between
```

bytes.

Trailing spaces are included in alpha-variables.

EXAMPLES:
```
10 S$ = DAC HEX(01)
20 A$ = B$(1) DAC B$(2)
30 STR(X$,5) = DAC Y$ DAC Z$
```

DATA

PURPOSE: Provide values assigned to variables in READ and MAT READ statements; a keyword used in disk statements.

SYNTAX
```
DATA n [,n] ...
```

COMMENTS:

n = A number, alphanumeric literal, or HEX(literal.

DATA statements are not executed and can be placed anywhere in the program. DATA values must agree with the variable type in the READ or MAT READ statement.

DATA may not be used in immediate mode.

EXAMPLE:
```
10 DATA 1,"SDS",HEX(0102)
```

DATALOAD

PURPOSE: A keyword denoting a disk operation; a Wang BASIC verb used to access tape cassettes, card readers, punched tape readers, and Teletypes. Only the mark sense card reader is supported by BASIC-2.

SYNTAX
```
Mark sense card reader
DATALOAD [file#,] argument [,argument] ...
         [/628  ]
```

REFER TO:
```
DATALOAD BA
DATALOAD BT
DATALOAD DA
DATALOAD DC
DATALOAD DC OPEN
```

Mark Sense Card Reader Manual

DATALOAD BA

PURPOSE: Load a 256-byte disk sector into a variable in "block address" mode.

SYNTAX
```
DATALOAD BA pd [file#,  ] (sector-addr [, return-var]) alpha-array
               [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

sector-addr = A numeric-expression or alpha-variable designating the address of the sector to be accessed. If an alpha-variable, the first two bytes are treated as a 16-bit, unsigned binary value.

return-var = A numeric-variable or an alpha-variable which receives the sector address of the sector following sector-addr.

The alpha-array must be at least 256 bytes in size. If larger, only the first 256 bytes are affected by DATALOAD BA.

DATALOAD BA does not update the device table.

EXAMPLES:
```
10 DATALOAD BA T (S) X$()
20 DATALOAD BA T /D11, (S$,S$) Y$()
30 DATALOAD BA T #X(4), (N+3*M) STR(Z$(),100)
```

REFER TO: DATASAVE BAT

DATALOAD BT

PURPOSE: A special purpose I/O statement used to access tape cassettes, card readers, punched tape readers, and Teletypes. Only the mark sense card reader is supported by BASIC-2.

SYNTAX Mark sense card reader

```
DATALOAD BT [([N=exp][,L=hh ][,S=hh ])] [file#,  ] alpha-variable
                              var     var    [address,] array-designator
```

REFER TO: The Mark Sense Card Reader Manual
DATASAVE BT

DATALOAD DA

PURPOSE: Read a logical record in "direct address" mode.

SYNTAX
```
DATALOAD DA pd [file#,  ] (sector-addr [,return-var]) arg [,arg] ...
               [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

sector-addr = A numeric-expression or alpha-variable designating the sector address of the logical record. If an alpha-variable, the first two bytes are treated as a 16-bit, unsigned binary value.

return-var = A numeric-variable or an alpha-variable which

receives the sector address of the sector following the last sector read.

arg = A variable or array-designator.

DATALOAD DA does not update the device table.

EXAMPLES:
```
10 DATALOAD DA T (S) A,B$,C$()
20 DATALOAD DA T /D11, (S$,S$) X$(),Y$()
30 DATALOAD DA T #X(4), (N+3*M) P(),STR(Q$(),,20)
```

REFER TO: DATASAVE DA

DATALOAD DC

PURPOSE: Read a logical record in disk catalogue mode.

SYNTAX
```
DATALOAD DC [file#,] arg [,arg] ...
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

arg = A variable or array-designator.

DATALOAD DC reads a logical record starting at the sector specified by the current-sector-address field in the device-table-slot. The current-sector-address is updated to point to the sector following the last sector read by DATALOAD DC.

Logical records are assumed to be in Wang format and must have been saved with a DATASAVE DC or DATASAVE DA statement, or with a $PACK/DATASAVE BA combination.

EXAMPLES:
```
10 DATALOAD DC A,B$,C$()
20 DATALOAD DC #X, D(),E$()
```

REFER TO: DATASAVE DC

DATALOAD DC OPEN

PURPOSE: Open an existing or temporary disk file for DC access.

SYNTAX
```
DATALOAD DC OPEN pd [file#,] file-name
                            TEMP [,] start, end
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

file-name = A literal-string or alpha-variable specifying a catalogued data-file-name.

start = A numeric-expression specifying the start sector of the temporary work file.

end = A numeric-expression specifying the last secotr of the temporary work file.

A D82 error results if file-name is not an active data file on the specified platter.

DATALOAD DC OPEN sets up parameters in the device-table-slot for later use by other DC disk statements.

EXAMPLES: 10 DATALOAD DC OPEN T "DATA"
20 DATALOAD DC OPEN F #N-1, TEMP X, X+100

REFER TO: DATASAVE DC OPEN

DATASAVE

PURPOSE: A keyword denoting a disk operation; a Wang BASIC verb used to access tape cassettes and Teletypes, neither of which is supported by BASIC-2.

REFER TO: DATASAVE BA
DATASAVE BT
DATASAVE DA
DATASAVE DC
DATASAVE DC CLOSE
DATASAVE DC END
DATASAVE DC OPEN

DATASAVE BA

PURPOSE: Save data in a 256-byte disk sector in "block address" mode.

SYNTAX DATASAVE BA pd [$][file#,] (sector-addr [,return-var]) alpha-array [address,]

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

sector-addr = A numeric-expression or alpha-variable designating the address of the sector to be written to. If an alpha-variable, the first two bytes are treated as a 16-bit, unsigned binary value.

return-var = A numeric-variable or an alpha-variable which receives the sector address of the sector following sector-addr.

$ = A parameter specifying that a read-after-write verification is to be performed.

If the alpha-array is shorter than 256 bytes, the remainder of the sector is filled with undefined data. If larger, only the first 256 bytes are written by DATASAVE BA.

DATASAVE BA does not update the device table.

An I99 error results if the read-after-write fails.

EXAMPLES:
```
10 DATASAVE BA T (S) X$()
20 DATASAVE BA T $ /D11, (S$,S$) Y$()
30 DATASAVE BA T #X(4), (M+SGN(N)*3) STR(Z$(),,200)
```

REFER TO: DATALOAD BA

DATASAVE BT

PURPOSE: A special purpose I/O statement used to access tape cassettes, punched card readers, and Teletypes, none of which are supported by BASIC-2.

DATASAVE DA

PURPOSE: Write a logical record in "direct address" mode.

SYNTAX
```
DATASAVE DA pd [$][file#,  ] (sector-addr[,return-var]) arg[,arg]...
                    [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

sector-addr = A numeric-expression or alpha-variable designating the start-sector-address to write the logical record. If an alpha-variable, the first two bytes are treated as a 16-bit, unsigned binary value.

return-var = A numeric-variable or an alpha-variable which receives the sector address of the sector following the last sector written.

arg = A numeric-expression, literal-string, variable or array-designator.

$ = A parameter specifying that a read-after-write verification is to be performed.

DATASAVE DA does not update the device table.

Logical records are written in Wang format and may span sectors.

An I99 error results if the read-after-write verification fails.

EXAMPLES:
```
10 DATASAVE DA T (S) A/A0,B$,C$()
20 DATASAVE DA T $ /D11, (S$,S$) X$(),Y$()
30 DATASAVE DA T #X(4), (N+3*M) P(),STR(Q$(),,20)
```

REFER TO: DATALOAD DA

DATASAVE DC

PURPOSE: Write a logical record in disk catalogue mode.

SYNTAX DATASAVE DC [$] [file#,] arg [,arg] ...

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

arg = A numeric-expression, literal-string, variable or array-designator.

$ = A parameter specifying that a read-after-write verification is to be performed.

DATASAVE DC writes a logical record starting at the sector specified by the current-sector-address field in the device-table-slot. The current-sector-address is updated to point to the sector following the last sector written by DATASAVE DC.

Logical records are written in Wang format and may span sectors.

An I99 error results if the read-after-write verification fails.

EXAMPLES:
```
10 DATASAVE DC FNX(A),B$,C$()
20 DATASAVE DC $ #X, D(),E$()
```

REFER TO: DATALOAD DC

DATASAVE DC CLOSE

PURPOSE: Close one or all disk files which are open in the device-table.

SYNTAX
```
DATASAVE DC CLOSE [file#]
                  [ ALL ]
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15, indicating which open file is to be closed. If omitted, slot #0 is assumed.

ALL = A parameter indicating that all open files are to be closed.

Only the start, end, and current sector pointers are zeroed in the device-table. Disk addresses are not modified by DATASAVE DC CLOSE.

EXAMPLES:
```
10 DATASAVE DC CLOSE
20 DATASAVE DC CLOSE #X
30 DATASAVE DC CLOSE ALL
```

DATASAVE DC END

PURPOSE: Write a data trailer record in a currently open data file.

SYNTAX DATASAVE DC [$] [file#,] END

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

END = A parameter indicating a data trailer record is to be written.

$ = A parameter specifying that a read-after-write verification is to be performed.

DATASAVE DC END writes a data trailer (end) record at the sector specified by the current-sector-address field in the device-table. DATASAVE DC END also updates the sectors used count in the last sector of the file (file trailer).

EXAMPLES:
```
10 DATASAVE DC END
20 DATASAVE DC$ #X, END
```

REFER TO: IF END

DATASAVE DC OPEN

PURPOSE: Create a disk-catalogued or temporary data file.

SYNTAX
```
DATASAVE DC pd [$] [file#,] ([old-name]) new-name
                            (space     ) new-name
                             TEMP [,]    start-sector, end-sector
```

COMMENTS:

pd = Platter-designator (F, R or T).

$ = A parameter specifying that a read-after-write verification is to be performed.

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

old-name = The name of a scratched disk catalogued file to be reopened under a new name. If omitted, the new-name is assumed.

new-name = The disk catalogue name to be assigned to a new or reopened data file.

space = A numeric-expression specifying the size of a new data file in sectors.

TEMP = A parameter indicating that a temporary data file is to be declared in the device table using the specified start-sector and end-sector.

start-sector = A numeric-expression specifying the starting sector address of a TEMP file. The address must be beyond the platter's catalogue end.

end-sector = A numeric-expression specifying the ending sector address of a TEMP file. The end-sector must be greater than start-sector and cannot exceed the

highest allowable sector-address of the disk platter.

DATASAVE DC OPEN updates the device-table.

The last sector of the file is reserved for the file trailer. Consequently, available room is one less than space.

EXAMPLES:
```
10 DATASAVE DC OPEN T (10)"DATA"
20 DATASAVE DC OPEN F$#X, () "DATA"
30 DATASAVE DC OPEN T/D11,("OLDNAME")"NEWNAME"
40 DATASAVE DC OPEN R TEMP X,Y
```

REFER TO: DATALOAD DC OPEN

DATE

PURPOSE: A system variable used to retrieve or modify the system date.

SYNTAX
```
Assignment
DATE = argument-1 PASSWORD argument-2

Retrieval
Alpha-variable = [...] DATE [...]
```

COMMENTS:

argument-1 = An alpha-variable or literal-string containing the date to be assigned in YYMMDD (year, month, day) in ASCII format.

argument-2 = An alpha-variable or literal-string containing the system password assigned by $INIT. If the password supplied is incorrect, an X79 error is signaled.

The system date can be maintained and modified regardless of whether or not a system clock is available. However, the date will not be automatically incremented at midnight.

The DATE function may not be used as an argument in another function, nor may DATE be used as an argument in a GOSUB' or DEFFN' statement

EXAMPLES:
```
10 DATE = "850101" PASSWORD "SYSTEM"
20 DATE = A$ PASSWORD B$
30 D$ = DATE
40 X$ = "Today's date: "&DATE
```

DBACKSPACE

PURPOSE: Decrement the current-sector-pointer in the device table.

SYNTAX
```
DBACKSPACE [file#,] expression [S]
                    BEG
```

COMMENTS:

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

expression = A numeric-expression specifying the decrement.

S = A parameter specifying that the decrement is to be

in sectors. If omitted, the decrement is in logical-records.

BEG = A parameter specifying that the current-sector is to be set equal to the start-sector in the device-table.

If the BEG or S parameters are used, DBACKSPACE does not access the disk. If neither is used, the system reads the disk file and sets the current-sector-pointer at the beginning of a logical-record.

If an attempt is made to backspace prior to the beginning of the file, the result is the same as if the BEG parameter was used.

EXAMPLES:
```
10 DBACKSPACE BEG
20 DBACKSPACE #1,X+3
30 DBACKSPACE #F,20S
```

REFER TO: DSKIP

DC

PURPOSE: A parameter denoting "disk catalogue" mode in disk operations.

REFER TO:
```
DATALOAD DC
DATALOAD DC OPEN
DATASAVE DC
DATASAVE DC CLOSE
DATASAVE DC END
DATASAVE DC OPEN
LOAD DC
SAVE DC
LIST DC
```

DEFFN

PURPOSE: Define a user-written function.

SYNTAX DEFFN a(v) = numeric-expression

COMMENTS:

a = A letter (A-Z) or digit (0-9).

v = A "dummy" numeric-scalar-variable.

The letter or digit is an identifying label used to match the user-defined function with its corresponding FN references.

The "dummy" variable specifies which variable in the expression will be replaced by the value of the argument passed to DEFFN by its FN reference. A numeric-scaler-variable of the same name occuring in the program is not modified.

The expression may contain FN references to other DEFFN functions. Nesting of functions in this manner may not exceed five levels.

EXAMPLES:
```
10 DEFFNA(Q) = Q↑2/X
20 DEFFN1(Q) = 5+FNA(Q)
```

REFER TO: FN

PURPOSE: Define a global partition.

SYNTAX DEFFN @PART part-name [FOR terminal# [,terminal#] ...]

COMMENTS:

part-name = A literal-string or alpha-variable specifying up to an 8-character name of the partition. If the name is blank, the partition is NOT declared global.

terminal# = A numeric-expression denoting a terminal which may access the partition. If not specified, all terminals may use the partition.

Variables explicitly or implicitly declared prior to the location of the DEFFN @PART in the program are entered into the partition's variable table during program resolution. Others are assumed to be part of global text.

Multiple DEFFN @PART statements are allowed, the last one executed being the one in effect.

Partitions in different banks can use the same global name.

EXAMPLE:
```
10 DEFFN @PART "GLOBAL"
20 DEFFN @PART A$ FOR 1,T(1),T(2)
```

REFER TO: SELECT @PART

DEFFN'

PURPOSE: Assign a character string to a Special Function Key (SFK) or define a marked subroutine entry point.

SYNTAX
```
DEFFN' integer   [literal-string [;literal-string] ...]
                 [(variable [,variable] ...)]
```

COMMENT:

integer = An integer such that 0 <= integer <= 255

literal-string = Text enclosed in quotes or a HEX(literal.

variable = Numeric-scaler-variable
Alpha-scaler-variable
Numeric-array-element
Alpha-array-element
Alpha-array-designator

If a literal-string is assigned to a DEFFN', and a SFK is pressed which outputs the decimal equivalent of the integer, the system outputs the corresponding literal-string as if it was directly keyed from the keyboard. The function is available in immediate mode and while processing INPUT and LINPUT statements in run-time mode. The function is supressed for those SFKs that double as editing keys when the system is in edit mode (either in immediate mode or during LINPUT).

If a literal-string is not assigned, the DEFFN' is assumed to be a subroutine entry point. In immediate mode, while processing an INPUT statement, or while processing a LINPUT statement in text entry mode,

the subroutine may be executed directly by keying the corresponding SFK. If the DEFFN' is accompanied by a variable list, the system will execute an INPUT to receive values for the variables.

Marked subroutines serving as entry points may also be accessed by GOSUB' statements.

Whether accessed via a SFK or a GOSUB', the number and type of data items (numeric or alphanumeric) passed to the optional variable list must be identical or an error will result.

EXAMPLES:
```
10 DEFFN '30"LISTSD";HEX(0D)
20 DEFFN '50
30 DEFFN '99(A,B$,C(1),D$(A),E$())
```

REFER TO: GOSUB'

DIM

PURPOSE: Define and allocate storage for non-common variables.

SYNTAX DIM dim-element [,dim-element] ...

COMMENTS:
```
dim-element = numeric-scaler-variable
              numeric-array-name (dim1 [,dim2])
              alpha-array-name   (dim1 [,dim2]) [length]
              alpha-scaler-variable [length]

