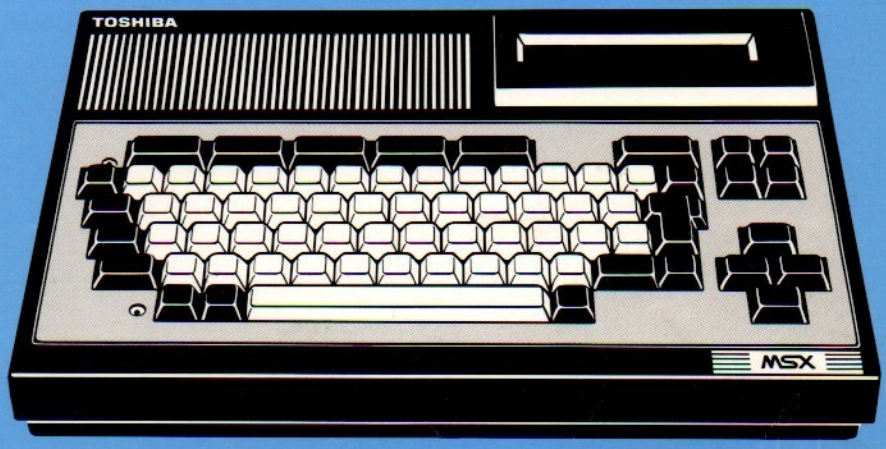
TOSHIBA

MSX

HOME COMPUTER
MSX BASIC REFERENCE MANUAL



TOSHIBA HOME COMPUTER MODEL HX-10
MSX BASIC REFERENCE MANUAL

NOTICE

1. The contents of this manual are subject to change without prior notice.

2. When using a special application program or computation procedure on the HX-10, it is advisable that the execution sequence, intermediate results, and final results be checked out carefully.

3. We are not responsible for any financial loss or lost profit which might result from the use of the computer.

PREFACE

This manual gives you detailed reference information for MSX BASIC, the programming language for the Toshiba HX-10 home computer. It is designed to let you look up the definitions and descriptions of MSX BASIC syntax, commands, statements and functions.

For operation details of the computer, refer to the "Toshiba Home Computer Owner's Manual."

For programming techniques or introductory information, read the books written about MSX BASIC.

CONTENTS

CHAPTER 1

SYNTAX

I INTRODUCTION TO MSX BASIC

1. WHAT IS MSX BASIC?

MSX BASIC is a powerful and versatile "standard" application language designed to offer software-level compatibility for different MSX micro-systems. It makes a wide variety of software packages written in MSX BASIC available to use on your HX-10 computer, and enables you to exchange software with your friends who have different MSX machines.
MSX BASIC is an extended version of Standard BASIC, version 4.5, developed by MICROSOFT CORPORATION.

2. BASIC LANGUAGE

The BASIC language consists of commands, statements, and functions. There is, however, no definite boundary between commands and statements.

- **Commands** . . . . . are used for controlling program entry, execution, editing, and management, chiefly in the Direct mode.
- **Statements** . . . . . are the basic elements of a BASIC program, and are used to tell the computer what it should do. Statements in a program are, as a rule, preceded by line numbers.
- **Functions** . . . . . allow you to do arithmetic operations or to manipulate numeric or character strings. They are used in statements.

3. BASIC Program

A BASIC program is a set of statements and/or commands arranged in the proper order to solve a specific problem or do a certain job. Your computer executes statements and commands in, as a rule, the ascending order of line numbers attached to them.

(1) Program line

Each program line consists of a line number and one, or more, statement or command. It is the simplest descriptive element for a BASIC program.

n n n n statement

(n n n n = line number)

- Up to 255 characters, including the line number, may be used to write a single program line.
- The line number must be an integer between 0 and 65529.

(2) Multi statements

Each program line consists, as a rule, of one statement or command. If you wish to lessen program lines, you can use more than one statement or command in a program line, by separating them with colons (:).

n n n n n statement : statement : statement

It is not possible, however, to continue a program line following the statements listed below:

REM, END, RETURN, GOTO

If specified, any statement or command, following these statements will be ignored.

4. EXECUTION MODES

(1) Command mode

When the computer is in the command mode (command wait state), it is ready to execute entered commands or statements or to input program lines.

The computer is placed in the command mode ("Ok" appears on the screen) whenever it is switched on or it has completed or stopped program execution.

(2) Direct mode

If you type in a command or statement with no line number attached to it when your computer is in the command mode the computer executes it immediately after you press the RETURN key. This type of execution is called the Direct mode.

(3) Program mode

If you type in statements or commands following line numbers, the computer assumes they are program lines and stores them in its internal memory for later execution.

The execution of a stored program can be initiated by using the RUN command or GOTO or GOSUB statement.

This type of execution is called the Program mode.

(4) Stopping or pausing execution

Simultaneously pressing the CTRL and STOP keys stops program execution in the Direct or Program mode, and returns the computer to the command mode.

Operating the STOP key during execution in Program mode will pause the execution (while program execution is being paused, only the STOP key remains operative). To restart program execution, press the STOP key a second time. If you simultaneously press the CTRL and STOP keys when program execution is being paused, the computer will return to the command mode.

