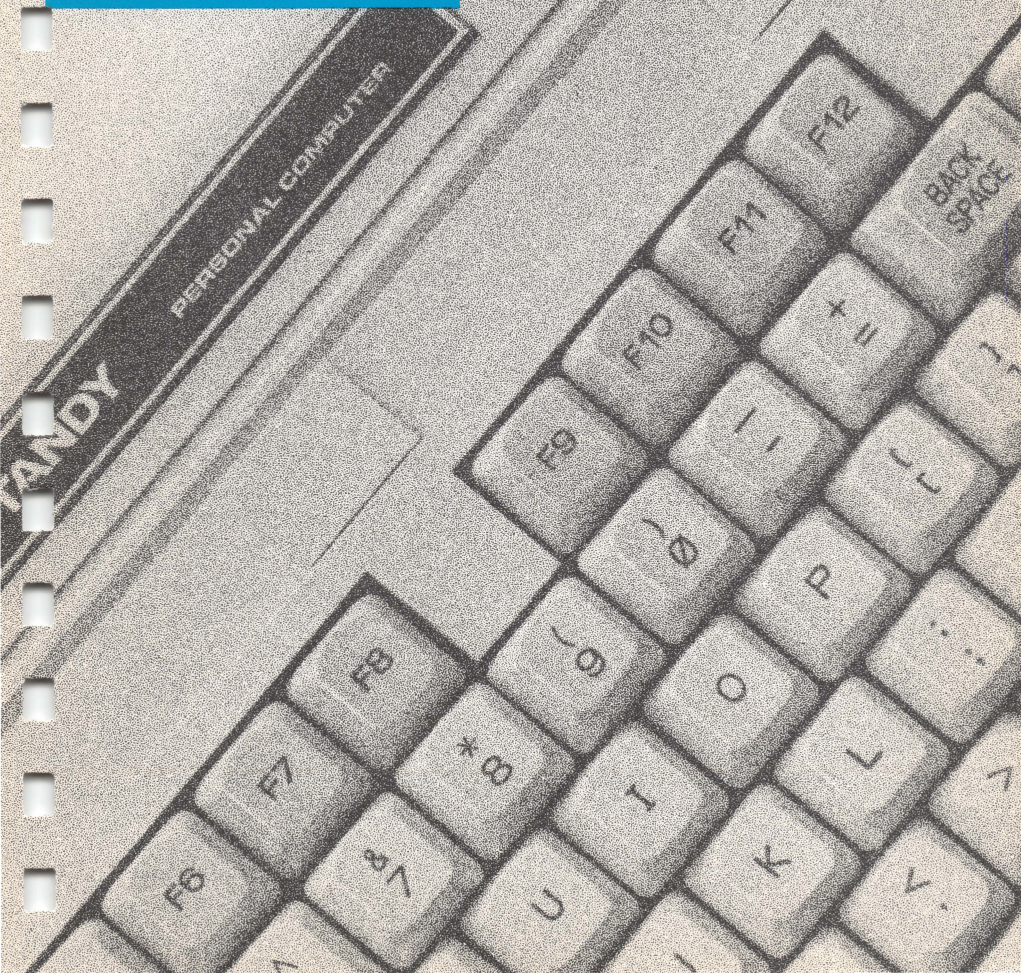
A Guide to the Tandy® 1000

☐ Introduction to the Tandy 1000/ A Tutorial to DeskMate™

☐ DeskMate, A Reference Manual

[✓ BASIC, A Reference Guide]



Tandy® 1000

BASIC

A Reference Guide

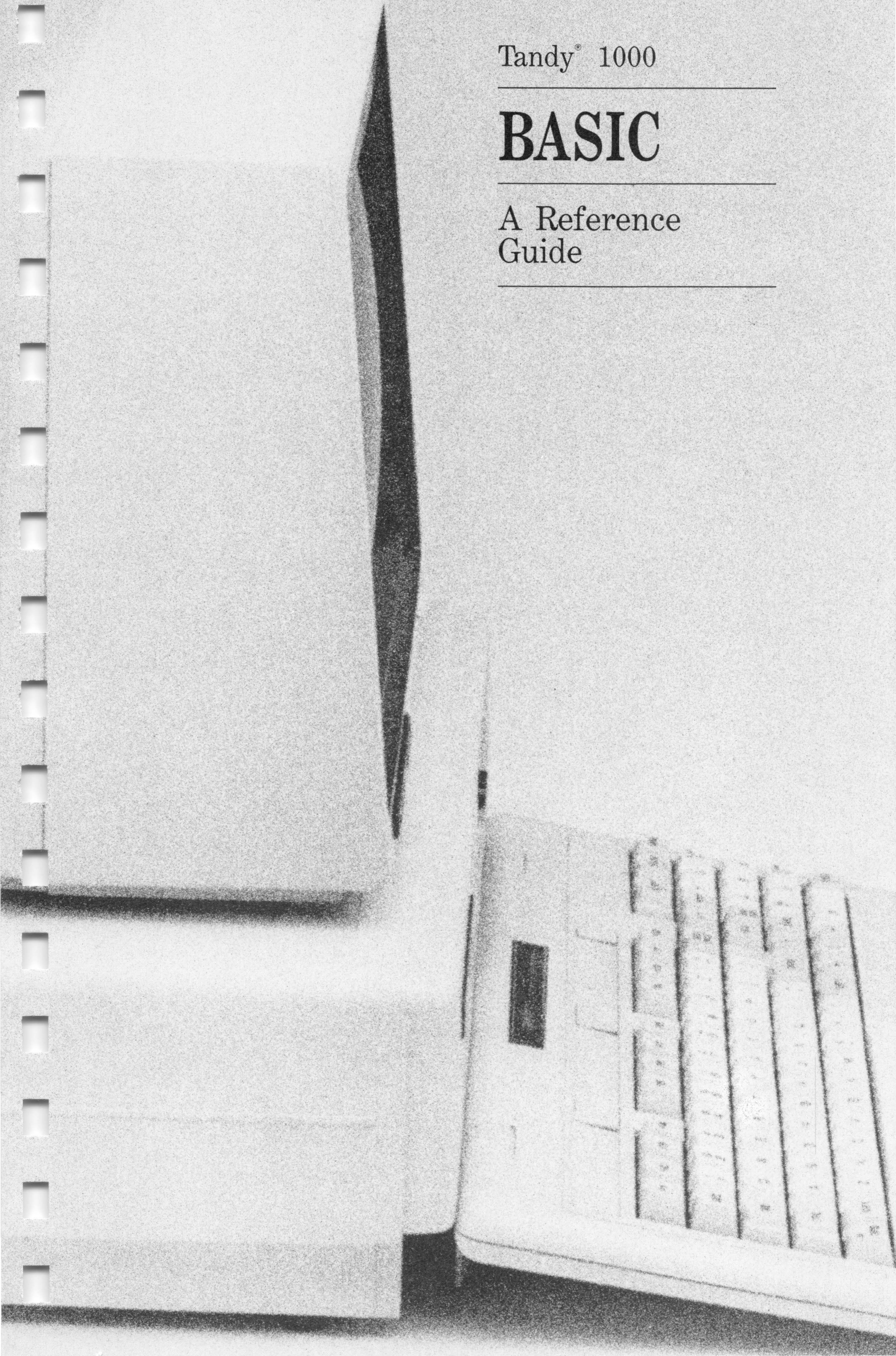


TABLE OF CONTENTS

BASIC

This guide outlines BASIC for MS™-DOS. It assumes that you are familiar with the BASIC language. For a full explanation of the concepts and commands referred to here, see your Radio Shack dealer for the *BASIC Reference Manual*, (Cat. No. 25-15Ø2). For a tutorial on how to use BASIC, Radio Shack carries the following book:

> *Learning BASIC for the Tandy 1000/2000*
> by David Lien, Cat. No. 25-15ØØ

Also see your local bookstore for tutorial books on BASIC.

Notations

The following notations are used throughout this guide:

CAPITALS	Material you enter exactly as it appears.
italics	Words, letters, characters, or values you put in command lines from a set of acceptable entries. Elsewhere, italics are used for emphasis.
. . . (ellipsis)	Items preceding the ellipsis may be repeated.
[ ]	Items enclosed in brackets are optional.
&H*nnnn*	*nnnn* is a hexadecimal number.
&O*nnnnn*	*nnnnn* is an octal number.
(KEYNAME)	A key on your keyboard.
b̸	Is used to indicate a space (ASCII Code 32) in text when spaces are an important part of a command or statement.

Loading BASIC

BASIC is supplied with your computer. To use BASIC, first load MS-DOS. To do so, turn on your computer and insert the MS-DOS/ BASIC diskette into Drive A; then press the reset button.

Enter the date and time as prompted. You can bypass these prompts by pressing ENTER; however, some BASIC statements make use of the system date and time. When the display shows the prompt:

```
A>
```

you can load BASIC, using this format:

BASIC [*pathname*][<*input-file*][>[>]*output-file*][/F:# *of files*] [/M:*highest memory location, maximum block size*][/C:*buffer size*][/S:*record length*][/D][/I]

For compatibility with other BASICs, you can type **BASICA**, instead of **BASIC**. Doing this causes the computer to load BASICA.COM, which in turn loads BASIC.

The only limitations imposed by BASICA are:

- /I is always on.
- The communications buffer size is limited to 4ØK bytes if the system has 1 RS-232 card or to 2ØK if it has 2 cards.

Examples:

```
BASIC
```
BASIC is loaded and 3 data files are reserved for your use.

```
BASIC MYFILE
```
BASIC is loaded. The program specified by MYFILE is loaded and executed.

```
BASIC MYFILE >DATA.IN
```
BASIC is loaded. The program specified by MYFILE is loaded and receives input from the file specified by DATA.IN rather than from the keyboard.

`BASIC MYFILE <DATA.IN >DATA.OUT`
BASIC and the program specified by MYFILE are loaded. BASIC now receives input from the file specified by DATA.IN. Output is directed to the file DATA.OUT, instead of the video display. Because 1 greater-than sign is used before the output-file, the output-file is overwritten. If 2 greater-than signs are used, the output is appended to the output-file.

`BASIC /F:1Ø /I`
BASIC is loaded. A maximum of 1Ø data files can be open at one time. /I, which tells BASIC not to dynamically allocate space during file operations, is required when using the /F option. BASIC reserves 6 files for your use and 4 for internal use. If the number of data files is not set, BASIC reserves 3 for your use.

`BASIC /S:256 /I`
BASIC is loaded. The maximum direct access record size is set at 256 bytes. If not defined, /S defaults to 128 bytes. /I, which tells BASIC not to dynamically allocate space during file operations, is required with the /S option.

`BASIC /C:128`
BASIC is loaded. The size of the receive buffer for RS232 is set to 128 bytes. If you omit /C, the buffer is set to 256 bytes. The transmit buffer is always 128 bytes.

`BASIC /M:32ØØØ,2Ø48`
BASIC is loaded with a reserved memory of 32768 bytes (2Ø48 x 16). The lower 32ØØØ bytes are used for BASIC with the 768 bytes above memory location 32ØØØ reserved for assembly-language routines. If the /M is omitted, the system reserves 64K bytes for BASIC. The *maximum block size* parameter must be set if you plan to use the SHELL statement.

`BASIC /D`
BASIC, including the Double Precision Transcendental math package, is loaded.

Filenames

Filenames (including program names) consist of 1-8 characters, beginning with an alpha character. Legal characters are the letters A-Z, the symbols `( ) { } @ # $ % & ! - _ ` ' ^ ~` and the numbers Ø-9. Examples of filenames are: DATĀFILE, Program1, Accnt-1, r'MAIL, A$b#C!.

Filenames can also contain an extension. Extensions consist of a period (.), followed by 1-3 characters. Legal extension characters are the same as for filenames. Examples of filenames with extensions are: DATAFILE.dat, Program1.BAS, MAIL.J27, A$b#C!.SSS.

To save or load files using other than the current directory or drive, you must use pathnames. Pathnames can include the drive, directory, filename, and filename extension. They can contain a maximum of 63 characters. An example of a pathname is a:\GAMESDIR\whizgame.bas.

Some BASIC commands, such as RMDIR (remove directory), require a directory path, instead of a pathname. A dirpath is the same as a pathname except that it excludes the filename.

Other commands let you specify a physical device for communication. The device names are: KYBD: (keyboard), SCRN: (screen), LPT1: (printer), and COM*n*: (Communications Channel 1 or 2).

Whenever you use a filename (with or without an extension), a complete pathname, a directory path, or a device name, you must enclose the entire name in quotation marks.

Loading and Running BASIC Programs

To load a BASIC program for execution or examination, type:

```
LOAD "MYFILE" (ENTER)
```

where MYFILE is the program to be loaded into memory. Because a path is not given, BASIC looks for MYFILE in the current directory.

Add **,R** after the filename or pathname to cause the file to execute automatically after loading. Using RUN instead of LOAD also causes a file to execute automatically after loading.

Saving BASIC Programs to Disk

The syntax for saving a BASIC program is:

```
SAVE "MYFILE" (ENTER)
```

saves the program in memory as MYFILE. Because a path is not specified, BASIC saves MYFILE in the current directory.

You can specify the drive and directory in which to save a file. For example, to name a file memos.bas and save it in the WORK directory on Drive B, type:

```
SAVE "B:\WORK\memos.bas" ENTER
```

Typing and Editing BASIC Programs

When BASIC displays the `OK` prompt, you can type in program lines or commands. When you press the ENTER key, BASIC looks at the first character of a line. If it is a digit, BASIC stores it in memory as a program line.