dim1, dim2 = an integer or numeric-scaler-variable such that:
             1 <= dim1 <= 65535 for one-dimensional arrays
             1 <= dim1, dim2 <= 255 for two-dimensional arrays

    length = an integer or numeric-scaler-variable such that:
             1 <= length <= 124
```

EXAMPLES:
```
10 DIM X(20)
20 DIM A,B$10,A(10),B$(X,5)L
```

REFER TO: COM

DISCONNECT

REFER TO: $DISCONNECT

DISK

PURPOSE: A keyword used in disk device assignments and disk operations.

REFER TO: SELECT

```
$FORMAT DISK
SCRATCH DISK
TRACE DISK
```

DSC

PURPOSE: Subtract unsigned packed decimal numbers with carry.

SYNTAX receiver = [...] DSC operand [...]

COMMENTS: receiver = alpha-variable [, alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

The difference of the packed decimal values of the operands is stored in the receiver. If an alpha-variable does not immediately follow the =, the receiver is considered an operand in its calculation.

The packed decimal format of the operands is not verified. If they do not contain valid numbers, the results are undefined.

Subtraction occurs from right to left and carry propogates between bytes.

Trailing spaces are included in alpha-variables.

EXAMPLES:
```
10 S$ = DSC HEX(01)
20 A$ = B$(1) DSC B$(2)
30 STR(X$,5) = DSC Y$ DSC Z$
```

DSKIP

PURPOSE: Increment the current-sector-pointer in the device table.

SYNTAX DSKIP [file#,] expression [S]
END

COMMENTS: file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

expression = A numeric-expression specifying the increment.

S = A parameter specifying that the increment is to be in sectors. If omitted, the increment is in logical-records.

END = A parameter specifying that the current-sector is to be set equal to the address of the data-trailer.

If the S parameter is used, DSKIP does not access the disk. If omitted, the system reads the disk file and sets the current-sector-pointer at the beginning of a logical-record.

A data-trailer record must have been written into the file with DATASAVE DC END if the END parameter used. Otherwise, a D87 error results.

If an attempt is made to skip past the end of the file, the current-sector-pointer is set to the address of the last sector of the file.

EXAMPLES:
```
10 DSKIP END
20 DSKIP #1,R
30 DSKIP #F,10S
```

REFER TO: DBACKSPACE

DT

PURPOSE: A LIST parameter.

REFER TO: LIST DT

ELSE

PURPOSE: A keyword used to specify that the following statement is to be executed if a previous IF or ON statement fails.

REFER TO: IF
IF END
ON GOSUB
ON GOTO
ON SELECT

END

PURPOSE: A statement which ends program execution and displays the amount of free space in memory; a keyword used in disk statements.

SYNTAX END

COMMENTS: In immediate mode (command), END displays:

1) If executed prior to program resolution, the memory available not including non-common variables defined in the program; or

2) If executed after program resolution, the memory available including all variable allocations.

The operator and value stacks remain intact.

In run-time mode (statement), execution of END:

1) Halts program execution;
2) Flushes the operator and value stacks; and
3) Displays the memory available including all variable allocations.

EXAMPLES: END
100 END

REFER TO: IF END
DSKIP
SCRATCH DISK

ERR

PURPOSE: A built-in numeric-function which returns the value of the last error condition encountered.

SYNTAX ERR [ ]

SYNTAX ... ERR [...]

COMMENTS: The value returned is such that 0 <= ERR <= 99. A value of zero indicates no error.

Execution of the ERR function, a RUN command, or a CLEAR command resets the value of ERR to zero.

EXAMPLES:
```
10 E = ERR
20 IF ERR >= 90 THEN STOP "Device is not ready"
30 X$ = ERR$(ERR)
```

REFER TO: ERR$

ERR$

PURPOSE: A built-in alpha-numeric function that returns the standard descriptive error message for a given error code.

SYNTAX alpha-variable = ERR$(error-code)

COMMENTS: error-code = A numeric-expression such that
1 <= error-code <= 99

The ERR$ function may not be used as an argument in another function, nor may ERR$ be used as an argument in a GOSUB' statement.

EXAMPLES:
```
10 X$ = ERR$(82)
20 Y$ = ERR$(ERR)
```

REFER TO: ERR

ERROR

PURPOSE: Provides a means of trapping recoverable errors during execution.

SYNTAX statement: ERROR statement [: statement] ...

COMMENTS: ERROR can be used to trap recoverable errors only, which includes all "C", "X", "D", and "I" errors plus P48.

ERROR must occur as the next statement following the statement causing the error, and must be on the same line.

If an ERROR statement is encountered, and the previous statement has executed properly, a branch is taken to the next line-number. Consequently, all statements following ERROR are considered to be part of ERROR processing.

The SELECT ERROR statement provides default values for certain computational errors. If defaults are used, ERROR will be ignored for those errors.

ERROR cannot be used in immediate mode.

EXAMPLES:
```
10 DATALOAD DC X$():ERROR E$=ERR$(ERR):PRINT E$:LOAD T "END"
20 LINPUT "DEVICE"D$:SELECT #1<D$>:ERROR PRINT"INVALID":GOTO 20
```

REFER TO: SELECT
ON ERROR

EXP

PURPOSE: Compute the value of e raised to the power of a numeric-expression.

SYNTAX ... EXP(expression) [...]

COMMENTS: The value of e = 2.718281828459 = EXP(1).

If less than zero, expression must be an integer.

EXAMPLE: 10 X = EXP(Y)

REFER TO: LOG

F

PURPOSE: A platter-designator parameter which refers to the "fixed" platter in disk operations.

COMMENTS: The F parameter may be used to over-ride the "B" device type in a SELECTed disk address, but may not be used with "D" addresses.

FIX

PURPOSE: Compute the integer portion of a numeric expression.

SYNTAX ... FIX(expression) [...]

COMMENTS: FIX truncates the fractional portion of the expression, leaving the integer. If the value of the expression is X, then FIX computes: SGN(X) * INT(ABS(X))

EXAMPLE: 10 X = FIX(Y)

REFER TO: INT

FN

PURPOSE: Reference a user-defined function.

SYNTAX ... FNid(numeric-expression) [...]

COMMENTS:

id = A letter (A-Z) or digit (0-9) corresponding to the name of the function defined by a corresponding DEFFN statement.

The numeric-expression is passed to the dummy variable in DEFFN. The

result of the evaluation of the expression in the DEFFN is returned.

EXAMPLES:
```
10 X = FNX(0)
20 Y = FN0(A+B) * FN1(FN2(Q))
```

REFER TO: DEFFN

FOR

PURPOSE: Define a program loop; specify valid terminals in DEFFN @PART.

SYNTAX FOR counter = expression-1 TO expression-2 [STEP expression-3]

COMMENTS:

counter = A numeric-scaler-variable used as a loop counter.

expression-1 = A numeric-expression which specifies the initial value of the counter.

expression-2 = A numeric-expression which specifies the value the counter may not surpass.

expression-3 = A numeric-expression which specifies the increment added to the counter when a corresponding NEXT statement is executed. If omitted, the STEP value is assumed to be 1.

All loops are executed at least once regardless of the value of the expressions. A negative STEP value is allowed. Loops may be nested within other loops.

EXAMPLE:
```
10 FOR A= 1 TO X(X)              10 FOR B=5 TO 2 STEP -1
20 ON A GOSUB 100,200,300        20 B$(B) = B$(B-1)
30 NEXT A                        30 NEXT B
```

REFER TO: DEFFN @PART

FORMAT

REFER TO: $FORMAT
$FORMAT DISK

FROM

PURPOSE: A keyword used as a separator in some BASIC-2 statements.

REFER TO: $PACK
PACK

G

PURPOSE: A SELECT parameter indicating gradians mode is to be used in trigonometric functions.

REFER TO: SELECT

REFER TO: $GIO

GOSUB

PURPOSE: Execute a subroutine (go to subroutine).

SYNTAX GOSUB line-number

COMMENTS: line-number = The program line-number marking the beginning of the subroutine.

GOSUB causes execution to be transferred to the specified line number. Upon encountering a RETURN statement, the system transfers execution to the statement immediately following the GOSUB. Subroutines may call other subroutines.

```
EXAMPLE: 10 GOSUB 40
         20 PRINT "SUBROUTINE EXECUTED"
         30 END
         40 PRINT "SUBROUTINE";
         50 PRINT "EXECUTING"
         60 RETURN
```

REFER TO: RETURN

GOSUB'

PURPOSE: Branch to a subroutine marked by a DEFFN' statement.

SYNTAX GOSUB' integer [(argument [,argument] ...)]

COMMENTS: integer = An integer constant such that 0 <= integer <= 255.

argument = A numeric constant, scaler-variable, or array-element.
An alpha literal, scaler-variable, array-element, or array designator
A numeric expression

GOSUB' causes the program to branch to the line containing a correspondingly numbered DEFFN' statement. If the DEFFN' cannot be located in the executing partition, an attempt will be made to locate the marked subroutine in the currently defined global partition.

Upon encountering a RETURN statement, the system transfers execution to the statement immediately following the GOSUB'. Subroutines may call other subroutines.

GOSUB' can optionally transfer values of the argument list to variables defined in the DEFFN' statement. The values are transferred sequentially. The type of value (numeric or alpha numeric) must be the same as the type of receiving variable.