5. CHARACTER SET USABLE IN BASIC

Characters usable in BASIC programs include uppercase alphabetic, lowercase alphabetic, alphasymbol, numeric, and graphic characters. For more details, refer to the "Character Code Table" in chapter 3, Section 1.
In addition to the meanings specified in statements, alphasymbol characters also have the following special meanings or functions:

+	(plus)	Arithmetic operator for addition, or the positive sign.
−	(minus)	Arithmetic operator for subtraction, the negative sign, or a range specification symbol (used in the LIST or other statements).
*	(asterisk)	Arithmetic operator for multiplication.
/	(slash)	Arithmetic operator for division.
\	(back slash)	Arithmetic operator representing integer quotient.
^	(hat)	Arithmetic operator for exponentiation.
=	(equal)	Relational operator for equal, or the assignment symbol.
>	(greater)	Relational operator representing greater than.
<	(less)	Relational operator representing less than.
(	(left parenthesis)	Symbol to help specify the priority order of arithmetic operations.
)	(right parenthesis)	Symbol to help specify the priority order of arithmetic operations.
%	(percent)	Variable type declaration symbol, for integer type.
!	(exclamation mark)	Variable type declaration symbol, for single-precision real type.
#	(sharp)	Variable type declaration symbol, for double-precision real type.
$	(dollar)	Variable type declaration symbol, for character type.
&	(ampersand)	Used in &H, &O, or &B, to represent a hex, octal, or binary constant, respectively.
.	(period)	The last execution line for BASIC (used in the AUTO, LIST, or LLIST statement), or the decimal point.
:	(colon)	Separator in multi-statement lines.
;	(semicolon)	Separator for variables or data in statements (PRINT statement, etc.)
,	(comma)	Separator for variables or data in statements (PRINT, DATA, or other statements).
?	(question mark)	Used in place of a PRINT statement.
'	(apostrophe)	Denotes a remark (used in place of a REM statement).
"	(quotation mark)	Character constant specification symbol.
	(blank)	Space

II PROGRAMMING

PROGRAMMING PROCEDURE

The following steps are usually required for MSX BASIC programming. For details on keyboard operation, refer to the "Toshiba Home Computer Owner's Manual."

1) Program entry
2) Program check
3) Program editing
4) Program execution
5) Error correction
6) Program saving

[Example]
The following program allows you to determine the product (C) of two numeric values A and B:

```
10 INPUT A
20 INPUT B
30 C = A * B
40 PRINT C
50 END
```

1. PROGRAM ENTRY

1) When entering a new program into your computer, clear the previous program, if any, from memory by executing the NEW command.
2) Type in the program line by line, with ascending line numbers preceding the program lines.
3) Be sure to press the RETURN key at the end of each line. (If the RETURN key is not pressed, the program line you have just typed in will not be stored in memory.)

[Example]
The key strokes required to enter the program given in the previous example are as follows:

```
[N][E][W][RETURN]
[1][0][ ][I][N][P][U][T][ ][A][RETURN]
[2][0][ ][I][N][P][U][T][ ][B][RETURN]
[3][0][ ][C][=][A][*][B][RETURN]
[4][0][ ][P][R][I][N][T][ ][C][RETURN]
[5][0][ ][E][N][D][RETURN]
```

The AUTO command lets the computer generate the automatic line numbering sequence if it is executed prior to program entry. It allows you to avoid having to type in a line number for each program line and to speed up programming.
The AUTO command may be executed just after executing the NEW command.

Type: [A][U][T][O][RETURN] ,
or [F2][RETURN]

To clear the AUTO command function, simultaneously press the CTRL and STOP keys.

2. PROGRAM CHECK

1) After completing program entry, use the LIST command to display all of the program lines on the screen, and verify that each line has been properly entered.
If you use a printer attached to the HX-10 computer, use the LLIST command to print a listing of the program on it.

To display program lines, type: [L][I][S][T][RETURN]
or [F4][RETURN]
To print program lines, type: [L][L][I][S][T][RETURN]

2) If part of the program list overflows the screen, you can specify a specific range of program lines you wish to display. For more details, refer to the LIST and LLIST command descriptions in Chapter 2, "LANGUAGE DESCRIPTION."

[Example]
Press [F4][RETURN] to list the program lines on the screen, and verify correct program entry.

Screen:

```
LIST
10  INPUT A
20  INPUT B
30  C=A*B
40  PRINT C
50  END
Ok
■
```

3. PROGRAM EDITING

Any portion of any program line listed on the screen can be edited by positioning the cursor to the character location to be edited. This screen edit capability is implemented by the screen editor.

1) Use the LIST command to display the portion of your program where the program line to be edited exists.
2) Place the cursor on the position where the character to be edited is located.
3) Execute the desired edit operation, such as delete, insert, or amend.
4) With the cursor left positioned on the line just edited, press the RETURN key.

Note 1: An edit operation is completed only if the RETURN key is pressed on the line just edited.

Note 2: If you change a line number and press the RETURN key, the new line number is added to the program, but the old line number and the characters following it will remain in memory as they are.