If the first character is not a digit, BASIC tries to execute the line as a command. For instance, type the following:

```
MILES=390 ENTER
GALLON=15 ENTER
PRINT MILES/GALLON ENTER
```

BASIC executes each command as it is entered.

Typing a Program

Each line must be preceded by a line number. At the `OK` prompt, type:

```
10 CLS
```

When the first line is completed, press ENTER.

Type the second line with its line number:

```
20 PRINT "COMPUTERS STORE CHARACTERS
   IN STRINGS" ENTER
```

Type the rest of the program in the same manner. It should look like this:

```
10 CLS
20 PRINT "COMPUTERS STORE CHARACTERS
   IN STRINGS"
30 INPUT "TYPE YOUR FIRST NAME -
   PRESS <ENTER>"; FIRSTNAME$
40 INPUT "TYPE YOUR SECOND NAME -
   PRESS <ENTER>"; LASTNAME$
50 PRINT "STRINGS CAN BE JOINED
   TOGETHER LIKE THIS," FIRSTNAME$ " "
   LASTNAME$
```

To execute this or any other BASIC program, at the OK prompt, type **RUN** and press (ENTER).

Editing the Program

There are 2 methods to edit BASIC program lines:

Method 1

You can retype the entire line. For example, to add the word CAN to the line, type:

```
20 PRINT "COMPUTERS CAN STORE
   CHARACTERS IN STRINGS" (ENTER)
```

Method 2

Use BASIC's special function keys for editing lines. A description of these keys and their functions follows.

If the line to edit is on the screen, you can use the arrow keys to move the cursor to the position at which you are going to make the changes. If the line is not displayed, you can edit it by typing:

```
EDIT line (ENTER)
```

where *line* is the number of the line to edit. After you make the changes, press (ENTER) to store them.

Special Function Keys

(CAPS)	switches to uppercase or uppercase/lowercase mode.
(SPACEBAR)	changes the current character to a blank and advances the cursor 1 position to the right.
(BACKSPACE) or (CTRL) (H)	backspaces the cursor, erasing the first character to the left. All characters to the right move left 1 position.
(CTRL)(BREAK) or (CTRL) (C)	interrupts line entry and starts over with a new line. Any changes previously made to the line are not saved.
(ENTER) or (CTRL) (M)	ends current line. BASIC reads the line.

ESC or CTRL U	erases the entire line from the screen, but not from memory.
CTRL L or CTRL HOME	clears the screen and positions the cursor at the first position in Row 1.
CTRL Z	clears the screen from the current cursor position to the end of the screen.
DELETE	deletes the character at the cursor position and moves all remaining characters 1 position to the left.
INSERT or CTRL R	turns on the insert mode if it is off; or off if it is on. The insert mode lets you add characters to the line at the cursor position.
HOME or CTRL K	moves the cursor to the first position in Row 1.
END or CTRL N	moves the cursor to the line end.
CTRL END or CTRL E	deletes all characters from the current cursor position to the end of the line.
TAB or CTRL I	advances the cursor to the next tab position. Tab positions are set at every 8 characters.
← or CTRL]	moves the cursor 1 position to the left.
→ or CTRL /	moves the cursor 1 position to the right.
↑ or CTRL 6	moves the cursor up 1 row to the character above the current cursor position.
↓ or CTRL −	moves the cursor down 1 row to the character below the current cursor position.
CTRL ← or CTRL B	moves the cursor left to the first character in the preceding word.
CTRL → or CTRL F	moves the cursor right to the first character in the next word.
CTRL G	rings the bell at the terminal.
CTRL J	issues a linefeed. This moves the cursor to the next line of the display without executing or storing the line.

Special Keys During Program Execution

HOLD	pauses execution. Press HOLD again to continue.
CTRL BREAK	terminates execution and returns you to BASIC's prompt.
ENTER or CTRL M	signifies the end of data entry. When a program or command prompts for data entry, press ENTER to end your response.

Function Key Settings

Your computer's function keys (F1-F12) are used to enter some keywords or commands. This chart shows the key assignments. See the KEY statement to display or reassign these keys.

F1	LIST
F2	RUN ENTER
F3	LOAD"
F4	SAVE"
F5	CONT ENTER
F6	,"LPT1:" ENTER
F7	TRON ENTER
F8	TROFF ENTER
F9	KEY
F10	SCREEN Ø,Ø,Ø ENTER
F11	(none)
F12	(none)

Typing Keywords Using The ALT Key

The ALT key provides a quick way to type BASIC keywords. To use ALT with the above keywords, press and hold down the ALT key while pressing the associated letter.

A	AUTO	J	(none)	S	SCREEN
B	BSAVE	K	KEY	T	THEN
C	COLOR	L	LOCATE	U	USING
D	DELETE	M	MOTOR*	V	VAL
E	ELSE	N	NEXT	W	WIDTH
F	FOR	O	OPEN	X	XOR
G	GOTO	P	PRINT	Y	(none)
H	HEX$	Q	(none)	Z	(none)
I	INPUT	R	RUN		

* MOTOR is a reserved word, but not recognized in this implementation of BASIC.

Data

Data, in the form of numbers, characters, or symbols, is information on which BASIC performs its operations. Data can be of 2 forms, string and numeric. As well, both string and numeric data can be of 2 types, variable and constant.

String Data

String data is a sequence of ASCII characters, graphics or non-ASCII symbols. The maximum length for a string is 255 characters.

Strings can contain either alpha or numeric characters. For example: "DOCUMENT 23", "LEVEL 13".

The dollar sign ($) is used to indicate a variable string name. For example:

```
NAME$="JOE"
```

When used in a program, strings are enclosed in quotation marks. When used in response to a prompt, strings do not require quotation marks.

Numeric Data

Numeric data consists of positive and negative numbers, which BASIC divides into 5 groups:

- **Integers** are whole numbers (without decimal points) in the range -32768 to +32767.
- **Single precision numbers** can be a maximum of 7 digits in the range $1Ø^{-38}$ to $1Ø^{+38}$. For example 1ØØØ1, -2ØØØ34, 123.456. If single precision numbers are more than 7 digits, they are displayed in the exponential form using the E form. For example: 1.756E5, .98E8, 1Ø4E-7.
- **Double precision numbers** can be a maximum of 16 digits and have a decimal point. They have the same range as single precision numbers. If they are more than 16 digits they are displayed in exponential format, using the D form. For example: 8.ØØ1ØØ7Ø8D12, -6.7765499824D16.
- **Hexadecimal** numbers represent a numeric system to the base of 16 instead of 1Ø. They can be 1 to 4 digits and are preceded by &H. The hexadecimal numbers are Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. For example: &HØ4, &HEE, &H4F, &H22.

- **Octal** numbers represent a numeric system to the base of 8. They can be 1 to 6 digits and are preceded by &O or &. The octal numbers are: Ø, 1, 2, 3, 4, 5, 6, 7. For example: &O7, &O123, &OØØ55, &66.

Numbering Systems

This chart shows the relationship of decimal, hexadecimal and octal numbers. The relationship of binary (base 2) numbers is also shown. This information is useful in graphic modes.

Decimal	Hexadecimal	Octal	Binary
Ø	Ø	Ø	ØØØØ
1	1	1	ØØØ1
2	2	2	ØØ1Ø
3	3	3	ØØ11
4	4	4	Ø1ØØ
5	5	5	Ø1Ø1
6	6	6	Ø11Ø
7	7	7	Ø111
8	8	1Ø	1ØØØ
9	9	11	1ØØ1
1Ø	A	12	1Ø1Ø
11	B	13	1Ø11
12	C	14	11ØØ
13	D	15	11Ø1
14	E	16	111Ø
15	F	17	1111

Numeric Constants

Numeric constants are values input to a program that are not subject to change. They can be in any of the 5 forms previously described. Notice that numeric constants:

- cannot contain punctuation. For example 1ØØ,ØØØ is not acceptable but 1ØØØØØ is acceptable.
- are evaluated when entered. If they are out of range for their type, an error message is immediately returned.
- are of several types. *Type* can be indicated by the use of symbols following the number. The symbols are:

  ! declares a single precision number. For example, 12.3456789Ø1234! is stored by BASIC as 12.34568.

E	declares a single precision exponential number. For example, the number 1.2E5 is stored as 120000.
#	declares a double precision number. However, single precision constants are not expanded by BASIC. For example, the number 1.5# is stored as 1.5 even though it is treated as a double precision number.
D	declares the number a double precision exponential number. For example, the number 1.2D2 is stored as 120.

Numeric Variables

BASIC classifies all numeric variables as single precision. You can change this classification by appending one of the following symbols to the variable name:

%	declares an integer variable. Examples are I%, FT%, COUNTER%.
!	declares a single precision variable. Examples are F!, NM!, BALANCE!.
#	declares a double precision variable. Examples are S#, AD#, TOTAL#.

The inclusion of one of these symbols creates a new and distinct variable name. For example, A%, A!, and A# can each represent a separate value.

Operators

An operator is a symbol or word that signifies an action to be performed on the associated data. Data items are called operands. The 4 types of operators are: arithmetic, string, relational, and logical.

Arithmetic Operators

^	Exponentiation. Calculates the power of a number. For example, 2^3 is 8 (2 to the power of 3 is the same as 2*2*2).
-	Negation or Unary Minus. Makes a number negative. For example, -10 is "negative ten."
*, /	Multiplication, Division. For example, 3*3 is 9, and 10/5 is 2.

\	Integer Division. BASIC rounds both operands to integers and truncates the result to an integer. Integer division is faster than standard division. For example, 10\4 is 2.
MOD	Modulus Arithmetic. BASIC performs integer division as described above and returns the *remainder* as an integer value. For example, 10 MOD 3 results in 1.
+, -	Addition, Subtraction. For example, 2+9 is 11, and 15–8 is 7.

When BASIC evaluates an arithmetic expression, all operands and the result are converted to the same degree of precision as the most precise operand. The arithmetic operators are listed in order of precedence, that is, the order in which BASIC executes them if 1 or more operators are in the same statement.

String Operator

The only string operator is the plus sign (+). It appends 1 string to another. For example:

```
PRINT "JOSEPH " + "P. " + "RAWLINGS"
```

prints `JOSEPH P. RAWLINGS`.

Relational Operators

The relational operators and their meanings are, in order of precedence:

=	Equal. Both operands are equal.
<	Less Than. The first operand is less than or precedes the second operand.
>	Greater Than. The first operand is greater than or follows the second operand.
>< or <>	Inequality. The operands are not equal.
<= or =<	Less Than or Equal To. The first operand is less than (precedes) or is equal to the second operand.
>= or =>	Greater Than or Equal To. The first operand is greater than (follows) or is equal to the second operand.

With numeric data, relational operators compare 2 pieces of data and the result is either *true* or *false*. If the relationship is true, BASIC returns -1. If the relationship is false, BASIC returns Ø (zero).

Relational operators are usually used within an IF/THEN statement. For example, the command:

```
IF A = 1 THEN PRINT "CORRECT"
```

displays the word `CORRECT` if the variable A is equal to 1.

With string data, relational operators compare character by character. When 2 characters do not match, BASIC checks to see which character has the lower ASCII value. The character with the lower value comes before the word with the higher value. Leading blanks are significant in string comparisons. The ASCII code for blank is 32.

```
PRINT "A" < "B"
```

compares the ASCII value of the 2 strings. The ASCII value for A is 65, and the ASCII value for B is 66. Because 65 is less than 66, BASIC returns -1.

Logical Operators

Logical operators, also known as Boolean operators, make comparisons of a set of true/false values and return a true or false result. This table shows the result for each logical operator given the described true/false values. True is 1 and false is Ø.

Operator	Meaning of Operation	First Operand	Second Operand	Result
NOT	When the result is the opposite of the operand.	1		Ø
		Ø		1
AND	When both values are true, the result is true. Otherwise, the result is false.	1	1	1
		1	Ø	Ø
		Ø	1	Ø
		Ø	Ø	Ø
OR	When both values are false, the result is false Otherwise, the result is true.	1	1	1
		1	Ø	1
		Ø	1	1
		Ø	Ø	Ø
XOR	When one of the values is true, the result is true. Otherwise, the result is false.	1	1	Ø
		1	Ø	1
		Ø	1	1
		Ø	Ø	Ø
EQV	When both values are true or both values are false, the result is true.	1	1	1
		1	Ø	Ø
		Ø	1	Ø
		Ø	Ø	1
IMP	The result is true unless the first value is true and the second is false.	1	1	1
		1	Ø	Ø
		Ø	1	1
		Ø	Ø	1

Hierarchy of Operators

This list shows the operators in the order that BASIC performs the operations in a statement. Remember, BASIC evaluates statements from left to right. Operators with the same level of hierarchy are shown on the same line.

```
^
unary -
* /
\
MOD
+ -
<  >  =  <=  >=  <>
NOT
AND
OR  XOR
EQV
IMP
```

Color and Graphics

Your computer has a wide range of color and graphics options. Color, as used in these references, indicates a color in the current screen mode.

Resolution

You have the option of 3 resolution screens, low, medium and high, as noted below. The horizontal length or points (x axis) is given first followed by the vertical length (y axis).