```
EXAMPLE: 10 GOSUB'10(40,"Enter Your Initials")
         20 STOP
```

```
40 PRINT TAB(X);:LINPUTX$,-Y$
50 RETURN
```

REFER TO: DEFFN'
RETURN

GOTO

PURPOSE: Transfer program execution to a specified line-number.

SYNTAX GOTO line-number

COMMENTS: line-number = Any valid line number in the program.

GOTO is allowed in immediate mode. In this case, GOTO transfers the program to the given line-number, but the statement is not executed until the HALT/STEP key is pressed or the program is CONTINUEd.

EXAMPLE: GOTO 100

REFER TO: ON GOTO
ON ERROR

HALT

PURPOSE: HALT program execution or a listing; STEP through program execution.

SYNTAX HALT key

COMMENTS: HALT stops program execution and returns control to the CI device at the completion of the currently executing statement. HALTed programs may be resumed with CONTINUE. (See CONTINUE for restrictions.)

Program execution may be single-stepped using HALT.

A listing may be terminated using HALT. However it may not be resumed.

Under certain conditions (hanging I/O), HALT may not be effective as the currently executing statement may not be able to complete. In this case, RESET must be used to regain control.

REFER TO: CONTINUE

HEX

PURPOSE: Specify any eight-bit code as a HEX literal.

SYNTAX ... HEX(hh[hh...]) [...]

COMMENTS: h = A hexadecimal digit (0-9, A-F)

The HEX literal may be used anywhere an alphanumeric literal imbedded in quotes may be used.

The number of hexadecimal digits must be even. Each pair of digits corresponds to a one-byte character. Spaces may not be imbedded in the digit string.

The HEX literal may be used to supply ASCII code for characters that do not exist on the keyboard.

EXAMPLES:
```
10 MAT SEARCH A$, =HEX(0000) TO Q$()
20 B$ = HEX(0202020F)
30 PRINTUSING 30, HEX(0E),"Grand Total",T
```

HEXOF

PURPOSE: Print a literal-string or alpha-variable in hexadecimal notation.

SYNTAX
```
PRINT [...] HEXOF (alpha-variable) [...]
                  (literal-string)
```

COMMENTS: HEXOF must be used with PRINT.

Trailing spaces are included in the value of the alpha-variable.

EXAMPLES:
```
10 PRINT HEXOF(A$(1)),HEXOF(A$(2))
20 PRINT HEXOF(STR(A$,X,Y))
```

REFER TO: PRINT

HEXPACK

PURPOSE: Convert an ASCII string of hex digits to their binary equivalent.

SYNTAX
```
HEXPACK alpha-var-1 FROM alpha-var-2
```

COMMENTS: HEXPACK converts the ASCII hex digits in alpha-var-2 to their half-byte binary equivalent and stores them in alpha-var-1.

Conversion Table

ASCII Character	Converted Binary Value	ASCII Character	Converted Binary Value
0	0000	8	1000
1	0001	9	1001
2	0010	A or :	1010
3	0011	B or ;	1011
4	0100	C or <	1100
5	0101	D or =	1101
6	0110	E or >	1110
7	0111	F or ?	1111

EXAMPLE:
```
10 HEXPACK A$() FROM H$()
```

REFER TO: HEXUNPACK

HEXPRINT

PURPOSE: Print the hex value of a variable in hexadecimal notation.

PURPOSE: Print the contents of an alpha-variable in hexadecimal.

SYNTAX HEXPRINT [-] alpha-variable [, alpha-variable] ...
[+] literal-string [, literal-string]

COMMENTS: - = A parameter specifying that the hexadecimal output is to be decompressed.

+ = A parameter specifying that the hexadecimal output is to be decompressed and printed with its ASCII translation.

EXAMPLES:
```
10 HEXPRINT A$
20 HEXPRINT B$(1),"XYZ"
30 HEXPRINT - X$()
40 HEXPRINT + Y$(),Z$()
```

REFER TO: HEXOF

HEXUNPACK

PURPOSE: Convert binary values into a string of ASCII hexadecimal characters.

SYNTAX HEXUNPACK alpha-var-1 TO alpha-var-2

COMMENTS: alpha-var-1 = An alpha-variable containing the binary values. Bits are processed from left to right in groups of four, each group being converted to a character.

alpha-var-2 = An alpha-variable which receives the converted character string, and which has a length at least twice that of alpha-var-1.

EXAMPLE: HEXUNPACK H$() TO A$()

REFER TO: HEXPACK

I

PURPOSE: A parameter used with LIST to display interrupt status.

REFER TO: LIST I

ID

REFER TO: #ID
#ID'
$ID

IDN

PURPOSE: A keyword used in matrix math to denote an identity matrix.

REFER TO: MAT IDN

PURPOSE: Execute a statement or branch to a line-number if a given condition is true.

SYNTAX IF condition THEN statement [:ELSE statement]
line-number

COMMENTS: statement = Any executable statement (DEFFN, % (image), and DATA are among the non-executable statements.)

line-number = Any valid line-number

condition = One or more relations separated by logical operators of the form:
relation [logical-operator relation] ...

logical operator = One of the keywords AND, OR, or XOR.

relation = A logical expression of the form:
operand relational-operator operand

operand = A literal-string, alpha-variable, or numeric-expression.

relational operator = Either =, <>, <, <=, >, or >=.

If the evaluated condition is true, the statement following THEN is executed. If a line-number follows THEN, an implied GOTO is assumed and a branch is taken.

If the condition is false, the optional ELSE statement is executed if present. If ELSE is omitted, no action takes place and execution continues with the following statement. Either statement can be another IF statement.

Evaluation of the condition is from left to right with the results of previous relations used as the left relational value (true or false) in the evaluation of succeeding logical expressions using logical-operators.

Operands appearing on either side of relational operators must be of the same type (numeric or alphanumeric).

An ELSE statement must appear on the same line and immediately following the IF statement.

EXAMPLES:
```
10 IF X<=R THEN 145
20 IF A$<B$ AND B$<C$ THEN D$=X$
30 IF X>Y THEN PRINT "GREATER": ELSE PRINT "LESS"
```

IF END

PURPOSE: Conditionally execute a statement based on the status of the end-of-file flag.

SYNTAX IF [#] END THEN statement [:ELSE statement]

IF END THEN [statement] [ELSE statement]
line-number

COMMENTS: statement = Any executable statement

line-number = Any legal program line-number.

- = A parameter that reverses the logic of the test such that the statement performs IF NOT END

The end-of-file flag is set when an attempt is made to read a data trailer record by a DATA LOAD DC or DATA LOAD DA statement.

If the flag is set (on) and IF END THEN is encountered, its statement is executed. If, instead, a line-number is included, a branch is taken to the specified line-number.

If the flag is reset (off), the optional ELSE statement is executed if present. If ELSE is omitted, no action takes place and execution continues with the following statement.

After evaluation of IF END THEN / ELSE, the end-of-file flag is turned off. The flag may also be turned off by any subsequent DATA LOAD DC or DATA LOAD DA statement, and CLEAR and RUN commands.

EXAMPLES:
```
10 IF END THEN A=5
20 IF-END THEN 200
30 IF END THEN X=Y: ELSE X=Z
```

REFER TO: DATASAVE DC END

IMAGE

REFER TO: %
CONVERT
PACK
UNPACK

INIT

PURPOSE: Initialize all bytes in one or more string-variables to a single value.

SYNTAX INIT (init-val) alpha-variable [,alpha-variable] ...
alpha-array-designator [,alpha-array-designator]

COMMENTS: init-val = A pair of hexadecimal digits, a literal-string, or an alpha-variable, the first byte of which is used in the initialization.

INIT is a carry-over from Wang BASIC, and allthough supported, is not part of the BASIC-2 language, where it has been replaced by the ALL function.

EXAMPLES:
```
10 INIT(00) X$
20 INIT(HEX(DE))A$(1),A$(2)
30 INIT("0")B$()
40 INIT(I$) C$(),D$,STR(E$,,5)
```

REFER TO: ALL

INPUT

PURPOSE: Request data from the terminal during program execution.

SYNTAX INPUT [literal-string [,]] variable [, variable]...

COMMENTS: literal-string = An optional prompt identifying requested data.

variable = One or more variables which will receive the keyed in data.

Mutiple data items requested must be separated by a comma. If a comma must be entered as part of the data, the entire data item must be enclosed in quotes.

A question mark is automatically output (following the prompt if included).

If the operator fails to enter data for all items, the system will reprompt the operator with another question mark until all items in the variable list have been satisfied. If a RETURN is keyed following a prompt with no other information entered, the INPUT statement is terminated. The contents of any variables in the variable list remain unchanged.

Data entered must be of the same type (numeric or alphanumeric) as the corresponding variable.

EXAMPLES:
```
10 INPUT X
20 INPUT "Your Name, Age, and Date of Birth", N$,A,D$
```

REFER TO: LINPUT
KEYIN
MAT INPUT

INPUT SCREEN

INPUT SCREEN

PURPOSE: Retrieve the contents of the terminal screen under program control.

SYNTAX INPUT SCREEN alpha-variable

COMMENTS: A 2236 MXE terminal controller and either a 2336 DE or 2336 DW terminal are required for execution of this statement. Otherwise a non-recoverable error results.

INPUT SCREEN causes the terminal to send 4,080 bytes of information to the CPU. If the alpha-variable is filled prior to completion of the transmission, the remaining bytes are lost.

The first 78 bytes contain the terminal identifaction message.

Bytes 79 and 80 contain the row and column positions of the cursor in binary.

The next 2,000 bytes (representing a 25 x 80 CRT) contain the

The next 2,000 bytes (representing a 25 x 80 CRT) contain the characters currently displayed. Row 25 contains all nulls (HEX(00)).

The remaining 2,000 bytes contain the display attributes for each of the display bytes in the following format:

Bit	Meaning if Set (Binary 1)
80	Character graphics (alternate character set)
40	Reverse video
20	Blink
10	Bright
08	Underlined
04	Contains a left-horizontal box segment
02	Contains a right-horizontal box segment
01	Contains a verticle box segment

Row 25 of the display attributes contains only box information occurring below the 24th row.

EXAMPLE: 10 DIM X$(4080)1: INPUT SCREEN X$()

INT

PURPOSE: Compute the greatest integer of a numeric expression.

SYNTAX ... INT(expression) [...]

COMMENTS: INT returns the largest whole number not greater than the expression.

EXAMPLE: 10 X = INT(Y)

REFER TO: FIX

INV

PURPOSE: A keyword used in matrix math to denote the inverse function.

REFER TO: MAT INV

K

PURPOSE: A parameter used with the SPACE function.

REFER TO: SPACEK

KEYIN

PURPOSE: Obtain a character from an input device.

SYNTAX (Type 1)
KEYIN [file-number,] alpha-variable [,,line-number-2]
[device-address,]

(Type 2)
KEYIN [file-number,] alpha-variable,line-number-1,line-number-2

KEYIN file-number, [alpha-variable,line-number-1,line-number-2]
[device-address,]

COMMENTS: file-number = An indirect device address of the form #n where n corresponds to a slot in the device table (0-15).

device-address = An explicit device address of the form /taa where t is the device type, and aa is the device controller address.

alpha-variable = The receiving variable for the input character.

line-number-1 = The line-number to which a branch will be made if a received character is a standard character.

line-number-2 = The line-number to which a branch will be made if the received character is a special character (ENDI bit set).

The device-address and file-number determine which input device supplies data to the KEYIN statement. When omitted, the current INPUT device is assumed.

Form 1 halts execution until a character is received.

Form 2 tests the given input device once. If the character is available, it is stored in the alpha-variable. Otherwise execution proceeds with the next statement.

EXAMPLES:
```
10 KEYIN /003, X$,,100
20 KEYIN #1,X$
30 KEYIN X$
40 KEYIN X$,200,300
50 KEYIN #2,Y$,400,500
```

REFER TO: INPUT
LINPUT
$GIO

LEN

PURPOSE: Determine the length, in bytes, of an alphanumeric-variable.

SYNTAX ... LEN (alpha-variable) [...]

COMMENTS: LEN is allowed where numeric functions are allowed.

Non-printable characters and blanks preceding the trailing spaces in an alpha-varable are included in the count, but trailing spaces are not counted.

A variable containing all blanks is assumed to have a length of 1.

The entire defined length of a variable may be obtained with the string function (L = LEN(STR(A$))).

EXAMPLES:
```
10 L = LEN(A$)
20 $GIO (A000 400D) B$ <,LEN(B$)>
30 IF LEN(C$) < 6 THEN C$=C$&ALL("*")
```

LET

PURPOSE: Assign the value of an expression to a variable.

SYNTAX [LET] numeric-variable [,numeric-variable] ... = numeric-expression

[LET] alpha-variable [,alpha-variable] ... = alpha expression

COMMENTS: The expression and variable must be of the same type (numeric or alphanumeric).

EXAMPLES:
```
10 LET A=0
20 LET X=X+1
10 R=SQR(X↑2+Y↑2)
```

LGT

PURPOSE: Compute the common logarithm (log base 10) of a numeric-expression.

SYNTAX ... LGT(expression) [...]

COMMENTS: The value of the expression must be positive.

EXAMPLE:
```
10 X = LGT(Y)
```

LIMITS

PURPOSE: Retrieve file bounds and status from the disk or device-table.

SYNTAX Form 1:
```
LIMITS pd [file#,] file-name, start, end, used [,status]
```

Form 2:
```
LIMITS pd [file#,] start, end, current
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

file-name = A literal-string or alpha-variable specifying the name of a disk-cataloged file.

start = A numeric-variable which receives the starting sector address of the file.

end = A numeric-variable which receives the last sector address of the file.

used = A numeric-variable which receives the number of sectors used in the file (as determined by the value stored in the file-trailer).

status = A numeric-variable which receives the status of the file:

-2 = file-name is a scratched data file
-1 = file-name is a scratched program file
 0 = file-name does not exist (start, end, and used are set to zero)
 1 = file-name is an active program file
 2 = file-name is an active data file.

current = A numeric-variable which receives the value of the current-sector-pointer in the device-table.

If the status variable is omitted in Form 1, and the file-name does not exist, a D82 error results.

Form 2 does not access the disk. Information is read directly from the device-table.

EXAMPLES:
```
10 LIMITS F "FILENAME",S,E,U
20 LIMITS T #X, F$,S,E,U,X
30 LIMITS T #4,S,E,C
```

LINE

PURPOSE: A parameter used by SELECT and RESTORE.

REFER TO: SELECT
RESTORE

LINPUT

PURPOSE: Enter and edit a line of alphanumeric data from the keyboard directly into an alpha-variable.

SYNTAX LINPUT [literal [,]] [?] [-] alpha-variable

COMMENTS:

literal = Optional text output to the CO device, usually used to prompt the operator.

alpha-variable = The variable to be modified by LINPUT.

When no optional parameters are present, LINPUT displays the prompts and the entire contents of the alpha-variable, and puts the terminal in edit mode. All edit functions are available and changes are made to the variable as they are made to its display on the CRT.

If the ? parameter is included, LINPUT is executed in text entry mode. In this mode, text defined by DEFFN' statements can be accessed and branches can be taken to marked subroutines via Special Functions Keys. Edit mode can be restored by pressing the EDIT key.

If the - parameter is included, the contents of the alpha-variable displayed are underscored (the variable in memory is not altered). This is useful for displaying the entire defined length of the variable.

EXAMPLES:
```
10 LINPUT "TODAY'S DATE" D$
20 LINPUT "FOOTER TEXT"?-X$()
```

REFER TO: INPUT

PURPOSE: List the program text in memory on the current LIST device.

SYNTAX LIST [S] [title] [D] [line-number-1] [,[line-number-2]]

COMMENTS:

S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

D = Specifies that program text is to be listed in decompressed form.

line-number-1 = If included, specifies the first line of program text to be output. The default is the lowest numbered line.

line-number-2 = If included, specifies the last line of program text to be output. The default is the highest numbered line.

EXAMPLES:
```
LIST
LISTS
LIST 800
LIST 800,
LIST "FINANCE PROGRAM"D 1000,8999
```

REFER TO:
```
LIST #
LIST '
LIST DC
LIST DT
LIST I
LIST T
LIST V
```


PURPOSE: Produce a line-number cross-reference.

SYNTAX LIST [S] [title] # [line-number-1] [,[line-number-2]]

COMMENTS:

S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

= Indicates a line-number cross-reference.

line-number-1 = If included, specifies the first line to be cross-referenced. The default is the lowest numbered line.

line-number-2 = If included, specifies the last line to be cross-referenced. The default is the highest numbered line.

EXAMPLES: LIST # 100
LIST S# ,500
LIST "LINE NUMBER X-REF"#

LIST '

PURPOSE: Produce a marked subroutine cross-reference.

SYNTAX LIST [S] [title] ' [integer]

COMMENTS:

S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

' = Indicates a marked subroutine cross-reference.

integer = Designates a specific DEFFN' subroutine to be cross-referenced. If no integer is specified, all DEFFN' subroutines are cross-referenced.

EXAMPLES: LIST S'
LIST '144
LIST "Marked Subs X-Ref" '

LIST DC

PURPOSE: Display the contents of the disk-catalogue-index.

SYNTAX LIST [S] [title] DC pd [file#,] [<file-type>] [-] [name-spec]
[disk-addr] [(sector)]

COMMENTS:

S = A parameter specifying that the list is to be scrolled in sections consisting of the number of lines specified by SELECT LINE.

title = A literal-string or alpha-variable containing a heading to be highlighted and printed. When included, the system outputs a page-eject prior to the listing.