Purpose	Edit operation	Edit Example
Correcting a character on a displayed line. (Example) Changing PRIMT to PRINT.	1) Use the cursor control keys ([⇨] [⇦] [⇧] [⇩]) to position the cursor on the character to be corrected. 2) Type the correct character. 3) Press the [RETURN] key.	PRIMT■ You wish to replace "M" with "N" (■ denotes the cursor.) PRI■T The cursor is positioned on the "M" with the [⇦] key. PRIN■ The correct character (N) is typed. The cursor shifts to the next location.
Deleting a character from a displayed line. (Example) Deleting X from PRIXNT to obtain PRINT.	1) Use the cursor control keys to position the cursor on the character to be deleted. 2) Press the [DEL] key. 3) Press the [RETURN] key.	PRIXNT■ You wish to delete the X. PRI■NT Position the cursor on the X. PRI■T Press the [DEL] key.

Purpose	Edit operation	Edit Example
		PRNT■ The letter I is missing.
Inserting a character into a displayed line.	1) Use the cursor control key to position the cursor to the character location where a character is to be inserted.	PR■T Position the cursor to the insertion location using ← key.
(Example) Changing PRNT to PRINT by inserting an I.	2) Press the INS key.	PRNT Press INS to select the Insert mode.
	3) Type the character to be inserted.	PRINT Type in the letter I from the keyboard.
	4) Press the INS key again.	PRI■T Press INS again to clear the Insert mode.
	5) Press the RETURN key.	
Amending an entire program line.	1) Type in the correct program line using the same line number as the program line to be corrected. 2) Press the RETURN key. The entire program line is replaced with the correct one.	Type in the correct program line, such as 40 PRINT C RETURN.
Deleting an entire program line.	1) Type in the line number of the line to be deleted. 2) Press the RETURN key.	Type, for example, 10 RETURN to delete the line numbered 10.
Deleting more than one program line.	1) Execute the DELETE command.	To delete the program lines between line numbers 20 and 60, type DELETE 20 – 60 RETURN.
Inserting a new line between two existing lines (with line numbers n and m).	1) Type in the program line to be inserted, with a line number between (but not equal to) n and m preceding it.	
Renumbering program lines.	1) Execute the RENUM command.	To renumber the program lines in an increment of 10 beginning with 100, type RENUM 100, 10 RETURN.

4. PROGRAM EXECUTION

1) After completing program editing, use the cursor control keys ([→] [←] [↑] [↓]) to position the cursor on a row where no program line exists. Or simultaneously press the SHIFT and HOME keys to erase the screen. These keys only erase the screen; they don't clear the program you just entered from memory.

2) Use the RUN command to start program execution.

   Type: [R] [U] [N] [RETURN] ,

   or [F5]

3) To stop program execution, simultaneously press the CTRL and STOP keys.

[Example]
Try to determine the product of two numbers A and B, assuming A=5 and B=4

Display	Key operation
(Program to determine the product of two numbers)	Simultaneously press [SHIFT] and [HOME]
■ **(CURSOR)**	Press [F5].
RUN ? ■	Press [5] then [RETURN].
? **5** ? ■	Press then [RETURN].
? **4** **20** **Ok**	(The product of A and B appears on the screen.)

When execution of the program in the previous example is completed, the screen will appear as follows:

```
RUN
? 5
? 4
 20
Ok
■
```

5. ERROR CORRECTION

• Error in direct mode

1) If an error occurs while in the Direct mode, the computer will return to the Command mode, with the pertinent error message showing on the screen.

   Example: Syntax error
   (Error message)

2) Most errors occurring in the Direct mode are due to a command entry error.
3) Try the correct command entry again.

• Error in program mode

1) If an error occurs while in the Program mode, program execution will be suspended and the computer will return to the Command mode, with the pertinent error message appearing on the screen, along with the line number where the error occurred.

   Example: Syntax error in 10
   (Error message) (Line number)

2) Trace the location of the error by using the error message as a guide. For error message meanings, refer to the "Error Code Table" in Chapter 3, Section 6.
3) After locating the error, correct it by referring to Section 3 (program editing).

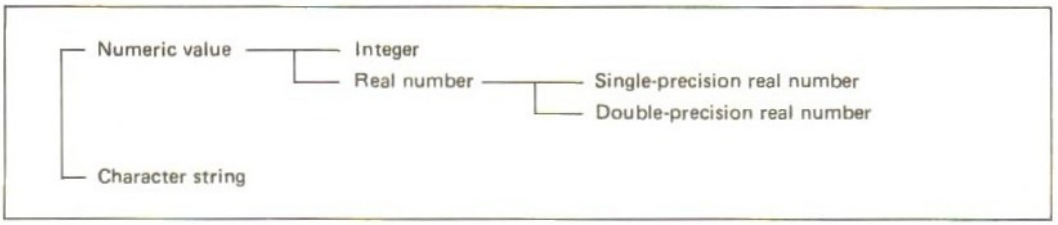
6. PROGRAM SAVING

1) Save your program on a cassette tape by using the CSAVE or SAVE command before turning your computer off. Once the computer is turned off, the program no longer exists in the internal memory.
2) Programs saved on a cassette tape can be loaded into your computer by using the CLOAD, LOAD, or MERGE command.
3) For details on connecting your cassette recorder to the computer, or on the program saving procedure, refer to the "Toshiba Home Computer Owner's Manual."