Low resolution	16Ø x 2ØØ points
Medium resolution	32Ø x 2ØØ points
High resolution	64Ø x 2ØØ points

The aspect ratio is a comparison of the number of points per inch vertically to the number of points per inch horizontally. This ratio is calculated by the formula:

$$\text{aspect ratio} = \frac{\text{vertical points}}{\text{area height}} \div \frac{\text{horizontal points}}{\text{area width}}$$

Colors

BASIC has three color sets: a 2-color set, a 4-color set and a 16-color set. They function as follows:

Color Set	**Attributes**
2 colors	Black and white. The background is black and the foreground is white. These colors cannot be changed.
4 colors	One set or *palette* of 4 colors. Each color is assigned a number. The numbers and their corresponding colors are:

No.	**Color**
Ø	*C.B.C.*
1	cyan
2	magenta
3	white

Screen Mode 1, however, has 2 palettes. The two sets and their corresponding numbers are:

No.	**Palette Ø**	**Palette 1**
Ø	*C.B.C.*	*C.B.C.*
1	green	cyan
2	red	magenta
3	brown	white

C.B.C. is the current background color and is initially set to black. You may change the background color to any of the colors in the 16-color set.

16 colors — One palette with 16 colors. Each color is numbered as shown below:

No.	**Color**	**No.**	**Color**
Ø	black	8	dark gray
1	blue	9	light blue
2	green	1Ø	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	gray	15	white

Video Pages

BASIC sets aside memory for the video display. The amount of memory necessary depends on the screen mode you choose. BASIC initially sets 16K aside for video memory. You can change this with the CLEAR statement.

Video memory is divided into pages. You can store information to one page while displaying another. The amount of memory required for each SCREEN mode is detailed in the following section.

Screen Modes

The color set and screen resolution are set using the SCREEN command.

There are 6 SCREEN modes as follows:

Screen Mode Ø (Text Mode)

Color Set:	16
Graphics Resolution:	not available
Text Width:	4Ø or 8Ø
Video Page Size:	If WIDTH=4Ø, 2Ø48 bytes, 8 pages max. If WIDTH=8Ø, 4Ø96 bytes, 4 pages max.

Screen Mode 1

Color Set:	4 (2 palettes)
Graphics Resolution:	medium resolution 32Ø x 2ØØ
Aspect Ratio:	5/6
Text Width:	4Ø
Video Page Size:	16384 bytes
Max. No. of Pages:	8

Screen Mode 2

Color Set:	2
Graphics Resolution:	high resolution 64Ø x 2ØØ
Aspect Ratio:	5/12
Text Width:	8Ø
Video Page Size:	16384 bytes
Max. No. of Pages:	8

Screen Mode 3

Color Set:	16
Graphics Resolution:	low resolution 16Ø x 2ØØ
Aspect Ratio:	5/3
Text Width:	2Ø
Video Page Size:	16384 bytes
Max. No. of Pages:	8

Screen Mode 4

Color Set:	4
Graphics Resolution:	medium resolution 32Ø x 2ØØ
Aspect Ratio:	5/6
Text Width:	4Ø
Video Page Size:	16384 bytes
Max. No. of Pages:	8

Screen Mode 5

Color Set:	16
Graphics Resolution:	medium resolution 32Ø x 2ØØ
Aspect Ratio:	5/6
Text Width:	4Ø
Video Page Size:	32768 bytes
Max. No. of Pages:	4

Screen Mode 6

Color Set:	4
Graphics Resolution:	high resolution 64Ø x 2ØØ
Aspect Ratio:	5/12
Text Width:	8Ø
Video Page Size:	32768 bytes
Max. No. of Pages:	4

Keywords

ABS(*number*)

Returns the absolute value of *number*.

```
PRINT ABS(-44)          X = ABS(Y)
```

ASC(*string*)

Returns the ASCII code (a decimal number) for the first character of *string*.

```
PRINT ASC("A")          N=ASC(B$)
```

ATN(*number*)

Returns the arctangent of *number* in radians.

```
PRINT ATN(7)           X = ATN(Y/3) * 57.29578
```

AUTO [*line*][,*increment*]

Automatically generates a line number when you press ENTER. If *line* already exists in memory, BASIC displays an asterisk after the number. To turn off AUTO, press BREAK.

Line is the starting line number. Default = Line 1Ø.

Increment is the increment to use when generating line numbers. Default = 1Ø.

```
AUTO                    AUTO 1ØØ,5Ø
```

BEEP [*switch*]

Sounds the speaker at 8ØØ Hz for ¼ second.

Use BEEP with SOUND to direct sound to the computer's speaker or an external speaker (or both).

```
BEEP ON: SOUND ON
```
directs sound to both speakers

```
BEEP OFF: SOUND OFF
```
turns off sound to both speakers

```
BEEP ON: SOUND OFF
```
directs sound to the internal speaker only

```
BEEP OFF: SOUND ON
```
directs sound to external speaker only

```
IF X > 2Ø THEN BEEP
```

BLOAD *pathname*[,*offset*]

Loads a memory image file into memory.

Offset is the number of bytes into the current segment where BASIC loads the image. It must be in the range Ø to 65535. Default = value set by BSAVE.

```
BLOAD "prog1.sav"     BLOAD "prog2.sav",Ø
```

BSAVE *pathname,offset,length*

Saves the contents of an area of memory as a disk file (memory image file).

Offset is the number of bytes into the current segment where BASIC starts saving. It must be in the range Ø to 65535.

Length is the length in bytes of the memory image file to be saved. It must be in the range 1 to 65535.

```
BSAVE "prog2.sav",5Ø,1ØØØ
```

CALL *variable* [(*parameter list*)]

Transfers program control to an assembly-language subroutine stored at *variable.*

Variable contains the offset into the current segment where the subroutine starts in memory. The offset must be on a 16-byte boundary.

Parameter list is the variables that are passed to the external subroutine.

```
CALL C                  CALL C (A$,Z,X)
```

CALLS *variable* [(*parameter list*)]

Transfers program control to an MS™-FORTRAN routine.

Arguments are described in the CALL statement.

```
CALLS X                 CALLS X (S$)
```

CDBL(*number*)

Converts *number* to double precision.

```
PRINT CDBL(465.342)         Z=CDBL(A)
```

CHAIN [MERGE] *pathname* [,[,*line*] [,ALL] [,DELETE *line-line*]]

Lets the current program load and execute another program.

Pathname is the program you want to chain. It must be saved in ASCII format. See SAVE.

Line is the line number where execution begins in the chained program. It must be preceded by a comma (,). If you use the ALL or DELETE option and do not specify a line number, you must specify a comma for *line*. Default = first program line of the chained program.

ALL tells BASIC to pass every variable in the current program to the chained program. If you omit ALL, the current program must contain a COMMON statement to pass variables to the chained program.

MERGE overlays the lines of the chained program with the current program.

DELETE deletes lines in the overlay so that you can merge in a new overlay.

```
CHAIN "prog2"
CHAIN "subprog.bas",, ALL
```

CHDIR *dirpath*

Changes the current directory to *dirpath.*

```
CHDIR "B:\ACCTS\RECVBLE"          CHDIR ".."
```

CHR$(*code*)

Returns the character corresponding to an ASCII or control *code.*

```
PRINT CHR$(35)          C$=CHR$(32)
```

CINT(*number*)

Converts *number* to an integer by rounding the fraction portion of *number*. *Number* must be in the range -32768 to 32767.

```
PRINT CINT(1.6)          Z=CINT(-1.67)
```

CIRCLE [STEP] (*x*,*y*),*radius* [,*color* [,*start*,*end* [,*aspect*]]]

Graphics. Draws an ellipse, the center of which is (*x*,*y*), on the screen.

STEP designates (*x*,*y*) as relative coordinates.

Radius is the major axis of the ellipse.

Start,*end* are the beginning and ending angles in radians. They must be in the range -6.283186 to 6.283186, or -2 * PI to 2 * PI.

Aspect is the ratio of the x-radius to the y-radius in terms of coordinates. If *aspect* is less than 1, *radius* is the x-radius and is measured in points in the horizontal direction. If *aspect* is greater than 1, *radius* is the y-radius and is measured in points in the vertical direction.

```
CIRCLE (150,100),50
```

CLEAR [*,memory location*] [*,stack space*] [*,video memory*]

Frees memory for data without erasing the program currently in memory. CLEAR erases all arrays, sets numeric variables to zero and string variables to null, and erases any information set using a DEF statement, such as DEF SEG and DEF FN. CLEAR also turns off the SOUND, PEN, and STRIG functions and resets the music background.

Memory location specifies the highest memory location available for BASIC.

Stack space specifies the amount of memory to set aside for temporarily storing internal data and addresses during subroutine calls and during FOR/NEXT loops. Default = 768 bytes or one-eighth of the memory available, whichever is smaller.

Video memory specifies the amount of memory to be set aside as video memory. If the amount is not a multiple of 16K, BASIC rounds it down to the nearest multiple of 16K. Default = 16K (16384).

```
CLEAR    CLEAR, 45000        CLEAR ,,,32768
```

CLOSE [*buffer,...*]

Closes access to a disk file or communications channel. If you omit *buffer*, BASIC closes all open files.

```
CLOSE                        CLOSE 1, 2, 8
```

CLS

Clears the screen (or active viewport) and returns the cursor to the home position. Home is Row Ø, Column Ø (the upper left corner of the screen).

```
CLS
```

COLOR [*background*] [,[*palette*]]
(Screen Mode 1)

COLOR [*foreground*][,[*background*]]
(Screen Modes 3-6)

Graphics. Selects the background color and either the palette or foreground colors, depending on the current screen mode.

Palette specifies which palette to use in Screen Mode 1 and may be Ø or 1.

```
COLOR 9,Ø                COLOR ,3
```

COLOR [*foreground*][,[*background*][,*border*]]

Text Mode Only. Selects the display colors for the foreground, background, and border for Screen Mode Ø. COLOR can use any of the colors in the 16-color set as foreground and border. Specify *color* + 16 as *foreground* to get a blinking foreground. *Background* can be Colors Ø to 7.

```
COLOR Ø,7               COLOR 1, Ø
```

COM(*channel*) *action*

Turns on, turns off, or temporarily halts the trapping of activity on the communications channel. *Channel* specifies communications channel 1 or 2.

COM() ON	enables communications trapping.
COM() OFF	disables communications trapping.
COM() STOP	temporarily suspends communications trapping.

COMMON *variable*[,*variable*,...]

Passes *variables* to a chained program. Both programs in the chain should contain a COMMON statement.

```
COMMON A, B$, C, D(),G$()
```

CONT

Resumes execution of a program stopped by either CTRL BREAK or the execution of a STOP or an END statement.

```
CONT
```

COS(*number*)

Returns the cosine of *number*, in radians.

```
PRINT COS(5.8)          Y = COS(X * .0174533)
```

CSNG(*number)*

Converts number to single precision. BASIC rounds the number when converting it to single precision.

```
PRINT CSNG(.1453885509)          Z=CSNG(A#)
```

CSRLIN

Returns the current row position of the cursor.

```
PRINT CSRLIN              A=CSRLIN
```

CVD(*8-byte string*)

Converts an 8-byte string to a double precision number. Use to restore data to numeric form after it is read from the disk.

```
A# = CVD(GROSSPAY$)      D=CVD(TOTAL$)
```

CVI(*2-byte string*)

Converts a 2-byte string to an integer. Use to restore data to numeric form after it is read from the disk.

```
A%=CVI(INVTRY$)          I=CVI(QTY$)
```

CVS(*4-byte string*)

Converts a 4-byte string to a single precision number. Use to restore data to numeric form after it is read from the disk.

```
A!=CVS(TOTAL$)          S=CVS(DOLLR$)
```

DATA *constant* [*,constant,...*]

Stores numeric and string constants to be accessed by a READ statement. String constants containing delimiters, such as leading or trailing blanks, colons, or commas, must be enclosed in double quotation marks when used in DATA statements.

```
DATA NEW YORK, CHICAGO, LOS ANGELES
```

DATE$[= *string*]

Sets the date or retrieves the current date.

String is a literal, enclosed in quotation marks, that sets the date by assigning its value to DATE$. Month may be any number Ø1-12, day may be Ø1-31, and year may be Ø1-99 or 198Ø-2Ø99. If you omit *string*, BASIC retrieves the current date.

```
DATE$="Ø4/17/85"         TODAY$ = DATE$
```

DEFDBL *letter* [*,letter,...*]

Defines any variables beginning with *letter(s)* as double precision variables.

```
DEFDBL A                  DEFDBL J-O
```

DEF FN*name* [(*argument list*)] = *expression*

Defines *name* as a function according to *expression.*

Name is a valid variable name.

Argument list is a list of dummy variables used in *expression.* They are replaced on a one-to-one basis with the variables or values given when the function is called.

Expression defines the operation to be performed.

```
DEF FNR = RND(1)*69+10
DEF FNW# (A#,B#)=(A#-B#)^2
```

DEFINT *letter* [,*letter*,...]

Defines any variables beginning with *letter(s)* as integer variables.

```
DEFINT L                      DEFINT A-G
```

DEF SEG[=*address*]

Assigns the current segment address. The segment address is used by BLOAD, BSAVE, CALL, PEEK, POKE, and USR.

Address can be expressed as an integer or a hexadecimal value. *Address* must be on a 16-byte boundary. Default=BASIC's data segment (DS).

```
DEF SEG                    DEF SEG=&HB800
```

DEFSNG *letter* [,*letter*,...]

Defines any variables beginning with *letter(s)* as single precision variables.

```
DEFSNG T                      DEFSNG Q-Z
```

DEFSTR *letter* [,*letter*,..]

Defines any variables beginning with *letter(s)* as string variables.

```
DEFSTR A                      DEFSTR G-M
```

DEF USR[*number*] = *offset*

Defines the user number and segment offset of a subroutine to be called by the USR function.