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

disk-addr = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

file-type = A parameter used to limit the list to files of a specified type:

<D> = Data files
<P> = Program files
<S> = Scratched files
<SD> = Scratched data files
<SP> = Scratched program files

\- = A parameter specifying that file-spec and sector logic are reversed.

file-spec = A literal-string or alpha-variable specifying a mask used to limit the list to file-names meeting (or not meeting) a criteria.

sector = A numeric-expression specifying a sector-address limiting the list to files occurring before (or after) the sector. If omitted, the entire platter is listed.

The "*" and "?" characters have special meaning in file-spec. Normally, any character in file-spec requires an exact match in the corresponding column-position of the disk file-name in order for the file-name to be included in the list. The "?" specifies that any character may occupy the column, and the "*" indicates that an unspecified number of any character may occur before and/or after other characters. File-specs shorter than eight characters are assumed to be padded with ALL("?").

EXAMPLES:
```
10 LIST DCT#4,
20 LIST "All 'GL' program files" DCT "GL"
30 LIST DCT/D15, <P>"*X"
40 LIST S "Scratched Data-files after sector 1000" DC <SD> (1000)
50 LIST "Program names without any X's" DCT /D11, <P> - "*X"
```

LIST DT

PURPOSE: List the contents of the device table.

SYNTAX LIST [S] [title] DT

COMMENTS:

S = A parameter specifying that the list is to be scrolled in sections consisting of the number of lines specified by SELECT LINE.

title = A literal-string or alpha-variable containing a heading to be highlighted and printed on a hard-copy device. When included, and the LIST device is not the CRT, the system outputs a page-eject prior to the listing.

If the CRT is the LIST device, the screen is cleared prior to display of the device table and box graphics are included for readability.

EXAMPLES:
```
LIST DT
10 LIST "DEVICE TABLE" DT
```

LIST I

PURPOSE: List the contents of the interrupt table.

SYNTAX LIST [S] [title] I

COMMENTS:

S = A parameter specifying that the list is to be scrolled in sections consisting of the number of lines specified by SELECT LINE.

title = A literal-string or alpha-variable containing a heading to be highlighted and printed. When included, the system outputs a page-eject prior to the listing.

REFER TO: SELECT
Chapter 8 of the Wang BASIC-2 Language Reference Manual

LIST T

PURPOSE: Produce a program text cross-reference.

SYNTAX LIST [S] [title] T alpha-variable [,alpha-variable] ...
literal-string [,literal-string]

COMMENTS:

S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

T = Indicates a program text cross-reference.

The character string(s) to be cross-referenced can be specified by a literal-string or alpha-variable. Spaces are ignored.

EXAMPLES: LIST T "ABC"
LIST ST A$(1),A$(2),"PRINT"
LIST "GIOs in Program",T "$GIO"

LIST V

PURPOSE: Produce a variable cross-reference list.

SYNTAX LIST [S] [title] [+] V [variable-name] [,[variable-name]]
[-]

COMMENTS:

S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

+ = A parameter indicating that the type, dimensions, and current stored values are to be listed for each

variable, rather than a line number cross reference.

- = A parameter that is currently equivalent to +.

V = Indicates a variable cross-reference.

variable-name = letter [digit] for numeric-scalars
letter [digit]$ for alpha-scalars
letter [digit](for numeric-arrays
letter [digit]$(for alpha-arrays

Both variables in the command must be the same type.

EXAMPLES:
```
LISTV
LIST SV A$(,A9$(
LIST "VARIABLE X-REF"V
LIST "ALL NUMERIC SCALERS"V A,
LIST +V
LIST "VALUES OF ALL 'B' ALPHA ARRAYS' +V B$(,B9$(
```

LOAD

PURPOSE: A keyword used in program loading statements.

REFER TO:
```
LOAD DC
LOAD DA
LOAD RUN
```

LOAD DA

PURPOSE: Load a program file from disk in "direct address" mode.

SYNTAX
Command:
```
LOAD DA pd [file#,  ] (addr [,var])
           [address,]
```

Statement:
```
LOAD DA pd [file#,  ] (addr, [,var]) [start] [,end] [BEG line-nbr]
           [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

addr = A variable or numeric-expression which specifies the address of the first sector of a program file. If an alpha-variable, the address is assumed to be a two-byte binary value.

var = A variable (must be common in statements) which receives the address of the sector following the last read by LOAD DA. If alphanumeric, the address is returned as a two-byte binary value.

start = The first line-number of program text to be cleared from memory during LOAD DA. If omitted, the lowest

line-number is assumed.

end = The last line-number of program text to be cleared from memory during LOAD DA. If omitted, the highest line-number is assumed.

BEG = A parameter specifying that program execution is to begin at the specified line-number following the load. If the line-number does not exist, the lowest line-number greater than BEG is used. If omitted, the BEG line-number is assumed to be start.

In addition to the execution options, the main difference between the statement and command, is that the statement clears program text, and the command does not. Programs loaded via the command are automatically threaded into existing text.

EXAMPLES:
```
   LOAD DA T (X)
   LOAD DA T /D12, (X,Y)
10 LOAD DA F (VAL(Q0$,2))
20 LOAD DA T#1, (Q$,Q$) ,4000
30 LOAD DA T/D21, (40) 1000,9000 BEG 100
40 LOAD DA T (C(1),X$) 1000 BEG 100
```

REFER TO: SAVE DA

LOAD DC

PURPOSE: Load a program file from disk in disk catalogue mode.

SYNTAX
```
Command:
LOAD [DC] pd [file#,  ] file-name
             [address,]

Statement:
LOAD [DC] pd [file#,  ] file-name [start] [,end] [BEG line-nbr]
             [address,] <exp> var
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

file-name = A cataloqued program file-name

start = The first line-number of program text to be cleared from memory during the LOAD. If omitted, the lowest line-number is assumed.

end = The last line-number of program text to be cleared from memory during the LOAD. If omitted, the highest line-number is assumed.

BEG = A parameter specifying that program execution is to begin at the specified line-number following the load. If the line-number does not exist, the lowest line-number greater than BEG is used. If omitted,

exp = A numeric-expression specifying the number of files to be simultaneously loaded from disk.

var = A common alpha-variable containing the names of the files to be loaded in 8-byte increments. Element boundaries are ignored.

In addition to the execution options, the main difference between the statement and command, is that the statement clears program text, and the command does not. Programs loaded via the command are automatically threaded into existing text.

EXAMPLES:
```
   LOAD T "PROGRAM"
   LOAD DC T /D12, "PROG1"
10 LOAD F "PROGRAM"
20 LOAD T#1, "PROG2" 20
30 LOAD DC T/D21, "PROG3" ,9000 BEG 100
40 LOAD R <Q> Q$() 1000, 3000 BEG 100
```

REFER TO: SAVE DC

LOAD RUN

PURPOSE: Initialize a partition; load and run a disk-catalogued program.

SYNTAX
```
LOAD RUN [pd] [file#,  ] [file-name]
             [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T). If omitted, T is assumed.

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

file-name = A catalogued program file-name. If omitted, the file-name "START" is assumed.

When executed either as a command or statement, LOAD RUN performs the following:

Clears all program text;
Clears all variables;
Assigns the CO device to PRINT and LIST operations;
Assigns the CI device to INPUT operations;
Clears file parameters in device-table-slot #0;
Clears device-table-slots #1 through #15;
Turns off Pause and TRACE;
SELECTs ERROR >60 and radians mode;
Clears all stacks and the interrupt table;
Clears the ERR function; and
LOADs and RUNs the requested program.

A check is first made for the existence of file-name. If it does not exist, a D82 error is signaled and none of the above operations is performed.

EXAMPLES: LOAD RUN