III CONSTANTS AND VARIABLES

1. DATA TYPES

BASIC handles both numeric values and characters as data. Numeric values are classified into the following type:



- **Numeric values**

  Numeric values usable in BASIC programs are positive or negative integers, real numbers, and zero.

  - **Integers**

    Between −32768 and +32767

  - **Real numbers**

    Include single- and double-precision real numbers:

    Single precision real number:

    A real number with six significant figures, ranging between −9.99999E +62 and +9.99999E+62. The valid range for the exponent is between E+62 and E−64. The minimum positive value expressible with a single-precision real number is, therefore, 1.0E−64.

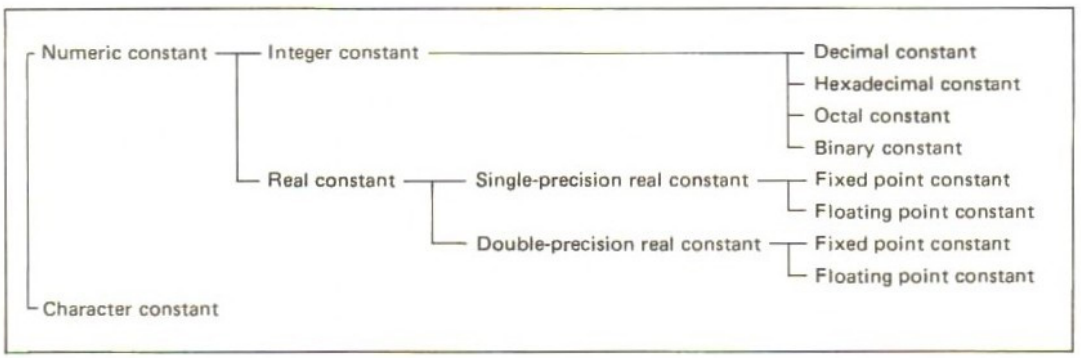
    Double precision real number:

    A real number with 14 significant figures, ranging between −9.9999999999999+62 and +9.9999999999999E+62. The valid range of the exponent is between E+62 and E−64. The minimum positive value expressible with a double-precision real number is, therefore, 1.0E−64.

- **Character string:** A string of up to 255 characters.

2. CONSTANTS

Constants are fixed or invariable values or data items used in programs. They include the following types:



(1) NUMERIC CONSTANTS

A negative numeric constant must always be prefixed with a negative sign (−). A positive numeric constant, however, does not have to be prefixed with a positive sign (+).

1) Integer constant

Integer constants may be expressed in decimal, hexa-decimal, octal, or binary notation.

Decimal constant:

- A decimal integer value ranging between −32768 and +32767.
- An integer or real number between −32768 and +32767, suffixed with " % ". For real numbers, all decimal places are truncated.

(Example) 123 −567 12.7%

Hexadecimal constant:

- A hexadecimal number is expressed with the characters 0 ~ 9 and A ~ F, prefixed with a &H, and ranges between &H0 and &HFFFF. Capital letters A through F correspond to 10 through 15 in decimal notation.
- Hex &H0 through &H7FFF are 0 through 32767 in decimal; hex &H8000 through &HFFFF are −32768 through −1 in decimal.

(Example) &HFF: 255 in decimal
&HFFFE: −2 in decimal

Octal constant:

- An octal number is expressed with the characters 0 through 7, prefixed with a &O, and ranges between &O0 and &O177777.
- Octal &O0 through &O77777 are 0 through 32767 in decimal; octal &O100000 through &O177777 are −32768 through −1 in decimal.

(Example) &O377: 255 in decimal
&O177776: −2 in decimal

Binary constant:

- A binary number is expressed by zeros and ones, prefixed with a &B, and ranges between &B0 and &B1111111111111111 (16 ones).
- Binary &B0 through &B111111111111111 (15 ones) are 0 through 32767 in decimal; &B1000000000000000 through &B1111111111111111 (16 ones) are −32768 through −1 in decimal.

(Example) &B11111111: 255 in decimal
&B1111111111111110: −2 in decimal

2) Single-precision real constant

Fixed-point real constant:

- A real number having six or less significant figures. A real number with more than six significant figures is a double-precision real number.
- A real number or integer suffixed with an exclamation mark (!). If it has seven or more significant figures, the 7th significant figure is rounded to the nearest whole number.

(Example) 9.87 0.0000345
44.44986869! (the 7th significant figure, 6, is rounded, resulting in 44.4499.)

Floating-point real constant:

- Consists of a mantissa having six or less significant figures and an exponent represented by using an E. The valid range is from −9.99999E+62 to +9.99999E+62, and the valid range of the exponent is from +62 to −64. A floating-point value with more than six significant figures in its mantissa is a double-precision floating point-real constant.

(Example) 1.234E (Exponent) +23 (Mantissa)

3) Double-precision real constant

Fixed-point real constant:

- A real number with 7 to 14 significant figures. If a real number contains 15 or more significant figures, the 15th significant figure is rounded to the nearest whole number; the resulting real number contains 14 significant figures.
- A real number or integer suffixed with a sharp (#). If a real number contains 15 or more significant figures, the 15th significant figure is rounded to the nearest whole number; the resulting real number has 14 significant figures.