Number may be an integer in the range Ø to 9. Default=USRØ.

Offset is the number of bytes from the current segment address where the subroutine begins. Must be an integer in the range Ø to 65535.

```
DEFUSR = Ø            DEF USR3 = &HØØ2Ø
```

DELETE *line1-line2*

Deletes *line1* through *line2* of the program in memory. If you omit *line1*, BASIC deletes from the beginning of the program. If you omit *line2*, BASIC deletes to the end of the program. Use a period (.) to indicate the current line.

```
DELETE 7Ø             DELETE .-11Ø
```

DIM *array*(*dimension*)[,*array*(*dimension*),...]

Sets aside storage for *arrays* with the *dimensions* you specify.

Array is the variable name of a string, integer, single precision, or double precision variable name.

Dimension is 1 or more integer numbers separated by commas that define the dimensions of the array.

```
DIM AR(1ØØ)           DIM L1%(8,25)
```

DRAW *string*

Graphics. Draws an image on the screen.

String specifies 1 or more of the movement commands listed below.

Movement Commands

Movement commands begin movement from the current graphics position, which is the coordinate of the last graphics point plotted with another graphics command. Current position defaults to the center of the screen if no previous graphics command has been executed.

U [*n*]	Moves up *n* points.
D [*n*]	Moves down *n* points.
L [*n*]	Moves left *n* points.
R [*n*]	Moves right *n* points.
E [*n*]	Moves diagonally up and right *n* points.
F [*n*]	Moves diagonally down and right *n* points.
G [*n*]	Moves diagonally down and left *n* points.
H [*n*]	Moves diagonally up and left *n* points.
M*x,y*	Moves to point *x,y*. If you precede *x* with a plus (+) or minus (−) sign, DRAW assumes it is a relative position. Otherwise, it is an absolute position.

Prefix Commands

Prefix commands can precede the movement commands. They must be enclosed in quotation marks.

B	plots no points after move.
N	returns to original position when move is complete.
A*angle*	sets angle of move. *Angle* may be 0 to 3 (0 = 0 degrees, 1 = 90 degrees, 2 = 180 degrees, and 3 = 270 degrees).
C*color*	sets color.
P*color,border*	paints using *color* and *border*.
S*factor*	sets scale factor. *Factor* is an integer in the range 1 to 255. The scale factor is *factor* divided by 4. Default=4 (scale of 1).
TA*angle*	moves at the specified angle. *Angle* is in the range -360 to +360. If *angle* is positive, movement is counterclockwise. If *angle* is negative, movement is clockwise.
X*variable*;	executes a substring. The X command lets you execute a second substring from the first string, much like the GOSUB statement. *Variable* is a string variable in your program that contains the substring you want to execute. The semicolon after *variable* is required.

```
DRAW "U30;"+"D30;"+"L40;"+"R40;"
```

EDIT *line*

Enters the Edit mode. BASIC displays *line* for editing. Use a period (.) to indicate the current line.

```
EDIT 100            EDIT .
```

END

Ends program execution and closes all files.

```
END
```

ENVIRON *"parameter id=text"* [;*"parameter id=text"*,...]

Advanced Statement. Lets you modify BASIC's Environment String Table, such as to change the PATH parameter for a child process or to pass parameters to a child process. BASIC's Environment String Table is initially empty.

Parameter id is the name of the parameter.

Text is the new parameter text. It must be separated from *parameter id* by an equal sign (=) or a space. If you omit *text*, or specify a null string or a semicolon (;), BASIC removes the parameter from the Environment String Table and compresses the table.

Parameter id=text must be enclosed in double quotation marks and be entered in uppercase characters.

```
ENVIRON "PATH=A:\"
ENVIRON "SALES=MYSALES"
```

ENVIRON$ [(*"parameter id"*)] [(*number*)]

Advanced Function. Returns the specified environment string from BASIC's Environment String Table.

Parameter id is the parameter for which to search and must be enclosed in quotation marks.

Number specifies which parameter to return by its position within the table.

Number and *parameter id* are mutually exclusive; only one may be specified on the command line.

```
PRINT ENVIRON$("PATH")
```

EOF(*buffer*)

Function. Detects the end of a file. *Buffer* is the number assigned to the file when you opened it.

Sequential files: EOF returns Ø (false), when the end-of-file record has not been read yet, and −1 (true), when it has been read.

Direct access files: EOF returns −1 (true) if the last executed GET statement was unable to read an entire record because of an attempt to read beyond the physical end of the file.

```
IF EOF(1) THEN GOTO 154Ø
```

EOF(*buffer*)

Communications. Detects an empty input queue for communications files. *Buffer* is the number assigned to the file when you opened it.

ASCII mode: EOF returns −1 (true) if a CONTROL-Z is received. EOF remains true until the device is closed.

Binary mode: EOF returns −1 (true) when the input queue is empty. EOF becomes false when the input queue is not empty.

```
IF EOF(3) THEN RETURN
```

ERASE *array*[,*array*,...]

Erases 1 or more *arrays* from memory. Lets you either redimension arrays or use their previously allocated space in memory for other purposes.

```
ERASE C                  ERASE G, H, I, Z$
```

ERDEV

Advanced Function. Returns the value of a device error within MS-DOS as set by the Interrupt 24 handler.

The lower 8 bits of ERDEV contain the Interrupt 24 error code.

```
ERDEV
```

ERDEV$

Advanced Function. Returns the name of the device (as set by the Interrupt 24 handler) when a device error occurs. If the error occurred on a character device, ERDEV$ returns the 8-byte character device name. If the error does not occur on a character device, ERDEV$ returns the 2-character block device name.

```
ERDEV$
```

ERL

Returns the number of the line in which an error has occurred. If no error has occurred, ERL returns 0. If the error occurs while you are entering something at the prompt, ERL returns 65535 (the largest number that can be represented in 2 bytes).

```
PRINT ERL                E = ERL
```

ERR

Returns the error code if an error has occurred.

```
IF ERR = 7 THEN 1000 ELSE 2000
```

ERROR *code*

Simulates a specified error during program execution.

Code is an integer expression in the range Ø to 255 specifying one of BASIC's error codes.

```
ERROR 1
```

EXP(*number*)

Returns the natural exponent of *number*, that is, *e* (base of natural logarithms) to the power of *number*. *Number* must be less than or equal to 88.Ø2968.

```
PRINT EXP(-2)      A=EXP(-6)
```

FIELD *buffer*, *length* AS *variable* [,*length* AS *variable*,...]

Divides a direct access buffer into fields so that you can send data from memory to disk and from disk to memory. Each field is identified by a string *variable* and is the *length* you specify. *Length* must be an integer in the range 1 to 255.

```
FIELD 3, 128 AS A$, 128 AS B$
```

FILES [*pathname*]

Displays the names of the files and directories on a disk.

If you specify *pathname*, BASIC lists all files that match that pathname. If you omit the filename when specifying *pathname*, BASIC lists all files and directories in the specified directory. Default = all files and directories in the current directory on the current drive.

```
FILES                  FILES "\BOOKS\"
```

FIX(*number*)

Returns the truncated integer of *number*.

```
PRINT FIX(2.6)     Z=FIX(B)
```

FOR *variable* = *initial value* TO *final value* [STEP *increment*]
NEXT [*variable*]

Establishes a program loop that allows a series of program statements to be executed a specified number of times.

Variable must be either integer or single precision.

Increment is the number BASIC adds to *initial value* each time the loop is executed. Default=1.

```
FOR I = 1 TO 5:PRINT I:NEXT
```

FRE(*dummy argument*)

Returns the number of bytes in memory not being used by BASIC. If you specify a numeric argument, BASIC returns the amount of memory available. If you specify a string argument, BASIC compresses the data before returning the amount of memory available. BASIC automatically compresses data if it runs out of workspace.

```
PRINT FRE("44")    PRINT FRE(44)
```

GET [*#*]*buffer*[*,record*]

Reads a record from a direct access disk file and places it in the specified *buffer*. The number sign (#) is not required.

Record is an integer in the range Ø to 16,777,215. Default = the next sequential record (after the last GET).

```
GET 1              GET 1,25
```

GET [*#*]*buffer,number*

Communications. Transfers data from the communications line to the communications buffer. The number sign (#) is not required.

Number is the number of bytes to transfer.

```
GET 1,8
```

GET *(x1,y1)-(x2,y2),array*

Graphics. Transfers points from an area on the display to an array.

(x1,y1) are the coordinates at which the image begins.

(x2,y2) are the coordinates at which the image ends.

Array is a numeric array to hold the image.

```
GET (0,0)- (100, 100), Z
```

GOSUB *line*

Branches to the subroutine, beginning at *line*. Every subroutine must end with a RETURN statement.

```
GOSUB 1000
```

GOTO *line*

Branches to the specified *line*.

```
GOTO 100                IF R = 13 THEN GOTO 80
```

HEX$(*number*)

Returns a string that represents the hexadecimal value of *number*.

```
PRINT HEX$(30)     Y$ = HEX$(X/16)
```

IF *expression* **THEN** *statement(s)*[**ELSE** *statement(s)*]

Tests a conditional expression and makes a decision regarding program flow.

If *expression* is true, BASIC executes the THEN *statement*. If *expression* is false, BASIC executes the matching ELSE *statement* or the next program line.

```
IF A < B THEN PRINT "A < B"
ELSE PRINT "B <= A"
```

INKEY$

Reads a character in the keyboard buffer, and returns a string.

Ø-byte string	=	no key is pressed.
1-byte string	=	the ASCII value of the key.
2-byte string	=	the key has an extended code. Byte 1 = ØØ. Byte 2 = the ASCII value.

INKEY$ does not echo the character to the display.

```
10 A$ = INKEY$:IF A$ = "" THEN 10
```

INP(*port*)

Returns the byte read from *port*. *Port* may be any integer from Ø to 65535.

```
PRINT INP(255)       A=INP(255)
```

INPUT[;] [*"prompt";*]*variable*[*,variable,...*]

Accepts data from the keyboard and stores it in 1 or more variables. BASIC stops execution and displays *prompt* followed by a question mark to indicate that the program is waiting for input. If you do not want BASIC to display the question mark, type a comma, instead of a semicolon, after *prompt*.

If INPUT is immediately followed by a semicolon (;), BASIC does not echo the ENTER key when you press it as part of a response.

INPUT# *buffer, variable*[*,variable...*]

Accepts data from a sequential device or file and stores it in a program *variable*. *Buffer* is the number assigned to the file when you opened it.

```
INPUT#1, A,B       INPUT#4, A$, B$, C$
```

INPUT$(*number* [,[#]*buffer*])

Inputs a string of characters from either the keyboard or a sequential access file. *Number* specifies the number of characters to be input and may be in the range 1 to 255.

If you include *buffer*, BASIC inputs the string from a sequential access file. If you omit *buffer*, BASIC inputs the string from the keyboard. The number sign (#) is not required.

```
A$ = INPUT$(5)    A$ = INPUT$(11,3)
```

INSTR([*number*,]*string1*,*string2*)

Searches for the first occurrence of *string2* in *string1* and returns the position at which the match is found.

Number specifies the position in *string1* to begin searching for *string2* and must be an integer in the range 1 to 255. Default = first character in *string1*.

```
INSTR (3, "1232123", "12")
A$ = "LINCOLN":PRINT INSTR(A$,"INC")
```

INT(*number*)

Converts *number* to the largest integer that is less than or equal to *number*. *Number* is not limited to the integer range.

```
PRINT INT(79.89) PRINT INT(-12.11)
```

IOCTL [#]*buffer*,*string*

Advanced Statement. Sends a control data string to a device driver. *Buffer* is the number assigned to the driver when you opened it. The number sign (#) is not required.

String is a string expression containing a series of commands called "control data." The commands are generally 2 to 3 characters long and may be followed by an alphanumeric argument. The commands are separated by semicolons (;). *String* may be a maximum of 255 bytes.

```
IOCTL #1,"PL56"
```

IOCTL$([*#*]*buffer*)

Advanced Function. Returns the control data string from a device driver that you have opened previously. *Buffer* is the number assigned to the driver when you opened it. The number sign (#) is not required.

```
IF IOCTL$(1) = "NR" THEN PRINT
"PRINTER NOT READY"
```

KEY *number,string*
KEY *action*

Assigns or displays function key values. *Number* indicates the function key (1-12) or the user key (15-2Ø) being defined. See KEY (*number*) *action.*

String is the string expression assigned to the key and may contain a maximum of 15 characters.

Action can be ON, OFF, or LIST.

KEY ON

Displays the function key assignment values on Line 25 of the screen. BASIC shows only the first 5 characters of the string. CTRL T is the same as KEY ON.

KEY OFF

KEY OFF erases the soft key assignments from Line 25. The assignments are still active, but the screen does not display them.

KEY LIST

KEY LIST displays all 15 characters of all 12 soft key assignments on the screen.

KEY(*number*) *action*

Turns on, turns off, or temporarily halts key trapping for a specified key.

KEY() ON	enables key trapping
KEY() OFF	disables key trapping
KEY() STOP	temporarily suspends key trapping

Number may be a number in the range 1 to 20, indicating the number of the key to trap. Function keys use their corresponding function key number (1-10). The cursor direction keys are:

↑	11
←	12
→	13
↓	14

User-defined keys are 15-20. Use the following syntax to define your own user keys:

KEY *number*, CHR$(*key*)+CHR$(*scan*)

Key is one or a combination of the following:

&H40	CAPS lock key
&H20	NUM LOCK key
&H08	ALT key
&H04	CTRL key
&H02	Left SHIFT key
&H01	Right SHIFT key

Scan is the scan code for a physical key on the keyboard.