```
      LOAD RUN F#1,"PROGRAM"
10 LOAD RUN "STARTUP"
20 LOAD RUN T/D10,
```

REFER TO: LOAD DC
SAVE DC

LOG

PURPOSE: Compute the natural logarithm (log base e) of a numeric-expression.

SYNTAX ... LOG(expression) [...]

COMMENTS: The value of the expression must be positive. e = 2.718281828459 = EXP (1).

EXAMPLE:
```
10 X = LOG(Y)
```

REFER TO: EXP

LS

PURPOSE: A parameter specifying the number of disk sectors to reserve for the disk catalog index (library sectors) in the SCRATCH DISK statement.

REFER TO: SCRATCH DISK

MAT

PURPOSE: A key word used to denote a matrix math or sort operation.

REFER TO: MAT ()*
MAT *
MAT +
MAT -
MAT =
MAT CON
MAT CONVERT
MAT COPY
MAT IDN
MAT INPUT
MAT INV
MAT MERGE
MAT MOVE
MAT PRINT
MAT READ
MAT REDIM
MAT SEARCH
MAT SORT
MAT TRN
MAT ZER

MAT ()*

PURPOSE: Multiply the elements in a matrix by the value of an expression.

SYNTAX MAT c = (k) * a

COMMENTS: a,c = Numeric-array-names.

k = A numeric-expression.

The result of the equation is stored in c, and c can appear on both sides of the equation.

If not previously dimensioned, MAT ()* will implicitly declare a 10x10 matrix. Matrix c is automatically redimensioned to that of matrix a.

EXAMPLE:
```
10 MAT A = (2)*A
20 MAT B = (LGT(X+Y))*A
```

MAT *

PURPOSE: Multiply two matrices.

SYNTAX MAT c = a * b

COMMENTS: a,b,c = Numeric-array-names

The number of columns in matrix a must equal the number of rows in matrix b. Matrix c is redimensioned to the number of rows in a and the number of columns in b.

If not previously dimensioned, MAT INPUT will implicitly declare a 10x10 matrix.

EXAMPLE:
```
10 MAT C = A * B
```

MAT +

PURPOSE: Add two matrices.

SYNTAX MAT c = a + b

COMMENTS: a,b,c = Numeric array names.

The matrices must have the same dimensions.

The matrix c receives the sum of each element and can appear on both sides of the equation.

If not previously dimensioned, MAT + will implicitly declare a 10x10 matrix.

EXAMPLES:
```
10 MAT C = A + B
20 MAT C = C + A
```

MAT –

PURPOSE: Subtract a matrix from another matrix

SYNTAX MAT c = a - b

COMMENTS: a,b,c = Numeric-array-names.

Matrix c receives the result of matrix a - matrix b, and can appear on eaither side of the equation.

The dimensions of matrix a must be the same as the dimensions of matrix b.

Matrix c is automatically redimensioned to the dimensions of matrices a and b.

If not previously dimensioned, MAT - will implicitly declare a 10x10 matrix.

EXAMPLE: 10 MAT A = B - C

MAT =

PURPOSE: Automatically redimension and replace each element in a matrix with its corresponding element in another matrix.

SYNTAX MAT c = a

COMMENTS: c,a = Numeric-array-names.

Each element in array a replaces a corresponding element in array c. Matrix c is automatically redimensioned to match a's dimensions. Both a and c must have been declared with the same number of dimensions.

If not previously dimensioned, MAT - will implicitly declare a 10x10 matrix.

EXAMPLE: 10 MAT X=Y

MAT CON

PURPOSE: Set each element in a matrix equal to one and optionally redimension the matrix.

SYNTAX MAT c = CON [(dim1[,dim2])]

COMMENTS: c = A numeric array name.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that, for one-dimensional-arrays, 1 <= dim1 <= 65535, and for two-dimensional arrays, 1 <= dim1 <= 255. If c is a vector, dim1 cannot exceed the number of elements originally dimensioned.

dim2 = A numeric-expression, the integer of which specifies the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

The number of dimensions cannot be changed.

If not previously dimensioned, MAT CON will implicitly declare a 10x10 matrix.

EXAMPLES:
```
10 MAT A = CON
20 MAT B = CON(100)
30 MAT C = CON(10,20)
40 MAT D = CON(2*X,3*Y)
```

MAT CONVERT

PURPOSE: Convert numeric-array data in internal format to alphanumerics in sort format for future use by MAT SORT or MAT MERGE.

SYNTAX MAT CONVERT numeric-array TO alpha-array [(s,n)]

COMMENTS:

s = A numeric-expression specifying the start byte to be used in each element of the alpha-array.

n = A numeric-expression specifying the number of bytes to use in each element of the alpha-array.

Each element in the numeric-array is converted and stored in the corresponding element in the alpha-array. The alpha-array must have at least as many elements as the numeric-array.

MAT CONVERT is a supported carry-over from Wang BASIC. Its function has been replaced by the BASIC-2 version of MAT MOVE.

EXAMPLES:
```
10 MAT CONVERT A() TO A$()
20 MAT CONVERT X() TO Y$() (5,4)
```

REFER TO: MAT MOVE

MAT COPY

PURPOSE: To provide a flexible means of transferring string data between variables.

SYNTAX MAT COPY [-] source-var [<[x][,y]>] TO [-] output-var [<[x][,y]>]

COMMENTS:

source-var = Any alphanumeric variable or string function.

output-var = Any alphanumeric variable or string function.

<x,y> = Specifies a segment of the corresponding variable to be used by MAT COPY.

x = A numeric expression specifying the starting byte.

y = A numeric expression specifying the number of bytes.

Both the source-var and the output-var are treated as contiguous

strings of bytes. Data is transferred from the source-var to the output-var one byte at a time until the output-var is filled. If the source-var does not supply enough data to fill the output-var, the output-var is padded with spaces.

A "-" before either variable indicates data is sent to or accepted into the variable in reverse order.

The STR(function may be used alternatively or in conjunction with <x,y> to specify a segment of a variable involved in a MAT COPY.

EXAMPLES:
```
10 MAT COPY A$ TO -A$()
20 MAT COPY B$() <X> TO B$() <,Y>
30 MAT COPY -STR(X$(),,10) TO -STR(X$(X),5)
```

MAT IDN

PURPOSE: Create and optionally redimension an identity matrix.

SYNTAX `MAT c = IDN [(dim1,dim2)]`

COMMENTS:

c = A numeric array name.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that 1 <= dim1 <= 255.

dim2 = A numeric-expression, the integer of which specifies the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

The matrix must be square. If redimensioned, dim1 must equal dim2.

If not previously dimensioned, MAT IDN will implicitly declare a 10x10 matrix.

EXAMPLES:
```
10 MAT A = IDN
20 MAT B = IDN(5,5)
30 MAT C = IDN(2*X,2*X)
```

MAT INPUT

PURPOSE: INPUT matrix values from the keyboard with optional redimensioning.

SYNTAX `MAT INPUT array-name [(dim1[,dim2]) [length]] [, ... ]`

COMMENTS:

array-name = The matrix name.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that, for one-dimensional-arrays, 1 <= dim1 <= 65535, and for two-dimensional arrays, 1 <= dim1 <= 255. If array-name is a vector, dim1 cannot exceed the number of elements originally dimensioned.

dim2 = A numeric-expression, the integer of which specifies

the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

length = The new element length for an alpha-array. The product dim1 * dim2 * length cannot exceed the number of bytes originally dimensioned.

MAT INPUT functions identically to INPUT. Data is stored row by row.

The number of dimensions cannot be changed.

If not previously dimensioned, MAT INPUT will implicitly declare a 10x10 matrix. Alpha elements will have length 16.

EXAMPLE:
```
10 MAT INPUT A
20 MAT INPUT A(2,4),B
30 MAT INPUT B$
40 MAT INPUT B$(5)12,C
```

MAT INV

PURPOSE: Calculate the inverse of a matrix..

SYNTAX MAT c = INV(a) [,[d][,n]]

COMMENTS:

c = The numeric-array-name of the matrix to receive the inverse of a. Matrix c is automatically redimensioned to match those of a.

a = The numeric-array-name of the matrix to be inverted.

d = A numeric variable set equal to the value of the determinate of a by MAT INV.

n = A numeric variable set equal to the value of the normalized determinate of a by MAT INV.

Matrix a must be square. Since inversion is done in place, both matrices c and a may be the same.

If not previously dimensioned, MAT INPUT will implicitly declare a 10x10 matrix.

EXAMPLES:
```
10 MAT A = INV(B)
20 MAT C0 = INV(C1),D
30 MAT X = INV(X),D,N(1)
```

MAT MERGE

PURPOSE: Merge two or more sorted input files into a sorted output file.

SYNTAX MAT MERGE merge-array[(x[,y])] TO control-var, work-var, loc-array

COMMENTS:

merge-array = A two-dimensional alphanumeric-array, each row of which contains sorted information from a data file to be merged.

x = A numeric expression whose integer value specifies

the start byte of the merge field within each array element.

y = A numeric expression whose integer value specifies the length of the merge field within each array element. If omitted, the remainder of the element is assumed.

control-var = An alpha-variable of length >= r+1, where r is the number of rows in the merge-array, used to store status information for the merge.

work-var = An alpha-variable of length >= 2*r, where r is the number of rows in the merge-array, used as a scratch area by MAT MERGE.

loc-array = An alpha-array containing at least c elements of length 2, where c is the number of columns in the merge-array. The loc-array stores subscripts output from the merge. Generally, merge efficiency increases with the number of loc-array elements.

See the BASIC-2 manual for a complete description of MAT MERGE.

EXAMPLE:
```
10 DIM M$(3,400)10,C$(4)1,W$(3)2,L$(1200)2
50 MAT MERGE M$() (2) TO C$(), W$(), L$()
```

REFER TO: MAT MOVE

MAT MOVE

PURPOSE: Move data from one array to another, element-by-element and optionally convert numeric data in Wang internal format to alphanumeric data in sort format, and vice versa.

SYNTAX MAT MOVE move-array [,locator-array][,n] TO receiver-array

COMMENTS:

move-array = alpha-array-designator [(x[,y])]
numeric-array-designator

locator-array = alpha-array-designator
alpha-array-element

receiver-array = alpha-array-element [(x[,y])]
numeric-array-element
alpha-array-designator [(x[,y])]
numeric-array-designator

n = A numeric-scaler-variable used as a counter, specifying the maximum number of elements to move. MAT MOVE returns the number of elements actually moved into this variable when the transfer is complete.

(x,y) = Designates a field within each alpha-array element.

x = A numeric-expression specifying the start byte of each field.

y = A numeric-expression specifying the number of bytes in the field. If not specified, the entire

remaining field length is assumed.

MAT MOVE comes in two forms, the simple and complex. The complex form is used between arrays of different types to automatically convert numeric data for sorting purposes.

If a locator-array is not specified, move-array data is transferred starting with the first element of the move-array, to the first element of the receiver-array, or, if a receiver-array-element is specified, starting at the specified receiver-array-element.

The locator-array is usually generated by MAT SORT or MAT MERGE. It is used to indirectly specify the subscript order that move-array elements are transferred to the receiver-array. Transfers will be made using the first subscript in the locator-array unless a specific starting element is specified by a locator-array-element. Locator array elements must be of length 2. The number of elements required is based on the requirements of the SORT or MERGE function.

EXAMPLES:
```
10 MAT MOVE A() TO B()
20 MAT MOVE A() TO B$(5,8)
30 MAT MOVE X$(Y+Z,Z),L$(X(X)),N, TO Z(X,Y)
```

REFER TO: MAT CONVERT

MAT PRINT

PURPOSE: Output matrix values to the PRINT device.

SYNTAX MAT PRINT array-name [separator array-name] ... [separator]

COMMENTS:

array-name = The name of the matrix to be printed.

separator = A comma or semicolon. A comma indicates that coloumn elements are to be printed in zoned format. A trailing comma is assumed.

The matrices are printed row by row. A one-dimensional array is treated as column-vector.

If not previously dimensioned, MAT PRINT will implicitly declare a 10x10 matrix. Alpha-array elements will be length 16.

EXAMPLE:
```
10 MAT PRINT A
20 MAT PRINT B;C$;
30 MAT PRINT D,E;
```

MAT READ

PURPOSE: Assign DATA statement values to and optionally redimension matrices.

SYNTAX MAT READ array-name [(dim1[,dim2]) [length]] [,' ...]

COMMENTS:

array-name = The name of the matrix.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that, for one-dimensional-arrays, 1 <= dim1 <= 65535, and for two-dimensional arrays, 1 <= dim1 <= 255. If

array-name is a vector, dim1 cannot exceed the number of elements originally dimensioned.

dim2 = A numeric-expression, the integer of which specifies the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

length = The new element length for an alpha-array. The product dim1 * dim2 * length cannot exceed the number of bytes originally dimensioned.

The number of dimensions cannot be changed.

If not previously dimensioned, MAT READ will implicitly declare a 10x10 matrix. Alpha-array elements will be length 16.

The matrix type must agree with the DATA type.

EXAMPLE:
```
10 MAT READ A(2,4)
20 MAT READ A$(B,C)D
30 MAT READ A,B,C$
```

REFER TO: RESTORE

MAT REDIM

PURPOSE: Redimension a matrix.

SYNTAX MAT REDIM array-name (dim1[,dim2]) [length] [, ...]

COMMENTS:

array-name = The name of the matrix.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that, for one-dimensional-arrays, 1 <= dim1 <= 65535, and for two-dimensional arrays, 1 <= dim1 <= 255. If array-name is a vector, dim1 cannot exceed the number of elements originally dimensioned.

dim2 = A numeric-expression, the integer of which specifies the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

length = The new element length for an alpha-array. The product dim1 * dim2 * length cannot exceed the number of bytes originally dimensioned.

The number of dimensions cannot be changed.

If not previously dimensioned, MAT REDIM will implicitly declare a 10x10 matrix. Alpha-array elements will be length 16.

EXAMPLE:
```
10 MAT REDIM B$(10)12,C$(2*X,4)
```

REFER TO: DIM
COM

PURPOSE: Search a variable or literal-string for substrings that meet a given condition.

SYNTAX MAT SEARCH var-1 [<[x][,y]>], relation var-2 TO pointer [STEP s]
lit-1 lit-2

COMMENTS:

var-1,var-2 = Any alphanumeric variable or string function.

lit-1,lit-2 = An alphanumeric literal enclosed in quotes or a HEX(literal.

<x,y> = Used to specify a segment of var-1 to be searched. It may not be used to search a portion of a literal.

x = A numeric expression specifying the the starting byte of var-1 to be searched.

y = A numeric expression specifying the number of bytes in var-1 to be searched.

relation = Any relational operator (=, <>, <, <=, >, <=).

pointer = An alphanumeric variable at least 2 bytes in length and preferably containing an even number of bytes.

s = A numeric expression such that 1 <= s <= 65535

MAT SEARCH scans var-1 or lit-1 looking for substrings matching the relation applied to var-2 or lit-2. The starting byte positions of substrings in var-1 and lit-1 meeting the condition are stored in pointer as a series of two-byte binary values.

If room in pointer allows, a value of HEX(0000) is stored as the final position indicating the end of search results. Remaining bytes are left unchanged. A HEX(0000) in the first two bytes indicates that no matches were found.

The length of substrings in var-1 and lit-1 compared is determined by the defined length or substring length of var-2 and lit-2. The search terminates when either avaiable space in pointer has been completely exhausted, or the remaining length in var-1 or lit-1 is less than the length of var-2 or lit-2.

The first search is performed starting at the first byte of var-1 or lit-1. Each additional search is performed starting at the next successive byte. The normal increment of one may be overridden by specifying a STEP increment.

EXAMPLES:
```
10 MAT SEARCH "D10D11D12", =D$ TO P$ STEP 3
20 MAT SEARCH A$() <,20>, <= HEX(0100) TO Q$()
```

MAT SORT

PURPOSE: Create a sort locator-array.

SYNTAX MAT SORT sort-array [(x[,y])] TO work-variable, locator-array

COMMENTS:

sort-array = An alpha-array-designator containing fields to be

x = A numeric expression whose integer value specifies the start byte of the sort field within each array element.

y = A numeric expression whose integer value specifies the length of the sort field within each array element. If omitted, the remainder of the element is assumed.

work-variable = An alpha-variable containing at least 2*e bytes, where e is the number of elements in sort-array, used as a scratch pad by MAT SORT.

locator-array = An alpha-array containing at least e elements of length 2, where e is the number of elements in sort-array, used to store the sorted sort-array element subscripts output by MAT SORT.

See the BASIC-2 manual for a complete description of MAT SORT.

EXAMPLE:
```
10 DIM S$(500)10,W$(500)2,L$(500)2
50 MAT SORT S$() (3,4) TO W$(), L$()
```

REFER TO: MAT MOVE

MAT TRN

PURPOSE: Replace a matrix with the transposition of another matrix.

SYNTAX MAT c = TRN(a)

COMMENTS:

a,c = Numeric-array-names.

Matrix c is replaced with the transposition of matrix a, and is redimensioned to the same dimensions as the transposition of matrix a.

If not previously dimensioned, MAT TRN will implicitly declare a 10x10 matrix.

EXAMPLE: MAT A = TRN(B)

MAT ZER

PURPOSE: Optionally redimension and zero a matrix.

SYNTAX MAT c = ZER [(dim1[,dim2]])

COMMENTS:

c = A numeric-array-name.

dim1 = A numeric-expression, the integer of which specifies the new number of rows in the matrix, such that, for one-dimensional-arrays, 1 <= dim1 <= 65535, and for two-dimensional arrays, 1 <= dim1 <= 255. If array-name is a vector, dim1 cannot exceed the number of elements originally dimensioned.

dim2 = A numeric-expression, the integer of which specifies the new number of columns in the matrix. The product dim1 * dim2 cannot exceed the number of elements originally dimensioned.

The number of dimensions cannot be changed.

If not previously dimensioned, MAT ZER will implicitly declare a 10x10 matrix.

EXAMPLE:
```
10 MAT A = ZER
20 MAT B = ZER(100)
40 MAT C = ZER(2*X,Y)
```

MAX

PURPOSE: Find the maximum value of a series of expressions and numeric-arrays.

SYNTAX ... MAX(arg [,arg] ...) [...]

COMMENTS: arg = A numeric-expression or numeric-array.

MAX scans the value of the expressions and each element of the array and produces the maximum value located.

EXAMPLE:
```
10 X = MAX(X*Y,X(),Y(),Z())
```

REFER TO: MIN

MERGE

PURPOSE: A keyword used in data merging operations.

REFER TO: MAT MERGE

MIN

PURPOSE: Find the minimum value of a series of expressions and numeric-arrays.

SYNTAX ... MIN(arg [,arg] ...) [...]

COMMENTS: arg = A numeric-expression or numeric-array.

MAX scans the value of the expressions and each element of the array and produces the minimum value located.

EXAMPLE:
```
10 X = MIN(X*Y,X(),Y(),Z())
```

REFER TO: MAX

MOD

PURPOSE: Compute the remainder of the division of two numeric expressions.

SYNTAX ... MOD(expression-1,expression-2) [...]

COMMENTS: MOD is the modulo function and returns a value based expression-1 divided by expression-2. If X1 is expression-1 and X2 is expression-2, MOD is the equivalent of:

```
X1 - INT(X1/X2)*X2
```

EXAMPLE:

```
10 X = MOD(Y,Z)
```

MOVE

PURPOSE: Copy one or all active disk files to another platter; a keyword used in other statements.

SYNTAX Form 1:

```
MOVE pd [file#,  ] TO pd [file#,  ]
        [address,]       [address,]
```

Form 2:

```
MOVE pd [file#,  ] source-name TO pd [file#,  ] [ ([destination]) ]
        [address,]                   [address,] [ ( space       ) ]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

source-name = A literal-string or alpha-variable specifying the active cataloqued file to be MOVEd.

destination = A literal-string or alpha-variable specifying the scratched file-name to be overwritten and renamed to source-name. If omitted, the source-name is assumed.

space = A numeric-expression specifying the number of additional sectors to be allocated to the new file. If omitted, the size of source-name is used.

Form 1 creates a new disk index of the same type (SCRATCH or SCRATCH') and size on the destination platter and transfers all active files in disk cataloque sequence. The source and destination platters must not be the same.

Form 2 MOVEs a single file. If destination is not specified, a new file is created with optional extra space. A pair of parentheses without destination or source specified assumes that a scratched file having the same name as source-name is to be overwritten.

EXAMPLES:

```
10 MOVE F TO R
20 MOVE T#1, TO T/D12,
30 MOVE T "PROGRAM" TO R(S+10)
40 MOVE F "DATA" TO R()
50 MOVE T/D11, X$ TO T/D13,(Y$)
```

REFER TO: MOVE-END

REFER TO: MOVE-END
MAT MOVE

MOVE END

PURPOSE: Change the size of the disk catalogue area.

SYNTAX MOVE END pd [file#,] = expression
[address,]

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

expression = A numeric-expression giving the new sector address of the end of the catalogue area such that sectors-used <= expression <= highest-disk-sector.

MOVE END does not alter the size of the disk catalogue index.

EXAMPLES:
```
10 MOVE END R = 19583
20 MOVE END T/D11, = MIN(X,52607)
```

REFER TO: SCRATCH DISK

MSG

REFER TO: $MSG

NEXT

PURPOSE: Increment a loop counter and check for loop termination.