(Example) 9.873333345
2344555555500
44.449#

Floating-point real constant:

- Consists of a mantissa having 7 to 14 significant figures and an exponent represented by using an E. The available range is between −9.9999999999999E +62 and +9.9999999999999E+62. The valid range of the exponent is between +62 and −64.

(Example) 1.2345678E (Exponent) +23 (Mantissa)

- When the exponent is represented by a "D" instead of an "E", the floating point real constant is a double-precision real number, even if its mantissa has 6 or less significant figures. The valid range is between −9.9999999999999D +62 and +9.9999999999999D+62. The valid range of the exponent is between +62 and −64.

(Example) 1.23D (Exponent) +23 (Mantissa)

(2) CHARACTER CONSTANT

A character constant consists of a string of not more than 255 characters enclosed in quotation marks (" ").

- A pair of quotation marks (" ") with nothing enclosed between them is called a blank constant, and is treated as one of the character constants.
- A pair of quotation marks (" ") with one or more spaces enclosed between them represents a blank character constant, and is discriminated from a blank constant.
- A numeric string enclosed in quotation marks, such as "123", is treated not as a numeric value but as a character constant. So arithmetic operations are not possible with it.

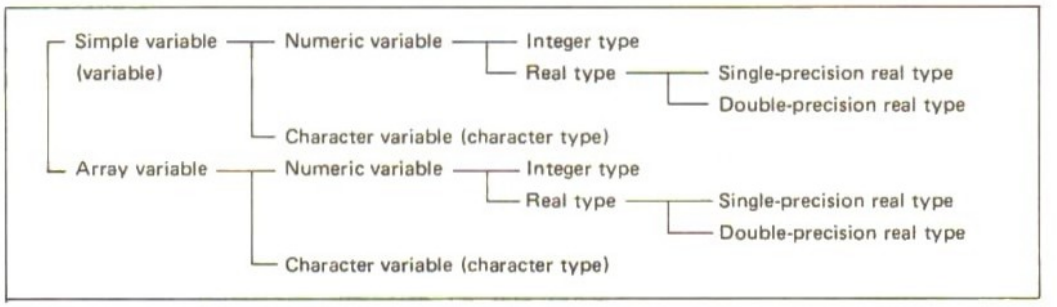
(Example) "PASOPIA"

3. VARIABLE

A variable is an assigned memory location used to store a data value. It has its own variable name composed of a character or group of characters that refers to the data which is variable.
You can assign a value to a variable by executing an assignment (LET), INPUT, or READ statement (a location for the variable is set aside in memory, and the value is transferred to the variable).
If a variable is referred to before a value is assigned to it, zero will be returned for a numeric variable, and a blank constant will be returned for a character variable.

(1) VARIABLE TYPE

Variables are classified into the following types depending on the data which is assigned to the variable. Only a specific data type can be assigned to a specific variable type.
A variable to which only one value can be assigned is called a simple variable, or, simply, a variable. A variable to which more than one value can be assigned is called an array variable.



(2) VARIABLE NAME

A variable name is specified with one or two alphanumeric characters and one type declaration symbol. The following gives some rules on variable naming.

1) The first character of a variable name must always be an alphabetic character. The second character may be either an alphabetic or numeric character.

   Example: A3=2 Value 2 is assigned to variable A3.
   3A=2 3A is not regarded as a variable.

2) If more than two characters are used to name a variable, only the first two characters are valid, with the third and subsequent characters ignored.

   Example: A3BC=2 Value 2 is assigned to variable A3.

3) All graphic symbols or blanks used in a variable name are ignored.

   Example: A 3=2 Value 2 is assigned to variable A3.
   A 3=2 Value 2 is assigned to variable A3.

4) Lowercase alphabetic characters used to name a variable are all converted into the corresponding uppercase alphabetic characters in the variable name.

   Example: a3=2 Value 2 is assigned to variable A3.

5) No reserved word (such as command, statement, function or operator names reserved for BASIC) can be used or contained in variable names. For details on reserved words, refer to the "Reserved Word List" in Chapter 3, Section 2.

   Example: AUTO3=2 The variable name "AUTO3" is not usable as it contains a reserved word "AUTO".

6) Suffix a variable name with a type declaration symbol to declare the variable type.

   % for integer type variables.
   ! for single-precision real-type variables.
   # or no symbol for double-precision real-type variables.
   $ for character variables.

- Variables with the same variable name but different type symbols are regarded by the computer as different variables.

  Example: A%, A! , and A are all different variables.

- When a variable name has no type symbol suffix, it is regarded by the system as a double-precision, real-type variable. A variable whose type is declared in a program by a DEFINT (define integer), DEFSNG (define single), DEFDBL (define double), or DEFSTR (define string) statement is regarded to have the respective variable type which is declared by those statements.

  Example: A# is the same variable as A. If DEFINT A is specified in a program, A% is the same as A.

(3) ARRAY VARIABLE

Sets of data can be handled more easily by using subscripted variables and arrays. We call a set of subscripted variables with the same name an array, and the individual variables array variables (elements).