KILL *pathname*

Kills (deletes) *pathname* from disk.

```
KILL "file.bas"
KILL "A:\REPORT\data"
```

LCOPY

Copies all text data on the screen to the printer.

```
LCOPY
```

LEFT$(*string,number*)

Returns the specified number of characters from the left portion of *string*. *Number* must be in the range 1 to 255.

```
PRINT LEFT$("BATTLESHIPS", 6)
```

LEN(*string*)

Returns the number of characters in *string*. Blanks are counted.

```
PRINT LEN("DOG") + LEN("TERRIER")
X = LEN(SENTENCE$)
```

[LET] *variable* = *expression*

Assigns the value of *expression* to *variable*. BASIC does not require assignment statements to begin with LET.

```
LET A$ = "A ROSE IS A ROSE"
LET B1 = 1.23
```

LINE [[STEP](*x1,y1*)]-[STEP](*x2,y2*),[*color*][,B[F]] [,*style*]

Graphics. Draws a line or a box on the video display.

STEP designates (*x,y*) as relative coordinates.

(*x1,y1*) are the coordinates at which the line begins. Default = last point referenced on the screen.

(*x2,y2*) are the coordinates at which the line ends.

With the **B** option, BASIC draws a box. The points that you specify are opposite corners.

If you specify both the **B** and **F** options, BASIC draws a box and fills the box in with *color*.

Style is a 16-bit integer that lets you select the line-style used when drawing normal lines and unfilled boxes. Each bit represents a point in the line. If the bit equals 1, then the point is drawn. If the bit equals zero, then the point is not drawn.

```
LINE (0,0)-(319,199)
LINE -(319, 199),BF
```

LINE INPUT[;][*"prompt"*;] *string variable*

Accepts an entire line (a maximum of 255 characters) from the keyboard, including delimiters (commas, quotation marks, etc.). BASIC stops execution and displays *prompt* to indicate that the program is waiting for input.

The only way to terminate the string input is to press ENTER. However, if LINE INPUT is immediately followed by a semicolon, pressing ENTER does not echo a carriage return to the display.

```
LINE INPUT A$
LINE INPUT "LAST, FIRST NAME? "; N$
```

LINE INPUT#*buffer*, *variable*

Accepts an entire line of data from a sequential access file, including delimiters (commas, quotation marks, etc). *Buffer* is the number assigned to the file when you opened it.

```
LINE INPUT#1, A$
```

LIST *startline-endline* [,*device*]

Lists a program in memory to the display.

Startline specifies the first line to be listed. Default = first line in the program.

Endline specifies the last line to be listed. Default = last line in the program.

Device may be either SCRN: (screen) or LPT1: (printer). Default = screen (SCRN:).

```
LIST                LIST 50-100, LPT1:
```

LLIST *startline-endline*

Lists program lines in memory to the printer. LLIST assumes an 80-character-wide printer. You may change this by using the WIDTH statement with the LPRINT option. *Startline* and *endline* are described in LIST.

```
LLIST               LLIST 68-90
```

LOAD *pathname* [,R]

Loads a BASIC program from disk into memory. The R option tells BASIC to run the program.

```
LOAD "A:prog1.bas"
LOAD "prog1.bas",R
```

LOC(*buffer*)

Returns the current record position within a file. *Buffer* is the number assigned to the file when you opened it.

Direct access files: LOC returns the record number accessed by the last GET or PUT statement.

Sequential access files: LOC returns the number of 128-byte records that have been read or written.

```
A=LOC(2)              IF LOC(1)>55 THEN END
```

LOC(*buffer*)

Communications. Returns the number of characters in the input queue. *Buffer* is the number assigned to the file when you opened it.

If more than 255 characters are in the input queue, LOC always returns 255. If fewer are there, LOC returns the actual number of characters waiting to be read.

```
IF LOC(X)>Ø THEN 1ØØØ
```

LOCATE [*row*][,[*column*][,[*cursor*][,[*start*][,*stop*]]]]

Positions the cursor on the screen at the position indicated by *row* and *column*. *Row* is in the range 1 to 25. *Column* is in the range 1 to 4Ø or 1 to 8Ø, depending on the current screen width.

Cursor indicates whether the cursor is visible or invisible. 1 = visible and Ø = invisible.

Start is the first scan line of the cursor.

Stop is the last scan line of the cursor.

Start and *stop* can be in the range Ø to 7.

```
LOCATE 1Ø,2Ø,1,4 LOCATE 24,1,1,3
```

LOF(*buffer*)

Returns the length of the file in bytes. *Buffer* is the number assigned to the file when you opened it.

```
Y = LOF(5)
```

LOF(*buffer*)

Communications. Returns the amount of free space in the input queue. *Buffer* is the number assigned to the file when you opened it.

You can use LOF to determine when an input queue is getting full so that transmission is stopped.

```
IF LOF(X) < 2Ø GOTO 1ØØØ
```

LOG(*number*)

Returns the natural logarithm of *number*. *Number* must be greater than zero.

```
PRINT LOG(3.14159)
Z = 1Ø * LOG(P5/P1)
```

LPOS(*number*)

Returns the logical position of the print head within the printer's buffer. *Number* can be Ø or 1 to indicate LPT1:.

```
IF LPOS(X)>6Ø THEN LPRINT
```

LPRINT [**USING** *format*;] *data*[*,data,...*]

Prints *data* on the printer. LPRINT and LPRINT USING assume a print width of 8Ø characters. You may change the width by using the WIDTH statement with the LPRINT option.

See PRINT and PRINT USING for more information on formatting the output.

```
LPRINT (A * 2)/3
LPRINT USING "#####.#"; 2.17
```

LSET *field name = data*

Moves *data* to the direct access buffer and places it in *field name*, in preparation for a PUT statement. *Field name* is a string variable defined in a FIELD statement. You must have used FIELD to set up buffer fields before using LSET.

Any numeric value that is placed in a direct access file buffer with an LSET statement must be converted to a string. See MKS$, MKD$, and MKI$.

```
LSET AD$ = "2000 EAST PECAN ST."
LSET TD$=D$
```

MERGE *pathname*

Loads a BASIC program and merges it with the program currently in memory. Program lines in *pathname* are inserted into the resident program in sequential order. The file must be in ASCII format; that is, it must have been saved with the A option.

If line numbers in *pathname* coincide with line numbers in the resident program, *pathname*'s program lines replace the resident program's lines.

```
MERGE "prog2.txt"
```

MID$(*oldstring*,*start*[,*length*]) = *newstring*

Replaces a portion of *oldstring* with *newstring*.

Start specifies the position of the first character you want to change.

Length is the number of characters you want to replace.

```
A$="ABCDEFGH"
MID$=(A$,3,4)="WXYZ"
```

MID$(*string, start* [*,length*])

Returns a substring of *string.*

Length is the number of characters in the substring. It must be in the range 1 to 255.

Start specifies the position in the string from which to get the substring.

```
PRINT MID$("WEATHERFORD", 3, 2)
A$=MID$(T$,4,5)
```

MKDIR *dirpath*

Creates the directory specified by *dirpath.*

```
MKDIR "A:\ACCTS\PAYABLE"
MKDIR "\ADDRESS"
```

MKD$(*double-precision expression*)

Converts a numeric value to an 8-byte string value. This is the inverse function of CVD.

Any numeric value that is placed in a direct access file buffer by an LSET or RSET statement must be converted to a string.

```
LSET YTD$ = MKD$(564.33)
RSET DAY$=MKD$(DAY)
```

MKI$(*integer expression*)

Converts a numeric value to a 2-byte string value. This is the inverse function of CVI.

Any numeric value that is placed in a direct access file buffer by an LSET or RSET statement must be converted to a string.

```
LSET TOT$ = MKI$(TOT)
RSET QTY$=MKI$(NUM)
```

MKS$(*single-precision expression*)

Converts a numeric value to an 4-byte string value. This is the inverse function of CVS.

Any numeric value that is placed in a direct access file buffer by an LSET or RSET statement must be converted to a string.

```
LSET AVG$ = MKS$(0.123)
RSET MIX$=MKS$(A)
```

NAME *old filename* AS *new filename*

Renames *old filename* as *new filename*. You cannot change directory names.

```
NAME "file.bas" AS "file.old"
```

NEW

Deletes the program currently in memory and clears all variables.

```
NEW
```

NOISE *source,volume,duration*

Generates noise through a TV monitor's speaker (external speaker). You must execute a SOUND ON statement before using NOISE.

Source selects the type of noise and may be an integer in the range 0 to 7. 0-3 selects periodic noise and 4-7 selects white noise.

Volume is an integer in the range 0 to 15 where 0 is the quietest and 15 is the loudest. Default = 8.

Duration may be in the range 0 to 65536. A *duration* of 18.2 equals 1 second.

```
NOISE 0,15,20
```

OCT$(*number*)

Returns a string that represents the octal value of a decimal *number*.

```
PRINT OCT$(30)    S$=OCT$(90)
```

ON COM(*channel*) GOSUB *line*

Transfers program control to a subroutine beginning at *line* when activity occurs on the specified communications channel.

Channel specifies communications channel 1 or 2.

Line is the subroutine line at which execution begins when activity occurs on the communications channel. Specifying Line Ø turns off communications trapping.

```
ON COM(1) GOSUB 1ØØØ
```

ON ERROR GOTO *line*

Transfers control to *line* if an error occurs. You must execute an ON ERROR GOTO before the error occurs. Specifying Line Ø turns off error trapping.

```
ON ERROR GOTO 15ØØ
```

ON *n* GOSUB *line*[,*line*,...]

Looks at *n* and transfers program control to the subroutine indicated by the *n*th line listed.

If *n* equals 1, BASIC branches to the first line listed in the statement. If *n* equals 2, BASIC branches to the second line listed, and so on. *N* must in the range Ø to 255. If *n* is Ø or greater than the number of line numbers listed, BASIC continues with the next statement.

```
ON Y GOSUB 1ØØØ, 2ØØØ, 3ØØØ
```

ON *n* GOTO *line*[,*line*,...]

Looks at *n* and transfers program control to the *n*th line listed.

If *n* equals 1, BASIC branches to the first line listed. If *n* equals 2, BASIC branches to the second line listed, and so on. *N* must be in the range Ø to 255. If *n* is Ø or greater than the number of line numbers listed, BASIC continues with the next statement.

```
ON MI GOTO 15Ø, 16Ø, 17Ø, 15Ø, 18Ø
```

ON KEY(*number***) GOSUB** *line*

Transfers program control to a subroutine, beginning at *line* when you press the specified key.

Number indicates the number of the key to trap. Function keys are 1 to 1Ø. The cursor direction keys are numbered:

↑	11
←	12
→	13
↓	14

User keys are numbered 15 through 2Ø. User keys are defined with the KEY statement.

Specifying Line Ø turns off key trapping for the specified key.

```
ON KEY(13) GOSUB 5ØØ
```

ON PEN GOSUB *line*

Transfers program control to the subroutine at *line* when you activate the light pen. Specifying Line Ø turns off pen trapping.

```
ON PEN GOSUB 1ØØØ
```

ON PLAY(*number***) GOSUB** *line*

Transfers program control to the subroutine, beginning at *line*, when the number of notes in the background music buffer goes from *number* to *number* minus 1.

Number must be in the range 1 to 32.

Specifying Line Ø turns off play trapping.

```
ON PLAY(3Ø) GOSUB 2ØØ
```

ON STRIG(*number*) **GOSUB** *line*

Transfers program control to the subroutine at *line* when you press one of the joystick's buttons.

Number specifies the button pressed and is one of the following:

Ø	left joystick, button 1
2	right joystick, button 1
4	left joystick, button 2
6	right joystick, button 2

Specifying Line Ø turns off joystick trapping.

```
    10 ON STRIG(Ø) GOSUB 1ØØØ
```

ON TIMER(*number*) **GOSUB** *line*

Transfers program control to the subroutine, beginning at *line*, when the specified time has elapsed.

Number indicates the number of seconds. It may be a value in the range 1 to 864ØØ (864ØØ seconds = 24 hours).

```
    ON TIMER(36ØØ) GOSUB 5ØØ
```

OPEN *mode*,[#]*buffer*,[*pathname*][*dev:*][,*record length*]
OPEN [*pathname*][*dev:*] [FOR *mode*] AS [#]*buffer* [LEN=*record length*]

Establishes an input/output path for a file or device.

Buffer specifies the I/O buffer in memory to use when accessing the file and may be in the range 1 to 15. The number sign (#) is not required.

If you do not specify *pathname*, you must specify *dev:*.

Record length sets the record length for direct access files and may be in the range 1 to 32768. Default = 128 bytes.

Mode specifies any of the following:

O or OUTPUT	sequential output mode
I or INPUT	sequential input mode
A or APPEND	sequential output and extend mode
R or RANDOM	direct input/output mode

In the first form of the syntax, you must use the abbreviated form of *mode* and enclose it in quotation marks.

In the second form of the syntax, you must specify the complete word for *mode*. You may not specify RANDOM. If you want to use direct access in the second form of the syntax, omit *mode*.

```
OPEN "R",2,"TEST.DAT"
OPEN "LPT1:" FOR OUTPUT AS #2
```

OPEN "COM*channel*: [*speed*] [,*parity*] [,*data*][,*stop*][,RS] [,CS[*seconds*]][,DS[*seconds*]] [,CD[*seconds*]][,*mode*][,PE][,LF]" AS [#]*buffer* [LEN = *number*]

Opens a file and allocates a buffer for RS-232C (Asynchronous Communications Adapter) communication.