SYNTAX NEXT counter [,counter] ...

COMMENTS:

counter = A numeric-scaler-variable corresponding to the counter specified in a previously executed FOR/TO loop definition statement.

The STEP value specified in the corresponding FOR/TO statement is added to the counter by the NEXT statement. The sum is checked against the exit value specified by FOR/TO. More than one loop may be checked for termination conditions in a single NEXT statement by specifying multiple counters in the order in which they must be evaluated.

A loop is terminated and its loop control data cleared from the stacks under one of the following three conditions:

1) The STEP value is positive and the counter is greater than the exit value;
2) The STEP value is negative and the counter is less than the exit

value; or
3) The STEP value is zero.

If a loop terminates, the next successive counter and FOR/TO combination in NEXT is evaluated. If no other counters remain, execution proceeds to the statement following NEXT.

If, after incrementing and evaluating a counter, a loop does not terminate, execution proceeds to the statement following the corresponding FOR/TO statement.

EXAMPLES: 10 NEXT X
20 NEXT K,J,I

REFER TO: FOR

NO

PURPOSE: A parameter used with the SELECT ROUND statement.

REFER TO: SELECT

NUM

PURPOSE: A numeric-function that returns the number of characters in an alphanumeric variable which represent a legal BASIC number.

SYNTAX ... NUM(alpha-variable) [...]

COMMENTS: The alpha-variable is scanned starting with the first byte.

The NUM function can return a value of zero if the first byte is not part of a legal BASIC number.

If the alpha-variable contains a legal number, the entire defined length of the variable is returned, including trailing spaces.

If only the initial portion of the variable represents a legal BASIC number and is followed by non-numeric-format characters, the NUM funtion retuns only the number of characters that can be successfully converted with a CONVERT statement.

EXAMPLES: 10 IF NUM(N$)=LEN(STR(N$)) THEN CONVERT N$ TO N
110 X = NUM(I$): IF X >= 5 THEN 120: ELSE PRINT "RE-ENTER": GOTO 100

OFF

PURPOSE: A parameter

REFER TO: TRACE
TRACE DISK
$IF OFF
$DISCONNECT

ON

PURPOSE: A keyword used in computed branches, error control, device readiness tests, and interrupt enabling.

REFER TO: ON GOTO
ON GOSUB
ON SELECT
ON ERROR
SELECT
$IF ON
$DISCONNECT

ON ERROR

PURPOSE: A general purpose error trapping statement.

SYNTAX ON ERROR variable-1, variable-2 GOTO line-number

COMMENTS:

variable-1 = An alpha-variable at least three bytes in length which receives the ASCII error code.

variable-2 = An alpha-variable at least four bytes in length which receives the line-number where the error occurred.

ON ERROR is a non-executable statement. It instructs the operating system to branch to the specified line-number when any execution error occurs.

To trap program load errors, variable-1 and variable-2 must be common.

ON ERROR over-rides the ERROR statement, and, when a branch is taken to the error line-number, all subroutine and loop stacks are cleared. Consequently, ON ERROR has limited practical applications.

ON ERROR is a carry-over from Wang BASIC and, although supported, is not officially part of the BASIC-2 language.

EXAMPLE: 10 ON ERROR E$,N$ GOTO 100

REFER TO: ERROR

ON GOSUB

PURPOSE: Branch to a subroutine starting at a line-number contained in a list of line-numbers, based on a pointer.

SYNTAX ON exp GOSUB [,[line-number,]] ... line-number [: ELSE statement]
var

COMMENTS:

exp = A numeric-expression whose truncated integer value is used as a pointer.

var = Any alpha-variable or string function, the binary value of the first byte of which is used as a pointer.

line-number = Any legal line-number

statement = Any executable statement (cannot be % (image), DATA, or DEFFN).

ON GOSUB is a computed GOSUB statement. The line-number to which a branch is taken is based on the value of the pointer. Line-numbers are numbered from left to right starting with 1.

Consecutive commas denote a null line-number.

If the pointer is less than 1, greater than the number of line-numbers in the list, or points to a null line-number, a subroutine branch is not taken.

If a subroutine branch is not taken and ELSE is included, its statement is executed. Otherwise execution proceeds to the next statement following ON GOSUB.

EXAMPLE:
```
10 ON A-2 GOSUB 1000,2000,3000: ELSE A=B
20 ON STR(B$(W),X) GOSUB 5500
```

ON GOTO

PURPOSE: Branch to a line-number contained in a list of line-numbers, based on a pointer.

SYNTAX
```
ON exp GOTO [,[line-number,]] ... line-number [: ELSE statement]
   var
```

COMMENTS:

exp = A numeric-expression whose truncated integer value is used as a pointer.

var = Any alpha-variable or string function, the binary value of the first byte of which is used as a pointer.

line-number = Any legal line-number.

statement = Any executable statement (cannot be % (image), DATA, or DEFFN).

ON GOTO is a computed GOTO statement. The line-number to which a branch is taken is based on the value of the pointer. Line-numbers are numbered from left to right starting with 1.

Consecutive commas denote a null line-number.

If the pointer is less than 1, greater than the number of line-numbers in the list, or points to a null line-number, a branch is not taken.

If a branch is not taken and ELSE is included, its statement is executed. Otherwise execution proceeds to the next statement following ON GOTO.

EXAMPLE:
```
10 ON MOD(A,4)+1 GOTO 150,200,250,300
20 ON B$ GOTO 100,220 :ELSE LOAD RUN
```

ON SELECT

PURPOSE: Conditionally select system options and I/O devices.

SYNTAX ON sel-val SELECT [;[sel-list;]] ... [sel-list] [: ELSE statement]

COMMENTS:

sel-val = An expression or alpha-variable.

sel-list = A list of parameters in the form:
select-parameter [,select-parameter] ...
which selects a system option or associates a device address with a class of I/O operations.

statement = Any executable statement (cannot be % (image), DATA, or DEFFN).

ON SELECT is a computed SELECT statement. The sel-list to which a branch is taken is based on the value sel-val. Sel-lists are numbered from left to right starting with 1.

Consecutive semicolons denote a null SELECTion.

If sel-val is less than 1, greater than the number of sel-lists, or points to a null SELECTion, SELECTions are not made.

If a SELECTion is not made and ELSE is included, its statement is executed. Otherwise execution proceeds to the next statement following ON SELECT.

All select-parameters in a sel-list are SELECTed simultaneously.

For more information on select-parameters, refer to SELECT.

EXAMPLES:
```
10 ON X SELECT PRINT 005, LIST 005; PRINT 215; PRINT 216
20 ON P$ SELECT P1;P2;P3;P4;P5;P6;P7;P8;P9: ELSE SELECT P
```

REFER TO: SELECT

OPEN

PURPOSE: A keyword used in I/O statements.

REFER TO: $OPEN
DATALOAD DC OPEN
DATASAVE DC OPEN

OR

PURPOSE: Perform a logical OR function on alpha-operand bits; a logical or operator.

SYNTAX receiver = [...] OR operand [...]

COMMENTS:

receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

If an alpha-variable does not immediately follow the =, the receiver is considered an operand in the logical OR evaluation.

The corresponding bits of each operand are compared and the result is stored in the receiver. If either or both are 1, the result is 1. Otherwise the result is 0.

The OR function should not be confused with the logical operator OR used in IF statements.

EXAMPLE:
```
10 A$ = OR B$
20 C1$,C2$ = STR(D$,5,4) OR ALL(0F)
```

REFER TO: IF

P

PURPOSE: A parameter.

REFER TO: SELECT
SAVE DA
SAVE DC

PACK

PURPOSE: Pack a list of numeric variables into an alpha-variable.

SYNTAX PACK (image) alpha-variable FROM pack-element [,pack-element] ...

COMMENTS:

image = A string of ASCII characters representing the image, or an alpha-variable containing the image. The maximum length of the image is 254 bytes and has the following format:
```
[+][#] ... [.][#] ... [↑↑↑↑]
[-]
```

pack-element = A numeric-expression or numeric-variable.

Two digits are packed per byte. A sign, if included, occupies the high-order half-byte, and specifies the signs of the mantissa and exponent. Otherwise the absolute value of the pack-element is assumed.

EXAMPLES:
```
10 PACK (####) A$() FROM 0,N()
20 PACK (-#.#####↑↑↑↑) B$ FROM X↑2
30 PACK(F$) C$(1) FROM X,-1,Y()
```

REFER TO: UNPACK

PART

REFER TO: #PART
$RELEASE PART

PASSWORD

PURPOSE: A keyword used in DATE and TIME assignments.

REFER TO: DATE

TIME

PI

REFER TO: #PI

PLOT

PURPOSE: Send plotting control commands to the PLOT device.

SYNTAX PLOT [exp1] <[exp2],[exp3],[parm]> [,<[exp2],[exp3],[parm]>] ...

COMMENTS:

exp1 = A numeric-expression specifying the number of times the succeeding plot commands are to be repeated such that 1 <= exp1 <= 999. If omitted, a value of 1 is assumed.

exp2 = A numeric-expression normally specifying the number of plotting increments on the X-axis to move such that -999 <= exp2 <= 999. Exp2 can also be used to set character size and spacing. If omitted, a value of zero is assumed.

exp3 = A numeric-expression normally specifying the number of plotting increments on the Y-axis to move such that -999 <= exp3 <= 999. Exp 3 can also be used to set character spacing. If omitted, a value of zero is assumed.

parm = A plotter command parameter, an alpha-variable, or a literal-string. If an alpha-variable or literal-string, parm is text to be output by the plotter. The legal command parameters are:

U Pen up
D Pen down
R Reset
C Set character size
S Set character spacing

If omitted, a U parameter is assumed.

Not all plotters support all parameters, and plotting increments vary by model. Refer to your plotter manual for further information.

POS

PURPOSE: Determine the position of a character satisfying a specified relation.

SYNTAX POS ([-] alpha-variable-1 relation alpha-variable-2)
literal-string-1 literal-string-2
hh

COMMENTS:

relation = Any relational-operator (=, <>, <, <=, >, >=)

hh = A pair of hexadecimal digits such that h = 0-9, A-F.

POS causes the system to scan alpha-variable-1 or literal-string-1 for a character matching the condition specified by the relation and alpha-variable-2, literal-string-2, or hh. If a match is found, POS returns the byte position in alpha-variable-1 or literal-string-1 shere the match was found. Otherwise POS returns zero.

The search proceeds from left to right. If the - parameter is included, the search is from right to left. The POS function terminates as soon as a match is found.

Only the first byte of alpha-variable-2 or literal-2 is used in the relational comparison.

```
EXAMPLES: 10 P = POS(Q$()<FF)
          20 ON POS(HEX(101816)=K$) GOTO 100,200,300
          30 IF POS(STR(A$(4),X1,X2)<=B$(B))>0 THEN DATA LOAD BAT #X,(X0)X$()
```

PRINT

PURPOSE: Send data to an output device using a system-defined format.

```
SYNTAX    PRINT [print-element] , print-element ...
                               ;
```

```
COMMENTS:  print-element = a numeric-expression;
                           an alpha-variable;
                           a literal-string;
                           AT();
                           BOX();
                           HEXOF(); or
                           TAB().
```

In immediate mode, PRINT output is sent to the currently selected CO device. Otherwise it is sent to the currently selected PRINT device.

The comma and semicolon print-element separators determine if zoned output is used. The semi-colon designates that zoned output is not used. The comma indicates that the next print element is to be output at the start of the next 16-column zone.

If a print-element cannot completely fit in the space remaining as defined by the column-width in SELECT, it is printed on the following line.

```
EXAMPLES: 100 PRINT HEX(0306);AT(2,30);"Southern Data Systems, Inc."
          110 PRINT A$;TAB(25);VAL(B$,2),HEXOF(STR(B$,,2));
```

PRINTUSING

PURPOSE: Print numeric or alphanumeric data in a specified format.

```
SYNTAX    PRINTUSING image-spec [, print-element] ... [;]
                                [; print-element]
```

```
COMMENTS:     image-spec = A reference to an image representing the format of
                           the output. The image-spec may be an alpha-literal,
                           an alpha-variable, or the line number of an image
```

print-element = An expression, alpha-variable, or literal-string.

If more print-elements are specified than can be edited into the image, the image is reused until all print-elements have been output.

If fewer print-elements are specified than there are format specifications in the image, the image is used up to the first non-edited format specification.

If print-elements are included in PRINTUSING, the image must contain at least one format specification.

A semi-colon following the final print-element suppresses the automatic carriage return.

EXAMPLE:
```
100 PRINTUSING 9600
200 PRINTUSING "Please sign your check."
300 PRINTUSING "##,###.##--    ",X(1),X(2),X(3)
400 PRINTUSING I$,"GRAND TOTAL",X
500 PRINTUSING 15,M$,D$,Y$;
```

REFER TO: PRINTUSING TO

PRINTUSING TO

PURPOSE: Store a formatted print line in an alpha-variable.

SYNTAX
```
PRINTUSING TO alpha-variable, image-spec [, print-element] ... [;]
                                         [; print-element]
```

COMMENTS: alpha-variable = Any alpha-variable.

image-spec = A reference to an image representing the format of the output. The image-spec may be an alpha-literal, an alpha-variable, or the line number of an image (%) statement.

print-element = An expression, alpha-variable, or literal-string.

PRINTUSING TO is similar to PRINTUSING except that output is stored in the specified alpha-variable.

The first two bytes of the alpha-variable function as an offset pointer. The binary value +3 indicates the first byte of the alpha-variable in which formatted output will be stored. The pointer is automatically incremented by the number of bytes transferred. Consequently, output generated from a successive PRINTUSING TO statement to the same variable will be automatically appended to previous output.

On first use of the alpha-variable, or when the variable is to be reused, the first two bytes should be set to HEX(0000).

Output is generated as if a device-type of 7 was SELECTed with a line-width of zero. Where normally appropriate, a carriage return (HEX(0D)) is edited into the alpha-variable and is reflected in the binary pointer.

EXAMPLES:
```
PRINTUSING TO A$,"###.##",X(7)
PRINTUSING TO B$(),I$,B(1),B(2);
```

REFER TO: PRINTUSING

PSTAT

REFER TO: $PSTAT

R

PURPOSE: A parameter used to reference the "removable" platter in disk operations, and to select radians mode in trigonometric operations.

REFER TO: SELECT
Disk statements

RE

PURPOSE: A keyword used to modify other keywords.

REFER TO: MAT REDIM

READ

PURPOSE: Obtain values from DATA statements and sequentially assign them to READ variables.