Format:		
	Variable name (subscript)	One dimensional array
	Variable name (subscript, subscript)	Two dimensional array
	Variable name (subscript, subscript, subscript · · ·)	n dimensional array
	(n numbers)	

1) The dimension of an array is represented by the number of subscripts enclosed in parentheses (and separated by commas) following a variable name.
   Up to 255 dimensions are available.
2) Subscripts may range from zero to the maximum number of memory locations available.
3) The size (number of elements) and dimension of an array are specified with a DIMENSION (DIM) statement.

   Example: 10 DIM A (5,2)
   This DIM statement declares a two-dimensional array named A, in which up to (5+1) x (2+1)=18 subscripted variables can be used.

   - Array with up to three-dimensions can be used without declaring them in a DIM statement. At this time, memory locations for eleven variables (0 to 10) are automatically set aside for each dimension.

(4) MEMORY LOCATIONS (CAPACITY) AVAILABLE TO EACH VARIABLE TYPE

Variable	Simple variable	Array variable
Integer type	5	5+2*(number of elements) + 2*(dimension) + 1
Single-precision real type	7	5+4*(number of elements) + 2*(dimension) + 1
Double-precision real type	11	5+8*(number of elements) + 2*(dimension) + 1
Character string	6 + (number of characters in the string)	5+3*(number of elements) + 2*(dimension) + 1 + (total number of characters in strings contained in the elements)

(5) SYSTEM VARIABLE

MSX BASIC has the following system variables reserved for itself:

TIME:	This variable refers to the initial value of an interval timer which increments by one at 1/50 second intervals. The timer can be preset by assigning the desired value to this variable.
SPRITE$(n):	This character array refers to sprite patterns.
VDP (n):	This array refers to the register value in the VDP.

(6) TYPE CONVERSION

One numeric data type may be converted into another numeric data type as needed (conversion between numeric and character data is accomplished by the STR$ or VAL function).

1) If a numeric value of a certain type is transferred to a variable of another numeric type, a value converted according to the variable type is assigned to the variable.

   Example: A%=1.234 Integer 1 is assigned to variable A%.

2) For logical operations, all values are converted to integer type, and the results are also obtained in integer type.

   Example: A=NOT 1.234 Real value 1.234 is converted into integer 1, and the result of the NOT operation on 1 is assigned to variable A.

- When a real value is converted to an integer, all decimal places are trancated. If the result of rounding exceeds the valid range of integers (−32768 to +32767), an error will occur.
- When a double-precision real number is converted to a single-precision real number, the 7th significant figure is rounded to the nearest whole number, resulting in a 6-digit real number.

Ⅳ OPERATIONS

1. EXPRESSION

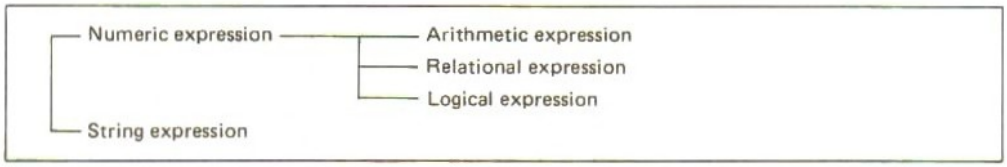
An expression is a combination of constants, variables, and/or functions connected by operators. A single constant, variable, or function with no operator may also be called an expression.

1) Numeric expressions and string expressions

The result of an operation specified by an expression is either a numeric value or a character string. Expressions which produce numeric results are called numeric expressions, while those which produce character strings are called string expressions.

2) Numeric expression types

Numeric expressions are classified further into arithmetic, relational, and logical expressions. These types of expressions all produce numeric results.



2. ARITHMETIC EXPRESSION

An arithmetic expression consists of one or more numeric constants, numeric variables, numeric functions, and/or numeric expressions connected by arithmetic operators. It always produces a numeric result.

Arithmetic operator	Operation	Entry format
+	Addition	X + Y
−	Subtraction	X − Y
*	Multiplication	X * Y
/	Division	X / Y
^	Power	X ^ Y
−	Negative sign	− X
\	Integer division	X \ Y
MOD	Remainder	X MOD Y

(1) Integer division

- For integer division (\ ), all real numbers are converted to integers, by rounding the first decimal place to the nearest whole number, before division is done.
- The quotient of integer division is an integer, with all decimal places truncated.

  Example: A=11.24\3 Integer 3 is assigned to variable A.
- The remainder of integer division is an integer and is obtained by use of the MOD operator.

  Example: A=11.24 MOD 3 Integer 2 is assigned to variable A.

(2) Division by zero

If division by zero is attempted, an error will occur. The error message "Division by zero" appears on the screen, and the computer returns to the command mode.

(3) Power of zero

The power of zero results, as a rule, in zero.
The zero'th power of zero (0^0) is one (1).
The negative number'th power of zero results in an error. The error message "Division by zero" appears on the screen, and the computer returns to the command mode.

(4) Overflow

If the result of an assignment or arithmetic operation exceeds the valid numeric range of the variable to which the result is transferred, an overflow will occur. The message "Overflow" apperas on the screen, and the computer returns to the command mode.