Channel specifies communications channel 1 or 2.

Speed specifies the baud rate and may be 75, 11Ø, 15Ø, 3ØØ, 6ØØ, 12ØØ, 24ØØ, 48ØØ, or 96ØØ. Default = 3ØØ.

Parity may be E for EVEN, O for ODD, M for MARK, S for SPACE, or N for NO. Default = E (EVEN).

Data specifies the number of bits and may be 5, 6, 7, or 8. Default = 7.

Stop may be either 1 or 2 to indicate the number of stop bits. Default = 2 for baud rates of 75 and 11Ø and 1 for all other baud rates.

Buffer indicates the buffer that accesses the file and may be in the range 1 to 15. The number sign (#) is not required.

Number specifies the maximum number of bytes that can be accessed in the communications buffer by GET and PUT statements. Default = 128 bytes.

RS suppresses the Request to Send signal. CS, DS, and CD control the following signals (in order): Clear to Send, Data Set Ready, Carrier Detect. *Seconds* specifies the number of milliseconds to wait before returning a Device Timeout error.

Mode can be BIN (for binary) or ASC (for ASCII). Default=BIN.

PE enables parity checking.

LF sends a line feed after every carriage return.

```
OPEN "COM1:" AS 1
OPEN "COM2:9600,N,8,1,BIN" AS 2
```

OPTION BASE *value*

Sets *value* as the minimum value for an array subscript. This statement must precede the DIM statement.

Value may be 1 or 0. Default = 0.

```
OPTION BASE 1
```

OUT *port, data byte*

Sends a *data byte* to a machine output *port.* A port is an input/output location in memory.

Port is an integer in the range 0 to 65535 and *data byte* is an integer in the range 0 to 255.

```
OUT 32,100
```

PAINT (*x,y*) [*color*[,*border*][,*background*]]

Graphics. Fills in an area on the display with a selected color or pattern.

(*x,y*) are the coordinates at which painting begins.

Color can be either a number or a string expression. If *color* is a number it specifies a color number available in the current screen mode. (See "Color and Graphics.")

If *color* is a string expression, it specifies the mask to be used for tiling in the form:

```
CHR$(&Hnn)+CHR$(&Hnn)+CHR$(&Hnn)...
```

Border is the color at which to stop painting.

Background is the color to skip when checking for borders while paint tiling. It is a 1-byte string expression.

PALETTE [*color,display color*]

Graphics. Changes the color associated with a particular color number in the current palette.

Color specifies the color in the current palette you want to change.

Display color specifies the new color you want BASIC to display when *color* is specified.

See "Color and Graphics."

```
PALETTE 3,7
```

PALETTE USING *array*(*subscript*)

Graphics. Changes the colors associated with more than 1 of the color numbers in the current palette.

Array is the name of an integer array in which you define the order in which colors are to be put in the current palette.

Subscript is the array position that contains the value of the color that you want put in the first palette position.

```
PALETTE USING A(Ø)    PALETTE USING A(2)
```

PCOPY *source page,destination page*

Copies the *source* video page to the *destination* video page.

```
PCOPY 3,5            PCOPY 6,4
```

PEEK(*memory location*)

Returns a byte from *memory location*. *Memory location* must be in the range -32768 to 65535. The value returned is an integer in the range Ø to 255.

```
A = PEEK (&H5AØØ)
```

PEN(*number*)

Returns the light pen's coordinates.

Number is a number in the range Ø to 9 that tells BASIC what to return.

Ø	Returns a -1 if pen button has been pressed since last poll. Returns a Ø if not.
1	Returns the x coordinate (horizontal) where the pen was last activated.
2	Returns the y coordinate (vertical) where the pen was last activated.
3	Returns a -1 if the pen button is pressed. Returns a Ø if it not.
4	Returns the last known valid x coordinate (horizontal).
5	Returns the last known valid y coordinate (vertical).
6	Returns the character row position where the pen was last activated.
7	Returns the character column position where the pen was last activated.
8	Returns the last known character row position.
9	Returns the last known character column position.

```
A = PEN(1)
```

PEN *action*

Turns on, turns off, or temporarily halts light pen event trapping.

PEN ON	enables event trapping.
PEN OFF	disables event trapping.
PEN STOP	temporarily suspends event trapping.

PLAY *string* [,[*string*][,*string*]]

Plays the musical notes specified by *string*.

PLAY supports 3 separate strings to allow independent control of each of 3 voices.

String is a string expression consisting of 1 or more single-character music commands.

Single character music commands:

A - G	plays notes A through G of 1 musical scale. You may include an optional number sign (#) or plus sign (+) to indicate a sharp note or a minus sign (–) to indicate a flat note.
L*n*	sets the duration of the notes that follow. *N* may be a value in the range 1 to 64 where: 1 indicates a whole note. 2 indicates a half note. 4 indicates a quarter note. 8 indicates an eighth note. 16 indicates a sixteenth note.
O*n*	sets the current octave. There are 7 octaves, Ø through 6. Octave 3 starts with middle C. Default = Octave 4.
>	changes the current octave to the next higher octave.
<	changes the current octave to the next lower octave.
N*n*	plays a note. *N* may be in the range Ø to 84.
P*n*	rests. *N* may be in the range 1 to 64.
T*n*	sets the number of quarter notes in 1 minute. *N* may be in the range of 32 to 255. Default = 12Ø quarter notes in 1 minute.
•	plays as a dotted note. BASIC plays the note one-half its length longer.
MF	plays the music in the foreground. Default = MF.
MB	plays the music in the background. A maximum of 32 notes and/or rests can play in background at a time. Default = MF.
MN	sets "music normal"; each note plays 7/8 of the duration as set by the L option. Default = MN.
ML	sets "music legato"; each note plays the full duration as set by the L option. Default = MN.
MS	sets "music staccato"; each note plays 3/4 of the duration as set by the L option. Default = MN.

X *variable*;	executes a substring. You can have 1 string execute another, which executes a third, and so on.
V*n*	sets the volume. *n* must be in the range Ø to 15. You must execute a SOUND ON statement to use this option. Default = 8.

The lowest note the multi-voice sound chip can produce is Note A of Octave Ø, which is 11Ø Hz. If you try to play a lower note, BASIC plays Note A of Octave Ø.

```
PLAY "C4F.C8F8.C16F8.G16A2F2"
```

PLAY(*number*)

Returns the number of notes currently in the background music queue.

Number is a dummy argument when SOUND is OFF. If you execute a SOUND ON, then *number* may be one of the following (Default = Ø):

Ø	returns the number of notes left to play on Voice Channel Ø.
1	returns the number of notes left to play on Voice Channel 1.
2	returns the number of notes left to play on Voice Channel 2.

```
X=PLAY(Ø)                    X=PLAY(2)
```

PLAY *action*

Turns on, turns off, or temporarily halts background music event trapping.

PLAY ON	enables play event trapping.
PLAY OFF	disables play event trapping.
PLAY STOP	temporarily suspends play event trapping.

PMAP(*coordinate,action*)

Returns the physical or world coordinate for the specified coordinate.

Coordinate is any x or y coordinate.

Action is one of the following:

Ø returns the physical x-coordinate for the specified world coordinate.
1 returns the physical y-coordinate for the specified world coordinate.
2 returns the world x-coordinate for the specified physical coordinate.
3 returns the world y-coordinate for the specified physical coordinate.

```
X=PMAP(2ØØ,Ø)                    Z=PMAP(5Ø,Ø)
```

POINT (*x*,*y*)
POINT (*action*)

Graphics. Returns the color number of a point on the screen or returns the current physical or world coordinates.

(*x*,*y*) are the coordinates of the point.

Action is one of the following:

Ø returns the current physical x-coordinate (horizontal).
1 returns the current physical y-coordinate (vertical).
2 returns the world x-coordinate if WINDOW is active. Otherwise, returns the physical x-coordinate.
3 returns the world y-coordinate if WINDOW is active. Otherwise, returns the physical y-coordinate.

```
IF POINT(1,1) <>Ø THEN PRESET (1,1)
ELSE PSET (1,1) X=POINT(1)
```

POKE *memory location, data byte*

Writes *data byte* into *memory location.*

Both *memory location* and *data byte* must be integers. *Memory location* must be in the range -32768 to 65535.

```
POKE &H5AØØ, &HFF
```

POS(*number*)

Returns the current column position of the cursor.

Number is a dummy argument.

```
IF POS(X) > 70 THEN IF A$ = CHR$(32)
THEN A$ = CHR$(13)
```

PRINT *data*[,*data*...]

Prints numeric or string *data* on the display. You can substitute a question mark (?) in place of the word PRINT.

If you use commas, the cursor automatically advances to the next tab position before printing the next item.

If you use semicolons or spaces to separate the data items, PRINT prints the items without any spaces between them.

```
PRINT "DO"; "NOT"; "LEAVE"; "SPACES"
PRINT "THE TOTAL IS",TTL
```

PRINT USING *format*; *data*[,*data*...]

Prints data using a format you specified. This statement is especially useful for printing report headings, accounting reports, checks, or any other documents that require a specific format.

Format consists of 1 or more field specifier(s), or any alphanumeric character. *Format* must be enclosed in quotation marks.

Data may be a string and/or numeric value(s).

Specifiers for String Fields:

!	prints only the first character in the string.
\spaces\	prints 2 + *n* characters from the string. (*N* is the number of spaces between the slashes.)
&	prints the string without modifications.

Specifiers for Numeric Fields:

#	prints the same number of digit positions as number signs (#). You may insert a decimal point at any position.
+	prints the sign of the number. The plus sign may be typed at the beginning or at the end of the format string.
-	prints a negative sign *after* negative numbers and a space after positive numbers.
**	fills leading spaces with asterisks.
$$	prints a dollar sign immediately before the number. You may not use exponential format with $$.
**$	fills leading spaces with asterisks and prints a dollar sign immediately before the number.
,	prints a comma before every third digit to the left of the decimal point.
^^^^	prints in exponential format. The 4 exponent signs are placed after the digit position characters. You may specify any decimal point position.
_	prints the next character as a literal character.

```
PRINT USING ".####^^^^"; 888888
PRINT USING "**$###,.##"; 1234.5
PRINT USING "###2.#-"; -768.660
PRINT USING "###.##"; 876.567
```

PRINT# *buffer*,[USING *format*] *data*[,*data*,...]

Writes data items to a sequential access file. PRINT# does not compress the data before writing it to disk. It writes an ASCII-coded image of the data.

See PRINT USING for information about the *format* parameter.

```
PRINT# 1,A          PRINT# 1, B$,T$
```

PSET [STEP] (*x*,*y*)[,*color*]
PRESET [STEP] (*x*,*y*)[,*color*]

Graphics. Draws a point on the display. If you use PSET, *color* defaults to the foreground color. If you use PRESET, *color* defaults to the background color.

(*x*,*y*) are the coordinates of the point. STEP designates (*x*,*y*) as relative coordinates.

```
PSET (1,1)          PRESET (1,1),0
```

PUT [#]*buffer*[,*record*]

Puts a *record* in a direct access file. The number sign (#) is not required.

Record is the number of the record to be written to the file and may be in the range 1 to 16,777,215. Default = current record number.

```
PUT 1               PUT 1,25
```

PUT [#]*buffer*,*number*

Communications. Transfers data from the communications buffer to the communications line. The number sign (#) is not required.

Number is the number of bytes to transfer.

```
PUT 2,80
```

PUT (*x*,*y*),*array*[,*action*]

Graphics. Transfers an image stored in an array to the screen.

(*x*,*y*) are the coordinates at which the image begins (the upper left corner of the image). Default = last point referenced.

Array is the array variable name that holds the image.

Action sets the type of interaction between the transferred image and the image already on the screen. *Action* may be PSET, PRESET, AND, OR, or XOR. Default = XOR.

RANDOMIZE [*number*]

Reseeds the random number generator.

Number may be an integer, or single- or double-precision number. If you omit *number*, BASIC suspends program execution and prompts you for a number before executing RANDOMIZE.

```
RANDOMIZE               RANDOMIZE 300
RANDOMIZE TIMER
```

READ *variable*[,*variable*,...]

Reads values from a DATA statement and assigns them to *variables*.

```
READ T                 READ N$, D$
```

REM

Inserts a remark line in a program. You may use an apostrophe (') as an abbreviation for REM.

```
REM AVERAGE VELOCITY          'TOTALS
```

RENUM [*new line*][,[*line*][,*increment*]]

Renumbers the program currently in memory. RENUM also changes all line number references appearing after GOTO, GOSUB, THEN, ON/GOTO, ON/GOSUB, ON ERROR GOTO, RESUME, and ERL.

Line is the line in the program at which BASIC starts renumbering. Default = first line.

New line is the new line number assigned to *line*. Default = Line 10.

Increment tells BASIC how to number the successive lines. Default = 10.

```
RENUM             RENUM 600, 5000, 100
```

RESET

Closes all open files on all drives.

```
RESET
```

RESTORE [*line*]

Restores a program's access to previously read DATA statements.

Line specifies the DATA statement to be accessed at the next READ statement. Default = first DATA statement.

```
RESTORE
```

RESUME [*line*]
RESUME NEXT

Resumes program execution after an error-handling routine.