SYNTAX READ variable [,variable]...

COMMENTS: variable = An alpha or numeric variable that receives a value of the same type from a DATA statement.

A READ statement must be used with a DATA statement.

When the number of variables is greater than the number of elements in the DATA statement(s), an error occurs.

When the number of variables is less than the number of DATA elements available, the next READ statement obtains data from the next sequential DATA element.

The RESTORE statement may be used to reset the DATA element pointer.

EXAMPLE: 10 READ X$,Y,Z$()

REFER TO: RESTORE
MAT READ

REDIM

PURPOSE: A keyword used in matrix redimensioning.

REFER TO: MAT REDIM

REFER TO: $RELEASE PART
$RELEASE TERMINAL

REM

PURPOSE: Insert comments in a program.

SYNTAX REM [%[↑]] text string

COMMENTS: text string = Any sequence of characters except the colon, which temrinates the ststament.

REM is not executable and can be placed anywhere in the program.

REM % and REM %↑ generate specialy formatted output when used with LISTD.

REM % causes a carriage return to be output prior to the text string, and the text string to be highlighted using a HEX(0E).

Rem %↑ causes a page-eject (HEX(0C)) to be output prior to the highlighted text string.

EXAMPLES:
```
10 REM Southern Data Systems, Inc.
20 REM % This is a highlighted remark
30 REM %↑ This highlighted remark prints at top-of-form
40 IF X+Y THEN REM remarks can appear anywhere in a program.
```

RENUMBER

PURPOSE: Reassign line-numbers to program text in memory.

SYNTAX RENUMBER [line#1] [-line#2] [TO line#3] [STEP s]

COMMENTS:

line#1 = If included, specifies the first line of program text to be renumbered. The default is the lowest numbered line.

line#2 = If included, specifies the last line of program text to be renumbered. The default is the highest numbered line.

line#3 = If included, specifies the increment. The default is the STEP value.

s = An integer that specifies the line-number increment such that 1 <= s <= 99. The default is 10.

All line-number references in the program are also renumbered.

EXAMPLES:
```
RENUMBER
RENUMBER 1500-1599 TO 1500
RENUMBER 9600-9840 TO 8000 STEP 2
RENUMBER -500
```

RESET

PURPOSE: Immediately stop program execution and partition I/O to all devices.

SYNTAX RESET Key

COMMENTS: RESET
1) Halts execution of the currently excecuting statement;
2) Resets all partition I/O devices;
3) CLOSEs all OPEN devices;
4) Flushes the system stacks;
5) Clears the interrupt table;
6) Turns off TRACE and PAUSE modes;
7) Selects the primary CI and CO devices;
8) Displays the READY and $MSG messages; and
9) Returns control to the console devices.

On multi-user systems, RESET sets the text pointer to the originating partition if global text was being executed. It can also be used to attach a terminal to a partition assigned to the null terminal.

HALT/STEP should be used in most cases to stop program execution. NEVER use RESET when writing to a disk.

RESTORE

PURPOSE: Reposition the data-pointer.

SYNTAX RESTORE [LINE line-number [,exp-1]]
[exp-2]

COMMENTS:
line-number = The line-number of a DATA statement. If a DATA statement does not exist on the specified line, the data-pointer is positioned at the next DATA statement in the program.

exp-1 = A numeric expression such that 1 <= exp-1 <= 65535 specifying the element within the DATA statement at which the data-pointer should be set. The truncated integer value is used.

exp-2 = A numeric expression such that 1 <= exp-2 <= 65535 specifying the element within the program at which the data-pointer should be set. The truncated integer value is used.

The data-pointer indicates the next DATA element that will be accessed by a READ statement. It is updated to point to the next consecutive DATA element with each successive READ.

RESTORE sets the data-pointer to access DATA elements in the partition in which the RESTORE is located.

RESTORE, RESTORE 0, and RESTORE 1 are equivalent.

EXAMPLES:
```
10 RESTORE
20 RESTORE X*2-1
30 RESTORE LINE 100
```

40 RESTORE LINE 100, X

REFER TO: READ
MAT READ

RETURN

PURPOSE: Return from a subroutine.

SYNTAX RETURN

COMMENTS: RETURN causes the program to return from a subroutine.

If the subroutine branch was made by executing a GOSUB or GOSUB' statement, execution continues at the statement following the GOSUB or GOSUB'.

If a DEFFN' subroutine was initiated from the keyboard in immeidate mode, execution halts and control is returned to the keyboard.

If a DEFFN' subroutine was initiated from the keyboard while executing an INPUT statement, RETURN branches back to the INPUT statement.

If a DEFFN' subroutine was initiated from the keyboard while executing a LINPUT statement, execution continues with the statement following the LINPUT.

RETURN clears subroutine return information and and intervening loop control information from the stacks.

EXAMPLE: 10 RETURN

REFER TO: GOSUB
GOSUB'

RETURN CLEAR

PURPOSE: Clear subroutine RETURN and FOR/TO loop control information from the stacks.

SYNTAX RETURN CLEAR [ALL]

COMMENTS: RETURN CLEAR is a dummy RETURN statement. By clearing the stacks of all information up to the last GOSUB, GOSUB', or Special Function Key-initated execution, it prevents program control from branching back to the keyboard or to the main program.

Execution continues with the statement following RETURN CLEAR.

If the ALL parameter is specified, all stack information is cleared.

EXAMPLES: 10 RETURN CLEAR
20 RETURN CLEAR ALL

REFER TO: RETURN

REWIND

PURPOSE: A Wang BASIC keyword used in tape cassette operations, which are no longer supported by BASIC-2.

RND

PURPOSE: Produce a random number between 0 and 1.

SYNTAX ... RND(n) [...]

COMMENTS:

n = A numeric-expression such that:
If n = 0, the first random number from the random number list is produced; or
If n <> 0, the next random number from the random number list is produced.

Following master initialization or a CLEAR command, the random number list "pointer" is moved to a random location (reseeded). Zero eases the debugging of programs using RND by allowing the programmer to produce the same series of random numbers during testing.

EXAMPLE: 10 X = RND(1)

ROTATE

PURPOSE: Rotate bits in an alpha-variable.

SYNTAX ROTATE [C] (alpha-variable, expression)

COMMENTS:

C = A parameter denoting that the entire alpha-variable is to be treated as a single, contiguous bit-string. If omitted, the bits in each byte are rotated.

expression = A numeric-expression the truncated integer value of which specifies the number of bits to rotate such that -8 <= expression <= 8. A positive value rotates the bits to the left, a negative value rotates the bits to the right.

All bytes are rotated including trailing spaces.

EXAMPLES:
```
10 ROTATE  (X$,-4)
20 ROTATEC (STR(Y$(),S,L),Z)
```

ROUND

PURPOSE: Round a numeric-expression to a specified decimal place; a SELECT parameter.

SYNTAX ... ROUND(expression,n) ...

COMMENTS:

COMMENTS: n = A numeric expression such that:
If n > 0, expression is rounded to the nth decimal place;
If n = 0, expression is rounded to the nearest integer; and
If n < 0, expression is rounded to the -(n)+1 places to the left of the decimal point.

IF X is the value of the expression, ROUND is equivalent to:
SGN(X) * INT(ABS(X)*10↑(FIX(N)) +.5) / 10↑(FIX(N))

EXAMPLE: 10 X = ROUND(X,N)

REFER TO: SELECT

RUN

PURPOSE: Resolve and execute the program currently in memory.

SYNTAX RUN [line-number [,statement-number]]

COMMENTS: When no parameters are specified, RUN clears all noncommon variables from memory before resolving the program. Common variables are not affected.

When line-number is specified, program execution begins on that line and non-common variables are not affected. Otherwise, execution begins at the lowest line-number.

When statement-number is specified, program execution begins with the specified statement within a multi-statement program line.

EXAMPLE: RUN
RUN 1500
RUN 600,2

S

PURPOSE: A parameter used in LIST and $HELP scrolling.

REFER TO: $HELP
LIST
LIST #
LIST '
LIST DC
LIST DT
LIST I
LIST T
LIST V

SAVE

PURPOSE: A keyword used in disk statements.

REFER TO: SAVE DC
SAVE DA
DATASAVE BA
DATASAVE BT
DATASAVE DA

```
DATASAVE DC
DATASAVE DC CLOSE
DATASAVE DC END
DATASAVE DC OPEN
```

SAVE DA

PURPOSE: Save program text in "direct address" mode.

SYNTAX
```
SAVE DA [<S >] pd [$] [file#,  ] [!] (sector [,var]) [start] [,end]
        [<SR>]       [address,] [P]
```

COMMENTS:

DA = A parameter indicating direct access mode.

<S> = A parameter indicating that all unnecessary spaces are to be removed.

<SR> = A parameter indicating that all unnecessary spaces and REM statements are to be removed.

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

$ = A parameter specifying that a read-after-write verification is to be performed.

! = A parameter indicating that the program text is to be scrambled.

P = A parameter indicating that the protect bit is to be set in the program file.

sector = A numeric-expression or alpha-variable indicating the first sector to be used to store program text. If an alpha-variable, the address is considered to be a two-byte binary value.

var = A variable which receives the address of the sector following the last saved. If alphanumeric, the value is a two-byte binary number.

start = The starting line-number of text to be saved. If omitted, the lowest line-number is assumed.

end = The last text line-number to be saved. If omitted, the highest line-number is assumed.

SAVE DA does not alter or check the disk index and must be used with care to prevent unintentional overwrites of valid data.

EXAMPLES:
```
10 SAVE DAT (400)
20 SAVE DAF$ (X,Q$) 100
30 SAVE DA <SR> T/D12, ! (X+300) 1000,9000
```

REFER TO: LOAD DA

PURPOSE: Save (or resave) program text in disk catalogue mode.

SYNTAX SAVE [DC][<S >] pd [$][f#,] [([name1])][!] name2 [start][,end][/com]
[<SR>] [da,] [(space)][P]

COMMENTS:

DC = An optional parameter indicating disk catalogue mode. If omitted, DC mode is assumed.

<S> = A parameter indicating that all unnecessary spaces are to be removed.

<SR> = A parameter indicating that all unnecessary spaces and REM statements are to be removed.

pd = Platter-designator (F, R or T).

f# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

da = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

$ = A parameter specifying that a read-after-write verification is to be performed.

name1 = A literal-string or alpha-variable containing the name of a scratched file to be overwritten. If omitted, name-1 is assumed to be name-2.

space = A numeric-expression inidicating the number of extra sectors to be reserved. If omitted, space = zero.

name2 = A literal-string or alpha-variable containing the new name of the file.

! = A parameter indicating that the program text is to be scrambled.

P = A parameter indicating that the protect bit is to be set in the program file.

start = The starting line-number of text to be saved. If omitted, the lowest line-number is assumed.

end = The last text line-number to be saved. If omitted, the highest line-number is assumed.

com = A alpha-variable or literal-string containing up to 234 characters of information to be stored in the program header sector, immediately following the time and date stamp.

If the parentheses contain a numeric-expression or are omitted, it is assumed that a new file is to be created. Otherwise, and existing scratched file is to be overwritten.

EXAMPLES:
```
10 SAVE T (4) "PROGRAM"
20 SAVE DCF$ () Q$
30 SAVE <SR> T/D12 ("OLDNAME","NEWNAME",1000
```

```
40 SAVE <S> T $ #4, (X$) ! Y$ 1000,9000
```

REFER TO: LOAD DC

SCRATCH

PURPOSE: Inactivate a disk cataloqued file.

SYNTAX
```
SCRATCH pd [file#,  ] file-name [,file-name] ...
           [address,]
```

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

file-name = A disk cataloqued file-name.

Scratched files may be rescratched without an error.

Scratched files are ignored by MOVE, and may be overwritten by SAVE and DATASAVE DC OPEN statements.

EXAMPLES:
```
10 SCRATCH T "PROG1"
20 SCRATCH R#2, "PROG2"
30 SCRATCH T/D11, "PROG3","DATA1"
```

REFER TO: DATASAVE DC OPEN
SAVE DC

SCRATCH DISK

PURPOSE: Create a cataloque index and cataloque area on disk.

SYNTAX
```
SCRATCH DISK ['] pd [file#,  ] [LS= exp-1,] END = exp-2
                   [address,]
```

COMMENTS:

' = A parameter indicating that the new, more efficient file-name hashing algorithm is to be used in the index.

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

LS= = A parameter indicating that a specified number of sectors, given by exp-1, is to be used for the index. If omitted, exp-1 is assumed to be 24.

exp-1 = A numeric expression such that 1 <= exp1 <= 255.

exp-2 = A numeric-expression specifying the sector address marking the end of the catalogue area.

SCRATCH DISK erases any previous disk index information, and overwrites the index sectors with catalogue control information.

The number of files that can be contained in the index is given by the expression: exp-1 * 16 - 1.

EXAMPLES:
```
10 SCRATCH DISK T END = 1023
20 SCRATCH DISK F #2, LS=30, END=19583
30 SCRATCH DISK ' T/D10, LS=255, END=52607
```

REFER TO: MOVE END

SCREEN

REFER TO: INPUT SCREEN

SEARCH

REFER TO: MAT SEARCH

SELECT

PURPOSE: Select system options and I/O devices.

SYNTAX SELECT select-parm [,select-parm] ...

COMMENTS: select-parm = A parameter denoting a system option, a class of I/O and associated I/O device address, or a device table slot and associated I/O device address.

Available Select Parameters

Parameter	Comments
D	Select degrees mode for trig functions
R	Select radians mode for trig functions
G	Select gradians mode for trig functions
ERROR [> error-code]	Supress math errors
[NO]ROUND	Math result rounding or truncation
P [digit]	User-readable output pause
LINE expression	Set height in lines for scroll functions
CI device-spec	Console-input device
INPUT device-spec	Input operations
CO device-spec [(w)]	Console-output device
PRINT device-spec [(w)]	Print device
LIST device-spec [(w)]	List device
PLOT device-spec	Plotting device
TAPE device-spec	$GIO and $IF ON/OFF device
DISK device-spec	Primary disk device
#n device-spec	Assign device table slot
ON [device-spec[GOSUB line]]	Assign and enable interrupts
ON CLEAR	Clear interrupt table
OFF[device-spec[GOSUB line]]	Assign and disable interrupts
ON ALERT GOSUB line	Enable $ALERT

TC device-spec — Switch MXE port to telecommunications
TERMINAL device-spec — Switch MXE port to terminal duties

error-code = An expression such that 60 <= error-code <= 69.

digit = A character from 0 to 9.

device-spec = A character string in the form /taa, where t is the device type, aa is the hardware address (the slash may be omitted where unambiguous); or an indirect assignment of the form <alpha-variable> where alpha variable contains the ASCII address in the form taa.

line = A line-number

w = A numeric-expression, the integer value of which specifies the line width of the device, such that 0 <= w <= 255.