3. RELATIONAL EXPRESSION

A relational expression consists of numeric values or character strings connected by relational operators. It always produces a numeric result: −1 for true, and 0 for false.
Relational expressions are chiefly used for comparing two data values in the IF statement.

Relation operator	Meaning	Entry format
=	Equal	X = Y
< > or > <	Not equal	X < > Y
<	X is smaller than Y	X < Y
>	X is larger than Y	X > Y
< = or = <	X is equal to or smaller than Y	X < = Y
> = or = >	X is equal to or larger than Y	X > = Y

Examples: IF X=Y THEN 100 ELSE 200
If X equals Y, control is passed to the line numbered 100. Otherwise, it is passed to the line numbered 200.

A=X=Y
The first equal (=) sign is an assignment symbol, while the second equal sign is a relational operator. If X equals Y, the value −1 is assigned to variable A. If X does not equal Y, then the value 0 is assigned to A.

(1) Comparison of character strings

In a relational expression, two character strings are compared character-by-character, beginning with the first character in both strings. Two character strings are equal when every character in one string is identical to its counterpart in the other string. If two character strings are not equal, the one containing the larger character code is identified as the larger. If two character strings have different lengths, the longer string is identified as the larger. Blanks contained in strings are counted in calculating the lengths of strings.

Examples:
"ABCDEF" equals "ABCDEF"
"AA" is smaller than "AB"
"ABCDEF" is larger than "ABCDE"
"AA " is larger than "AA "
"A A" is smaller than "AA "

(2) Comparison of a numeric value with a character string is not possible. Numeric values are always compared with numeric values, and character strings are always compared with character strings.

4. LOGICAL EXPRESSION

A logical expression can contain one or more numeric constants, numeric variables, numeric functions and/or numeric expressions connected by logical operators. It always produces an integer result.
Logical expressions are used to compare more than one relational expression, chiefly in an IF statement, or to perform bit manipulation or Boolean operations.

Logical operator	Meaning	Entry format
NOT	Negation (not)	NOT X
AND	Logical product (and)	X AND Y
OR	Logical sum (or)	X OR Y
XOR	Exclusive or	X XOR Y
IMP	Implication	X IMP Y
EQV	Equivalence	X EQV Y

Example: 10 IF X > 10 AND X < 100 THEN 100
If the value of variable X is more than 10 and less than 100, control is passed to the line numbered 100.

10 IF X > 10 OR Y < 100 THEN 100
If the value of variable X is more than 10 or the value of variable Y is less than 100, control is passed to the line numbered 100.

(1) Truth tables for logical operations

X	NOT X
1	0
0	1

X	Y	X AND Y	X OR Y	X XOR Y	X IMP Y	X EQV Y
1	1	1	1	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	1	0
0	0	0	0	0	1	1

(2) Logical operation

Every logical operation is performed after all numeric values between −32768 and 32767 are converted into their two's complement. If a logical operation is attempted on a value outside this range, an error will occur.
A logical operation is performed on each bit of the operands.

Example: A=7 OR 8
The two's complement of values 7 and 8 are &B111 and &B1000, respectively. The result of a logical OR operation on these values is &B1111 (15 in decimal). So value 15 is assigned to variable A.

5. STRING EXPRESSION

A string expression consists of more than one character string linked with one or more plus (+) sign. The result of a string expression is always a character string.

Example: A$="ABC" : B$="DE" : C$=A$+B$

The character string ABCDE is assigned to variable C$.

6. FUNCTION

MSX BASIC provides a set of previously programmed functions that simplify writing many kinds of programs. They return the results of special functional operations (e.g. square root, absolute value, trigonometric functions, etc.) for given data (arguments). Functions may also be used for numeric or character string manipulation.

(1) Numeric functions and string functions

Functions are classified into numeric functions and string functions depending on whether they return numeric values or character strings.

- Numeric function
- String function

(2) Built-in functions and user-defined functions

In addition to the functions already built into MSX BASIC, there are also functions programmable by the user (user-defined functions). User-defined functions are defined with the DEF FN statement.

- Built-in function
- User-defined function

(3) Real numbers used in arguments

Arguments specified with integers or single-precision real numbers are all treated as double-precision real numbers in the operations of functions.

(4) Integers used in arguments

Arguments specified with integers or single-precision real numbers are truncated, in general, into integers (with all decimal places rounded off) for operations of functions.

7. ORDER OF PRIORITY OF OPERATIONS

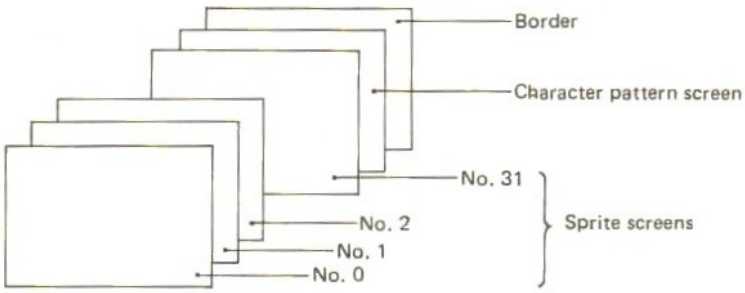
1. Operations enclosed in parentheses
2. Functions
3. Power (^)
4. Negative sign (−)
5. Multiplication (*) and division (/)
6. Integer division (\ )
7. Remainder (MOD)
8. Addition (+) and subtraction (−)
9. Relational operations (<, >, =, etc.)
10. Logical operation (NOT)
11. Logical operation (AND)
12. Logical operation(OR)
13. Logical operation (XOR)
14. Logical operation (EQV)
15. Logical operation (IMP)

Operations with the same priority are executed from left to right.