RESUME *line* branches to the specified line number. Default = line in which the error occurred. RESUME NEXT branches to the statement following the point at which the error occurred.

```
RESUME                    RESUME 10        RESUME NEXT
```

RETURN [*line*]

Returns control from a subroutine executed by a GOSUB to the specified *line*. Default = line immediately following the GOSUB.

```
RETURN                    RETURN 40
```

RIGHT$(*string,number*)

Returns the specifed number of characters from the far right portion of *string*. *Number* must be an integer in the range 1 to 255.

```
PRINT RIGHT$("WATERMELON",5)
PRINT RIGHT$("PUPPY",25)
```

RMDIR *dirpath*

Removes (deletes) the directory specified by *dirpath*. The directory being deleted must be empty except for the "." and ".." symbols. Use the MS-DOS COPY command and/or the ERASE command to remove files from the directory.

```
RMDIR "NAMES"
RMDIR "A:\ACCTS\PAYABLE"
```

RND[(*number*)]

Returns a random number between 0 and 1.

If *number* is negative, RND starts the sequence of random numbers at the beginning. If *number* is 0, RND repeats the last number generated.

```
PRINT RND(1)        A = RND(0)
```

RSET *field name = data*

Sets *data* in a direct access buffer *field name* in preparation for a PUT statement.

```
RSET A$ = CVI(Z)
```

RUN [*line*]
RUN *pathname*[,R]

Executes a program. *Line* is the program line at which BASIC begins execution. Default = first line.

If you specify the R option, BASIC does not close the open files before loading the new program into memory. If you omit the R option, BASIC closes all open files before loading the program.

```
RUN                 RUN 100  RUN "program.a"
```

SAVE *pathname* [,A]
SAVE *pathname* [,P]

Saves a program on disk with the specified name.

The A option saves the program in ASCII format. Default = compressed format.

The P option saves the file in an encoded binary format. The only operations that can be performed on the file are RUN, LOAD, and CHAIN.

```
SAVE "A:file1.bas"
SAVE "\EDUC\mathpak.txt", A
```

SCREEN (*row, column,*[*number*])

Returns the ASCII code for the character at the specified row and column. *Row* is an integer in the range 1 to 25. *Column* is an integer in the range 1 to 4Ø or 1 to 8Ø, depending on the screen width.

Number is applicable only for text mode. If *number* is specified and is non-zero, BASIC returns the color number in the range 1 to 16 instead of the ASCII code of the character.

In the graphics modes, if the location does not contain a standard ASCII character, BASIC returns a value of zero.

```
A = SCREEN(2Ø,2Ø)
PRINT SCREEN(1Ø,1Ø,1)
```

SCREEN [*mode*][,[*burst*][,[*active page*]
[,*display page*]] [,*erase*]]

Sets the screen attributes to be used by all other graphics statements.

Mode is an integer in the range Ø to 6 that sets the valid coordinates and the number of colors you can use.

Burst enables or disables color. In Screen Mode Ø (text mode), set *burst* to Ø to disable color or 1 to enable color. In Screen Modes 1 and 4, set *burst* to Ø to enable color or 1 to disable color. *Burst* has no effect in Screen Modes 3, 5, and 6 where color is always enabled or in Screen Mode 2, which is black and white.

Active page selects the video page to which BASIC will write. All output statements to the screen go to the selected *active page*. Default = Page Ø or the current active page, which is initially Page Ø.

Display page selects the video page for BASIC to display. Default = *active page*.

Erase tells BASIC how much video memory to erase. *Erase* can be one of the following. Default = 1.

Ø	Do not erase video memory, even if the screen mode changes.
1	Erase the union of the new page and old page if *mode* or *burst* changes.
2	Erase all video memory if *mode* or *burst* changes.

```
SCREEN Ø,Ø          SCREEN 2
```

SGN(*number*)

Determines *number*'s sign. If *number* is a negative number, SGN returns -1. If *number* is a positive number, SGN returns 1. If *number* is zero, SGN returns Ø.

```
    PRINT SGN(-55)     Y = SGN(A * B)
```

SHELL [*command*]

Advanced Statement. Loads and executes another program (*.EXE* or *.COM*) as a child process to the original program. After the child process ends, control returns to the BASIC program at the statement following the SHELL statement.

Command is a string expression containing the name of the program you want to run.

```
    SHELL (ENTER)
```

SIN(*number*)

Returns the sine of *number*. *Number* must be in radians.

```
    PRINT SIN(7.96)    S=SIN(T)
```

SOUND *frequency,duration*[,[*volume*][,[*voice*]]]
SOUND ON
SOUND OFF

Generates a sound with the *frequency* and *duration* specified. While a SOUND statement is producing sound, the program continues to execute.

Frequency specifies the desired tone in Hertz. The lowest frequency that can be produced is 11Ø Hz. The frequency 32767 is treated as the silence frequency.

Note	Frequency	Note	Frequency
Middle C	523.25	G	783.99
D	587.33	A	88Ø.ØØ
E	659.26	B	987.77
F	698.46	C	1Ø46.5Ø

Duration is an integer in the range 1 to 65535, specifying the duration in clock ticks. Clock ticks occur 18.2 times per second.

Volume is an integer in the range Ø to 15, where Ø is the lowest volume and 15 is the highest volume. Default = 8.

Voice is an integer in the range Ø to 2. Default = Ø.

SOUND ON enables the external speaker that supports multivoice sounds using the PLAY or SOUND statements.

SOUND OFF disables the external speaker.

See also BEEP.

```
SOUND 2Ø, 5ØØ, 6
```

SPACE$(*number*)

Returns a string of *number* spaces. *Number* must be in the range Ø to 255.

```
PRINT "COST" SPACE$(4) "QUANTITY"
SPACE$(9) "TOTAL"
```

SPC(*number*)

Prints *number* blanks. *Number* is in the range Ø to 255.

```
PRINT "HELLO" SPC(15) "THERE"
```

SQR(*number*)

Returns the square root of *number*. *Number* must be greater than zero.

```
PRINT SQR(155.7)
```

STICK(*action*)

Returns the coordinates of the joysticks.

Action may be one of the following:

Ø	reads all 4 coordinates, and returns the horizontal (x) coordinate for left joystick.
1	returns the vertical (y) coordinate for left joystick.
2	returns the horizontal (x) coordinate for right joystick.
3	returns the vertical (y) coordinate for right joystick.

You must read Ø before reading 1, 2, and 3.

```
STICK (2)          STICK (Ø)
```

STOP

Stops program execution.

```
STOP
```

STR$(*number*)

Converts *number* to a string.

```
S$ = STR$(X)       PRINT STR$(-234)
```

STRIG ON
STRIG OFF

Enables the STRIG function.

STRIG ON lets you execute STRIG function statements to return the status of the joystick buttons.

If you execute a STRIG OFF statement, you cannot execute the STRIG function.

STRIG(*number*)

Returns the status of joystick buttons. (L refers to the left joystick and R to the right joystick.)

Number is a number in the range Ø to 7 to test the status of the joystick buttons.

Ø	Tests to see if Trigger L1 has been pressed and released since the last STRIG(Ø) function was executed. BASIC returns a -1 if it has been pressed and a Ø if not.
1	Tests to see if you are currently pressing Trigger L1. BASIC returns a -1 if you are pressing it and a Ø if not.
2	Tests to see if Trigger R1 has been pressed and released since the last STRIG(2) function was executed. BASIC returns a -1 if it has been pressed and a Ø if not.
3	Tests to see if you are currently pressing Trigger R1. BASIC returns a -1 if you are pressing it and a Ø if not.
4	Tests to see if Trigger L2 has been pressed and released since the last STRIG(4) function was executed. BASIC returns a -1 if it has been pressed and a Ø if not.
5	Tests to see if you are currently pressing Trigger L2. BASIC returns a -1 if you are pressing it and a Ø if not.
6	Tests to see if Trigger R2 has been pressed and released since the last STRIG(6) function was executed. BASIC returns a -1 if it has been pressed and a Ø if not.
7	Tests to see if you are currently pressing Trigger R2. BASIC returns a -1 if you are pressing it and a Ø if not.

```
A = STRIG(Ø)          Z = STRIG(4)
```

STRIG(*number*) *action*

Turns on, turns off, or temporarily halts joystick trapping.

STRIG ON	enables joystick trapping.
STRIG OFF	disables joystick trapping.
STRIG STOP	temporarily halts joystick trapping.

Number is a value of 0, 2, 4, or 6 to indicate the joystick button you are trapping (L = Left, R = Right):

0	indicates Trigger L1.
2	indicates Trigger R1.
4	indicates Trigger L2.
6	indicates Trigger R2.

```
STRIG(0) ON          STRIG(6) OFF
```

STRING$(*number,character*)

Returns a string containing the specified number of *character*. *Number* must be in the range 0 to 255.

Character is a string or an ASCII code.

```
B$ = STRING$(25, "X")
PRINT STRING$(50, 10)
```

SWAP *variable1,variable2*

Exchanges the values of 2 variables of the same type.

```
SWAP F1#, F2#
```

SYSTEM

Returns you to the MS-DOS command level.

```
SYSTEM
```

TAB(*number*)

Spaces to position *number* on the display.

Number must be in the range 1 to 255.

```
PRINT "NAME" TAB(25) "AMOUNT":PRINT
```

TAN(*number*)

Returns the tangent of *number*. *Number* must be in radians.

```
    PRINT TAN(7.96)      S = TAN(X)
```

TIME$[=*string*]

Sets or retrieves the current time. BASIC uses a 24-hour clock.

String is a literal, enclosed in quotation marks, that sets the time by assigning its value to TIME$. If you omit *string*, BASIC retrieves the current time.

```
    TIME$ ="14:15"       A$=TIME$
```

TIMER

Returns the number of seconds since midnight or since the last system reset. You can use TIMER as the argument for the RANDOMIZE statement to reseed the random number generator.

```
    PRINT TIMER          A = TIMER
```

TIMER *action*

Turns on, turns off, or temporarily halts timer event trapping.

TIMER ON	enables timer event trapping.
TIMER OFF	disables timer event trapping.
TIMER STOP	temporarily suspends timer event trapping.

TROFF
TRON

Turns the trace function on/off. The tracer lets you follow program flow. TRON turns on the tracer and TROFF turns it off.

```
    TRON                 TROFF
```

USR[*number*](*argument*)

Calls a user's assembly-language subroutine identified by *number* and passes *argument* to that subroutine.

The *number* you specify must be the same as the corresponding DEF USR statement for that routine. Default = Ø.

VAL(*string*)

Calculates the numerical value of *string.*

```
PRINT VAL("1ØØ")  PRINT VAL("1234E5")
```

VARPTR (*variable*)
VARPTR ([*#*]*buffer*)

Returns the offset into BASIC's data segment of a variable or a disk buffer.

When used with *variable,* VARPTR returns the address of the first byte of data identified with *variable.*

When used with *buffer,* VARPTR returns the address of the file's control block. The number sign (#) is not required.

```
PRINT VARPTR(3)   A = VARPTR(A$)
```

VARPTR$(*variable*)

Returns a 3-byte string representing a memory address of a variable:

Byte Ø	=	*type*
Byte 1	=	*low byte of address*
Byte 2	=	*high byte of address*

Type is 2 for integer variables, 3 for string variables, 4 for single precision variables, and 8 for double precision variables.

```
A$ = VARPTR$(A!)
```

VIEW [SCREEN] [(*x1,y1*)-(*x2,y2*)[,[*color*][,[*border*]]]]

Graphics. Creates a rectangular viewport that redefines the screen parameters. This defined area, a window, becomes the only place in which you can draw graphics displays.

(*x1,y1*) specifies the upper-left corner of the viewport.

(*x2,y2*) specifies the lower-right corner of the viewport.

SCREEN specifies that all coordinates used in drawing are absolute to point Ø,Ø on the screen. If you omit SCREEN, all coordinates specified are relative to the viewport coordinates.

```
VIEW (1Ø,1Ø)-(1ØØ,1ØØ)
VIEW SCREEN (2Ø,25)-(1ØØ,15Ø)
```

VIEW PRINT *top line* TO *bottom line*

Creates a text viewport that redefines the text screen parameters.

Top line specifies the first line of the text viewport. It may be in in the range 1 to 24, but must be less than *bottom line*. Default = Line 1.

Bottom line specifies the last line of the text viewport. It may be in the range 1 to 24, but must be greater than *top line*. Default = Line 24.

```
VIEW PRINT 1 TO 15
```

WAIT *port, number1* [,*number2*]

Suspends program execution until a machine input *port* develops a specified bit pattern. *Number1* and *number2* are integers in the range Ø to 255.

```
WAIT 32,2
```

WHILE *expression*
WEND

Executes a series of statements in a loop as long as a given condition is true.

If *expression* is true, BASIC executes the statements after the WHILE statement until it encounters a WEND statement. Then BASIC returns to the WHILE statement and checks *expression.* If it is still true, BASIC repeats the process. If it is not true, execution resumes with the statement following the WEND statement.

```
WHILE NUM             WEND
```

WIDTH [LPRINT] *size*
WIDTH *buffer, size*
WIDTH *device, size*

Sets the line width in number of characters for the display, printer, or communications channel.

Buffer is the number assigned to the file in the OPEN statement.

Device is a valid device, enclosed in quotation marks, that specifies the device for which you are setting the width. It may be SCRN:, LPT1:, COM1:, or COM2:.