See the BASIC-2 manual for further information on SELECT.

```
EXAMPLES: 10 SELECT D, ERROR >63, NO ROUND, P3, LINE 24, CI/001, INPUT 001
          20 SELECT CO 005(80), PRINT 215(132), LIST 005(80), PLOT C14
          30 SELECT TAPE 07B, DISK D11, #1 B10, #15 /310
          40 SELECT ON /01A GOSUB 1000, OFF /01B GOSUB 2000
          50 SELECT ON CLEAR
          60 SELECT ON ALERT GOSUB 5500
          70 SELECT TC /A02, TERMINAL /A04
```

REFER TO: ON SELECT

SELECT @PART

PURPOSE: Specify the global parition in use.

SYNTAX SELECT [...] @PART partition-name [...]

COMMENTS: partition-name = A literal-string or alpha-variable containing a global partition name of 1-8 bytes. When the name is defined by all spaces, the originating partition is chosen for global operations.

The global partition can be selected from the memory bank of the calling partition or from the universal global area of Bank #1.

```
EXAMPLE:  10 SELECT @PART "GLOBAL"
          20 SELECT PRINT <P$>, @PART G$
```

REFER TO: DEFFN @PART

SGN

PURPOSE: Compute the logical sign of a numeric expression.

SYNTAX ... SGN(expression) [...]

COMMENTS: SGN [illegible]

COMMENTS: SGN is the signum function and returns a value based on the sign of the expression. The values are: 1 (positive); -1 (negative); and 0 (zero).

EXAMPLE: 10 X = SGN(Y)

SIN

PURPOSE: Calculates the sine of a numeric-expression.

SYNTAX ... SIN(expression) [...]

COMMENTS: The absolute value of the expression must be less than 1E10 and can represent degrees, radians, or gradians depending on the current SELECT parameter.

EXAMPLE: 10 X = SIN(Y)

REFER TO: SELECT

SORT

REFER TO: MAT SORT

SKIP

PURPOSE: A Wang BASIC command used in tape cassette operations, which are no longer supported in BASIC-2.

REFER TO: DSKIP

SPACE

PURPOSE: Determines the amount of partition memory available during execution.

SYNTAX ... SPACE ...

COMMENTS: The memory used by the value stack is taken into account.

EXAMPLE: 10 X = 16000 - SPACE

REFER TO: SPACEK

SPACEK

PURPOSE: Returns the size of the partition in kilobytes (K).

SYNTAX ... SPACEK ...

COMMENTS: Prior to memory configuration, SPACEK returns the total memory available to the system in K.

EXAMPLE: 10 X = SPACEK*1024-SPACE: REM Compute memory used by current program

REFER TO: SPACE

SQR

PURPOSE: Calculate the square root of a numeric-expression.

SYNTAX ... SQR(expression) [...]

COMMENTS: The expression must be positive.

EXAMPLE: 10 X = SQR(Y)

STEP

PURPOSE: A keyword used to specify an increment.

REFER TO: FOR
MAT SEARCH

STOP

PURPOSE: Halt program execution.

SYNTAX STOP [literal] [#]

COMMENTS: When executed, STOP displays the word "STOP" on the CO device, followed by the literal message if specified, and the line-number on which the STOP occurs if the # parameter is included.

STOP can be placed anywhere in the program.

The CONTINUE command or the HALT key may be used to resume program execution halted by a STOP.

EXAMPLES: 10 STOP
20 STOP "Printer Not Ready"
30 STOP #
40 STOP "An Error has occurred at Line"#

REFER TO: CONTINUE

STR

PURPOSE: Define a substring of an alpha-variable.

SYNTAX [...] STR(alpha-variable [,[s][,n]]) [...]

COMMENTS: alpha-variable = An alpha-scaler-variable, an alpha-array-element, or

an alpha-array-designator.

s = An expression whose integer value defines the position of the first character of the substring.

n = An expression whose integer value defines the number of characters in the substring.

The expressions s and n must contain values such that:
s,n > 0; s+n-1 < maximum number of characters in the alpha-variable.

If s is omitted, the substring begins with the first character in the alpha-variable.

If n is omitted, the remainder of the alpha-variable is used, including trailing spaces.

If both s and n are omitted, the entire value is used, including trailing spaces.

EXAMPLES:
```
10 STR(A$(4),5,2)=B$()
20 PRINTUSING "##/##/##",STR(D$,3,2),STR(D$,5,2),STR(D$,,2)
30 X$="VOID":STR(X$,6)=STR(X$)
40 LINPUT STR(A$(),,MIN(480,LEN(STR(A$()))))
```

SUB

PURPOSE: Subtract a binary number from a binary number in an alpha-variable with optional carry.

SYNTAX `receiver = [...] SUB[C] operand [...]`

COMMENTS:

receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

The difference of the binary values of the operands is stored in the receiver. If an alpha-variable does not immediately follow the =, the receiver is considered an operand in its calculation.

If C is omitted, the characters are subtracted from right to left with no bit carries between characters. When C is included, the value of the operand is considered to be a single, multi-byte binary number and carry propogates between bytes.

Trailing spaces are included in an alpha-variable value.

EXAMPLES:
```
10 A$ = SUB ALL(20)
20 A$ = SUBC HEX(008000)
30 STR(A$,,2)= B$ SUBC C$
```

T

PURPOSE: A LIST parameter and a platter-designator specifying that the device type in the disk address determines the platter to be used.

REFER TO: LIST T
Disk statements

TAB

PURPOSE: Print output starting at a specified column.

SYNTAX PRINT [...] TAB(expression) [...]

COMMENTS: expression = A numeric expression such that
0 <= expression <= 255

TAB must be used with PRINT.

TAB moves a cursor or print element to a position specified by the expression by inserting spaces in a line. Any characters that exist in the columns preceding the TAB are replaced by spaces.

Columns are numbered starting at zero. If the cursor or column counter is located at a column beyond that specified by TAB, the function is ignored.

EXAMPLES:
```
10 PRINT TAB(35);C$
20 PRINT TAB(10);"NAME";TAB(40);N$
```

REFER TO: PRINT

TAN

PURPOSE: Calculates the tangent of a numeric-expression.

SYNTAX ... TAN(expression) [...]

COMMENTS: The absolute value of the expression must be less than 1E10 and can represent degrees, radians, or gradians depending on the current SELECT parameter.

EXAMPLE:
```
10 X = TAN(Y)
```

REFER TO: SELECT

TAPE

PURPOSE: A parameter which defines a class of I/O.

REFER TO: SELECT

TC

PURPOSE: A parameter used to convert a 2236MXE port to telecommunications operations.

REFER TO: SELECT

TEMP

PURPOSE: A parameter which designates that

opened on a disk platter.

REFER TO: DATALOAD DC OPEN
DATASAVE DC OPEN

TERM

REFER TO: #TERM

TERMINAL

PURPOSE: A keyword used to select a 2236 MXE port for terminal operations, and with $RELEASE.

REFER TO: SELECT
$RELEASE TERMINAL

THEN

PURPOSE: A keyword used in IF statements.

REFER TO: IF

TIME

PURPOSE: A system variable used to retrieve or modify the system time-of-day.

SYNTAX Assignment
TIME = argument-1 PASSWORD argument-2

Retrieval
Alpha-variable = [...] TIME [...]

COMMENTS: argument-1 = An alpha-variable or literal-string containing the time to be assigned in HHMMSS (hours, minutes, seconds) in ASCII 24-hour format.

argument-2 = An alpha-variable or literal-string containing the system password assigned by $INIT. If the password supplied is incorrect, an X79 error is signaled.

Retrieval returns the time in HHMMSSCC (hours, minutes, seconds, centi-seconds) ASCII 24-hour format. If a value of "99999999" is retrieved, the system does not have a clock.

The TIME function may not be used as an argument in another function, nor may TIME be used as an argument in a GOSUB' or DEFFN' statement

EXAMPLES:
```
10 TIME = "123000" PASSWORD "SYSTEM"
20 TIME = T$ PASSWORD X$
30 T$ = TIME
40 X$ = "The Time is: "&TIME
```

PURPOSE: A keyword used as a separator.

REFER TO: FOR
UNPACK
$UNPACK
MAT SEARCH
RENUMBER

TRACE

PURPOSE: Display variable assignment, loop control and branching data during program execution.

SYNTAX TRACE [-T] [V variable-name]
TRACE [OFF]

COMMENTS: Trace mode can be specified by executing a TRACE statement within a program or in Immediate Mode. TRACE OFF turns off the Trace Mode. TRACE does not generate output in immediate mode.

IF the V and variable name are included, only assignments affecting the specified variable are output in addition to loop control and branching data.

If the -T parameter is included, TRANSFER TO references are not included in the TRACE output.

EXAMPLES: TRACE
TRACE V X$
TRACE -T V X
TRACE -T
TRACE OFF

REFER TO: TRACE DISK

TRACE DISK

PURPOSE: Send disk I/O TRACE output to the CO device.

SYNTAX TRACE DISK [OFF]

COMMENTS: OFF = A parameter which turns off TRACE DISK

TRACE DISK output is of the following form:
address op-code sector partition

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

op-code = The type of operation being performed.

Code	Operation
00	Read
40	Write
80	Read after write
2001	Copy

2010 Start multi-sector write
2011 End multi-sector write
2012 Verify

sector = A five-digit decimal sector address.

partition = The two-digit number of the partition performing the operation.

EXAMPLES: TRACE DISK
TRACE DISK OFF

REFER TO: TRACE

TRAN

REFER TO: $TRAN

TRN

PURPOSE: A keyword used to perform transpositions in matrix math.

REFER TO: MAT TRN

UNPACK

PURPOSE: Unpack an alpha-variable to a list of numeric variables.

SYNTAX UNPACK (image) alpha-variable TO numeric-var [,numeric-var] ...

COMMENTS:

image = A string of ASCII characters representing the image, or an alpha-variable containing the image. The maximum length of the image is 254 bytes and has the following format:
[+][#] ... [.][#] ... [↑↑↑↑]
[-]

numeric-var = A numeric-variable.

Data UNPACKed was usually packed in a PACK statement. The images should be the same to prevent changes in value. Data is unpacked sequentially to the variable list (element by element in the case of numeric arrays).

EXAMPLES: 10 UNPACK (####) A$() TO M,N()
20 UNPACK (-#.#####↑↑↑↑) B$ TO X
30 UNPACK (F$) C$(1) TO C,B(C)

REFER TO: PACK

V

PURPOSE: A LIST parameter.

REFER TO: LIST V

VAL

PURPOSE: Convert an alphanumeric value to a numeric value.

SYNTAX VAL(alpha-variable [,n])
literal-string

COMMENTS: n = A numeric integer constant representing the number of bytes to be converted such than 2 <= n <= 6.

From 1 to 6 bytes may be converted. If n is ommitted, one byte is assumed.

VAL is the inverse of the BIN function and is allowed wherever numeric functions are allowed.

EXAMPLES:
```
10 DATA LOAD DAT(VAL(X$(1)-S)A,B()
20 PRINT "The numeric value of ";HEXOF(STR(X$,,3));" is VAL(X$,3)
```

REFER TO: BIN

VER

PURPOSE: Verifies that the contents of an alpha-variable or literal-string correspond to a specified format.

SYNTAX VER(alpha-variable, format-spec)
literal-string,

COMMENTS: format-spec = An alpha-variable or literal-string.

The first expression is verified against the format defined in the second alpha-variable or literal-string.

VER returns the number of consecutive characters conforming to the format-spec. Verifications are maded on a character-by-character basis. The VER function terminates upon the first occurrance of a value in the alpha-variable or literal-string not meeting format requirements.

Format Character	Verification Criteria
A	Alphabetic only (A-Z, a-z)
#	Numeric only (0-9)
N	Alphabetic or numeric (A-Z, a-z, 0-9)
H	Hexadecimal (0-9, A-F)
P	Packed decimal (HEX(00)-HEX(99))
+	Sign (+, -, or blank)
X	Any character
Other	Exact match required

VER may be used wherever numeric functions are allowed.

EXAMPLES:
```
10 IF VER(S$,"##/##/##")<8 THEN GOSUB'100("ILLEGAL DATE FORMAT")
20 Q$=ALL("H"): IF VER(H$,Q$)=LEN(STR(H$)) THEN HEXUNPACK H$ TO A$
```

PURPOSE: Check the validity of data on disk.

SYNTAX VERIFY pd [file#,] [(start,end)] [error-variable]
[address,]

COMMENTS:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

address = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

start = A numeric-expression specifying the first sector to be verified.

end = A numeric-expression specifying the last sector to be verified.

error-variable = A numeric-variable which receives the address + 1 of a sector failing verification. If zero, the verify was successful.

VERIFY checks disk validity by performing cyclic and longitudunal redundancy checks on sector data and comparing the results to those stored with the sector.

If start and end are omitted, sectors zero through the last sector used are verified. Start and end must be specified on non-indexed platters.

If error-variable is omitted, error messages are output to the CO device. HALT may be used to exit verification following the next generated error message.

If error-variable is included, verification aborts on encountering an error.

EXAMPLES:
```
10 VERIFY T
20 VERIFY F#1, (0,3000)
30 VERIFY T/D11,X
40 VERIFY R (S,S+8000)E
```

REFER TO: COPY

XOR

PURPOSE: Perform a logical XOR (exclusive-or) function on alpha-operand bits; a logical operator.

SYNTAX receiver = [...] XOR operand [...]

COMMENTS:

receiver = alpha-variable [,alpha-variable] ...

operand = An alpha-variable, literal-string, ALL or BIN.

If an alpha-variable does not immediately follow the =, the receiver is considered an operand in the logical XOR evaluation.

The corresponding bits of each operand are compared and the result is stored in the receiver. If either bit, but not both, is a 1, the result is 1. Otherwise the result is 0.

The XOR function should not be confused with the logical operator XOR used in IF statements.

EXAMPLE:
```
10 A$ = XOR B$
20 C1$,C2$ = STR(D$,5,4) XOR ALL(0F)
```

REFER TO: IF

ZER

PURPOSE: A keyword used to zero a numeric-array in matrix math.

REFER TO: MAT ZER