Size may be an integer in the range Ø to 255 that specifies the number of characters in a line. For the screen, *size* may be 2Ø, 4Ø, or 8Ø.

```
WIDTH 4Ø              WIDTH LPRINT 1ØØ
WIDTH "SCRN:", 4Ø
```

WINDOW [SCREEN] [(*x1,y1*)-(*x2,y2*)]

Lets you change the physical coordinates of the screen (or current viewport) by defining "world coordinates."

(*x1,y1*) are the world coordinates for the upper-left corner of the screen.

(*x2,y2*) are the world coordinates for the lower-left corner of the screen.

The SCREEN option tells BASIC to set the coordinates similar to the screen display in that the lesser y-coordinate is in the upper-left corner of the screen. If you omit SCREEN, BASIC inverts the y-coordinates to show a true Cartesian coordinate system. That is, the lesser y-coordinate is in the lower-left corner of the screen.

WINDOW lets you plot points outside the normal screen coordinate limits by setting new world coordinates to the screen.

```
WINDOW (1984,100000)-(1987,300000)
```

WRITE *data[,data,...]*

Writes data to the screen.

```
WRITE D, B, V$
```

WRITE*#buffer, data[,data,...]*

Writes *data* to a sequential-access disk file.

Buffer is the number assigned to the file when you opened it.

```
WRITE#1, A$,B$
```

BASIC Error Codes and Messages

Error Number	Error Message
1	NEXT without FOR
2	Syntax error
3	Return without GOSUB
4	Out of DATA
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
1∅	Redimensioned Array/Duplicate Definition
11	Division by zero
12	Illegal direct
13	Type mismatch
14	Out of string space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefined user function
19	No RESUME
2∅	RESUME without error
21	Unprintable error
22	Missing operand
23	Line buffer overflow
24	Device Timeout
25	Device Fault
26	FOR without NEXT
27	Out of paper
29	WHILE without WEND
3∅	WEND without WHILE
5∅	FIELD overflow
51	Internal error

Error Number	**Error Message**
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O Error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file name
66	Direct statement in file
67	Too many files
68	Device Unavailable
69	Communication buffer overflow
7Ø	Disk write protected
71	Disk not Ready
72	Disk media error
73	Advanced Feature
74	Rename across disks
75	Path/file Access Error
76	Path not found
77	Deadlock

Keyboard ASCII and Scan Codes

Scan Code	Keyboard Legend	ASCII Codes Normal	SHIFT	CTRL	ALT	Remarks
Ø1	ESC	1B	1B	1B	x8B	
Ø2	! 1	31	21	xE1	x78	
Ø3	@ 2	32	4Ø	xØ3	x79	
Ø4	# 3	33	23	xE3	x7A	
Ø5	$ 4	34	24	xE4	x7B	
Ø6	% 5	35	25	xE5	x7C	
Ø7	^ 6	36	5E	1E	x7D	
Ø8	& 7	37	26	xE7	x7E	
Ø9	* 8	38	2A	xE8	x7F	
ØA	(9	39	28	xE9	x8Ø	
ØB	) Ø	3Ø	29	xEØ	x81	
ØC	_ –	2D	5F	1F	x82	
ØD	+ =	3D	2B	xF5	x83	
ØE	BACKSPACE	Ø8	Ø8	7F	x8C	
ØF	TAB	Ø9	xØF	x8D	x8E	
1Ø	Q	71	51	11	x1Ø	
11	W	77	57	17	x11	
12	E	65	45	Ø5	x12	
13	R	72	52	12	x13	
14	T	74	54	14	x14	
15	Y	79	59	19	x15	
16	U	75	55	15	x16	
17	I	69	49	Ø9	x17	
18	O	6F	4F	ØF	x18	
19	P	7Ø	5Ø	1Ø	x19	
1A	[{	5B	7B	1B	xEB	
1B	] }	5D	7D	1D	xFØ	
1C	ENTER	ØD	ØD	ØA	x8F	MAIN KEYBOARD
1D	CTRL	*	*	*	*	CONTROL MODE
1E	A	61	41	Ø1	x1E	
1F	S	73	53	13	x1F	

Scan	**Keyboard**	**ASCII Codes**				
Code	**Legend**	**Normal**	**SHIFT**	**CTRL**	**ALT**	**Remarks**
2Ø	D	64	44	Ø4	x2Ø	
21	F	66	46	Ø6	x21	
22	G	67	47	Ø7	x22	
23	H	68	48	Ø8	x23	
24	J	6A	4A	ØA	x24	
25	K	6B	4B	ØB	x25	
26	L	6C	4C	ØC	x26	
27	; :	3B	3A	xF6	xF8	
28	' "	27	22	xF7	xF1	
29	↑	x48	x85	x9Ø	x91	
2A	SHIFT	*	*	*	*	LEFT SHIFT
2B	←	x4B	x87	x73	x92	
2C	Z	7A	5A	1A	x2C	
2D	X	78	58	18	x2D	
2E	C	63	43	Ø3	x2E	
2F	V	76	56	16	x2F	
3Ø	B	62	42	Ø2	x3Ø	
31	N	6E	4E	ØE	x31	
32	M	6D	4D	ØD	x32	
33	, <	2C	3C	xF9	x89	
34	. >	2E	3E	xFA	x8A	
35	/ ?	2F	3F	xFB	xF2	
36	SHIFT	*	*	*	*	RIGHT SHIFT
37	PRINT	1Ø	*	x72	x46	SCREEN PRINT TOGGLE
38	ALT	*	*	*	*	ALTERNATE MODE
39	SPACEBAR	2Ø	2Ø	2Ø	x2Ø	
3A	CAPS	*	*	*	*	CAPS LOCK
3B	F1	x3B	x54	x5E	x68	
3C	F2	x3C	x55	x5F	x69	
3D	F3	x3D	x56	x6Ø	x6A	
3E	F4	x3E	x57	x61	x6B	
3F	F5	x3F	x58	x62	x6C	

Scan Code	Keyboard Legend	ASCII Codes Normal	SHIFT	CTRL	ALT	Remarks
4∅	F6	x4∅	x59	x63	x6D	
41	F7	x41	x5A	x64	x6E	
42	F8	x42	x5B	x65	x6F	
43	F9	x43	x5C	x66	x7∅	
44	F1∅	x44	x5D	x67	x71	
45	NUM LOCK	*	*	*	*	NUMBER LOCK
46	HOLD	*	*	*	*	FREEZE DISPLAY
47	7 \	37	5C	x93	*	
48	8 ~	38	7E	x94	*	
49	9 PG UP	39	x49	x84	*	
4A	↓	x5∅	x86	x96	x97	
4B	4 |	34	7C	x95	*	
4C	5	35	xF3	xFC	*	
4D	6	36	xF4	xFD	*	
4E	→	x4D	x88	74	xEA	
4F	1 END	31	x4F	x75	*	
50	2 ‘	32	6∅	x9A	*	
51	3 PG DN	33	x51	x76	*	
52	∅	3∅	x9B	x9C	*	
53	▪ DELETE	2D	x53	x9D	x9E	
54	BREAK	x∅∅	x∅∅	*	*	SCROLL LOCK BIT TOGGLE CONTROL BRK ROUTINE (INT 1BH)
55	+ INSERT	2B	52	x9F	xA∅	
56	▪	2E	xA1	xA4	xA5	NUMERIC KEYPAD
57	ENTER	∅D	∅D	∅A	x8F	NUMERIC KEYPAD
58	HOME	x47	x4A	x77	xA6	
59	F11	x98	xA2	xAC	xB6	
5A	F12	x99	xA3	xAD	xB7	

* No ASCII code is generated, but the special function described in the Remarks column is performed.

– No ASCII code is generated.

x Values preceded by "x" are extended ASCII codes (codes preceded by an ASCII NUL)

The (ALT) key provides a way to generate the ASCII codes of decimal numbers in the range 1 to 255. Hold down the (ALT) key while you type *on the numeric keypad* any decimal number in the range 1 to 255. When you release ALT, the ASCII code of the number typed is generated and displayed.

Note: When the NUM LOCK light is off, the Normal and SHIFT columns for these keys should be reversed.

ASCII Character Codes

ASCII Code	Character	Control Character
000	(null)	NUL
001	☺	SOH
002	☻	STX
003	♥	ETX
004	♦	EOT
005	♣	ENQ
006	♠	ACK
007	(beep)	BEL
008	◘	BS
009	(tab)	HT
010	(line feed)	LF
011	(home)	VT
012	(form feed)	FF
013	(carriage return)	CR
014	♫	SO
015	☼	SI
016	►	DLE
017	◄	DC1
018	↕	DC2
019	‼	DC3
020	¶	DC4
021	§	NAK
022	▬	SYN
023	↨	ETB
024	↑	CAN
025	↓	EM
026	→	SUB
027	←	ESC
028	(cursor right)	FS
029	(cursor left)	GS
030	(cursor up)	RS
031	(cursor down)	US

ASCII Code	Character	ASCII Code	Character
032	(space)	070	F
033	!	071	G
034	”	072	H
035	#	073	I
036	$	074	J
037	%	075	K
038	&	076	L
039	’	077	M
040	(	078	N
041	)	079	O
042	*	080	P
043	+	081	Q
044	,	082	R
045	-	083	S
046	.	084	T
047	/	085	U
048	0	086	V
049	1	087	W
050	2	088	X
051	3	089	Y
052	4	090	Z
053	5	091	[
054	6	092	\
055	7	093	]
056	8	094	∧
057	9	095	—
058	:	096	‘
059	;	097	a
060	<	098	b
061	=	099	c
062	>	100	d
063	?	101	e
064	@	102	f
065	A	103	g
066	B	104	h
067	C	105	i
068	D	106	j
069	E	107	k

ASCII Code	Character	ASCII Code	Character
108	l	146	Æ
109	m	147	ô
110	n	148	ö
111	o	149	ò
112	p	150	û
113	q	151	ù
114	r	152	ÿ
115	s	153	Ö
116	t	154	Ü
117	u	155	¢
118	v	156	£
119	w	157	¥
120	x	158	Pt
121	y	159	ƒ
122	z	160	á
123	{	161	í
124	¦	162	ó
125	}	163	ú
126	~	164	ñ
127	⌂	165	Ñ
128	Ç	166	ª
129	ü	167	º
130	é	168	¿
131	â	169	⌐
132	ä	170	¬
133	à	171	½
134	å	172	¼
135	ç	173	¡
136	ê	174	«
137	ë	175	»
138	è	176	░
139	ï	177	▒
140	î	178	▓
141	ì	179	│
142	Ä	180	┤
143	Å	181	╡
144	É	182	╢
145	æ	183	╖

ASCII Code	Character	ASCII Code	Character
184	╕	220	▄
185	╣	221	▌
186	║	222	▐
187	╗	223	▀
188	╝	224	α
189	╜	225	β
190	╛	226	Γ
191	┐	227	π
192	└	228	Σ
193	┴	229	σ
194	┬	230	μ
195	├	231	τ
196	─	232	Φ
197	┼	233	Θ
198	╞	234	Ω
199	╟	235	δ
200	╚	236	∞
201	╔	237	Ø
202	╩	238	ϵ
203	╦	239	∩
204	╠	240	≡
205	═	241	±
206	╬	242	≥
207	╧	243	≤
208	╨	244	⌠
209	╤	245	⌡
210	╥	246	÷
211	╙	247	≈
212	╘	248	°
213	╒	249	●
214	╓	250	•
215	╫	251	√
216	╪	252	η
217	┘	253	²
218	┌	254	■
219	█	255	(blank 'FF')

Notes

UNIT SPECIFICATIONS

System Unit

Processor:	8088
Size:	
Length:	354mm (13.9 in.)
Depth:	290mm (11.4 in.)
Height:	97mm (3.8 in.)

Weight:
3.71 Kg (81b 4 oz) With 1 Diskette Drive

Transformer:

Input	110 Vac 60 Hz
Output to system	Pin 1-17 Vac, Pin 2-GND, Pin 3-17 Vac

Environment:
Air Temperature
System ON—60 to 90 degreees F (15.6 to 32.3 degrees C)
System OFF—50 to 100 degrees F (10 to 43 degrees C)
Humidity
System ON—8% to 80%
System OFF—8% to 80%

Diskette Drive

Power:

Supply		
Voltage	+5Vdc Input	+12Vdc Input
Nominal	+5Vdc	+12Vdc
Ripple		
	+5Vdc Input	+12Vdc Input
0 to 50 kHz	100mV	100mV
Tolerance		
	+5Vdc Input	+12Vdc Input
Including Ripple	+/-5%	+/-5%
Standby Current		
	+5Vdc Input	+12Vdc Input
Nominal	600 mA	400 mA
Worst Case	700 mA	500 mA
Operating Current		
	+5Vdc Input	+12Vdc Input
Nominal	600 mA	900 mA
Worst Case	700 mA	2400 mA

Environment:

Temperature	
Operating	50 to 122 degrees F (10 to 44 degrees C)
Non-operating	-40 to 140 degrees F (-40 to 60 degrees C)
Relative Humidity	
Operating	20% to 80% (noncondensing)
Non-operating	5% to 95% (noncondensing)

RADIO SHACK

A Division of Tandy Corporation
U.S.A.: Fort Worth, Texas 76102
CANADA: Barrie, Ontario L4M 4W5

TANDY CORPORATION

AUSTRALIA
91 Kurrajong Avenue
Mount Druitt, N.S.W. 2770

BELGIUM
Parc Industriel
5140 Naninne (Namur)

U.K.
Bilston Road Wednesbury
West Midlands WS10 7JN

8749579-11/85-BCo

Printed in U.S.A.