V SCREEN CONTROL

1. SCREEN TYPES

MSX BASIC can handle three screen types: i.e. one character pattern screen, up to 32 sprite screens, and one border.



- **Character pattern screen**
  This screen includes text and graphic modes, and allows you to list program lines, display messages, and draw various graphic figures or backgrounds.

- **Sprite screens**
  These screens allow you to display preprogrammed graphic patterns at specified locations on the screen to let you make animated pictures.

- **Border**
  This screen allows you to specify desired colours. It is not possible to display text or graphic information on this screen.

- **Screen type priority order**
  The sprite screens have the highest priority on the monitor, the character pattern screen has the next, and the border has the lowest priority.
  Among the sprite screens, screen No. 0 has the highest priority, and a descending order of priority is assigned to all the remaining screens.
  When more than one screen is on the monitor, the screen with a lower priority is hidden behind the screen with a higher priority. This eliminates the need for invisible line treatment in your program.

2. CHARACTER PATTERN SCREEN

The character pattern screen is used for text and graphic modes, and each mode is subdivided into two submodes. These modes and submodes are specified with the SCREEN statement.

Mode	Submode	Character capacity (initial value)	Resolution	Sprite
Text mode	40 x 24 Text	Max. 40 columns x 24 rows (24 rows x 37 columns)	———	No
	32 x 24 Text	Max. 32 columns x 24 rows (24 rows x 29 columns)	———	Yes
Graphic mode	High-resolution graphic	———	256 x 192	Yes
	Multicolour	———	64 x 48	Yes

- The character pattern screen modes and submodes can be specified with the integers 0 through 3 placed after the SCREEN statement:

  0 . . . . . 40 x 24 Text mode (SCREEN 0)
  1 . . . . . 32 x 24 Text mode (SCREEN 1)
  2 . . . . . High-resolution graphic mode (SCREEN 2)
  3 . . . . . Multicolour mode (SCREEN 3)

- Text mode is used for displaying characters and text symbols, and allows you to list program lines or view data. In text mode the following commands and statement are usable for screen control:

  PRINT, PRINT USING, WIDTH, LOCATE, CSRLIN, and POS.

- Graphic mode allows you to draw graphic figures or symbols on the screen. Characters and text symbols can also be displayed in this mode.
  In graphic mode the following commands and statements are usable for screen control:

  CIRCLE, DRAW, LINE, PAINT, PSET, PRESET, and POINT.

  When displaying characters or text symbols in this mode, use the PRINT# statement. The INPUT statement is not valid in graphic mode.

- The CLS and COLOR statements are usable in both text and graphic modes.

- No sprite screen is usable when the 40 x 24 text mode is selected.

- Power-on default is the 40 x 24 text mode in which up to 37 characters can be displayed per row.

(1) 40 x 24 TEXT MODE (SCREEN 0)

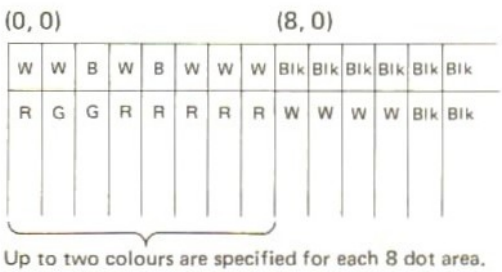
- The maximum capacity per screen is 24 rows of 40 characters each, whereas the default capacity is 24 rows of 37 characters each. The number of characters per row can be specified with the WIDTH statement.
- Each character cell consists of a 6 x 8 dot matrix. In the 40 x 24 Text mode, therefore, parts of graphic symbols may be lacking, since they require an 8 x 8 dot matrix.
- Up to two colours are specifiable: one for the text, and the other for the background.
- The sprite screen is not available in this mode.
- Graphic mode statements or commands (e.g. CIRCLE, DRAW, etc.) are not available in this mode.

(2) 32 x 24 TEXT MODE (SCREEN 1)

- The maximum capacity per screen is 24 rows of 32 characters each, whereas the default capacity is 24 rows of 29 characters each. The number of characters per row can be specified with the WIDTH statement.
- Each character cell consists of an 8 x 8 dot matrix.
- Up to three colours are specifiable for text, background, and border.
- Graphic mode statements or commands (CIRCLE, DRAW, etc.) are not available in this mode.

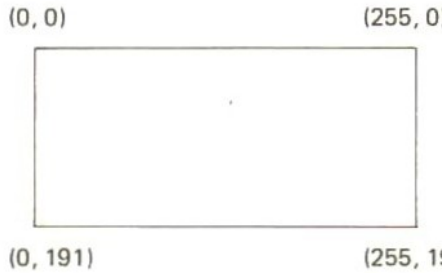
(3) HIGH-RESOLUTION GRAPHIC MODE (SCREEN 2)

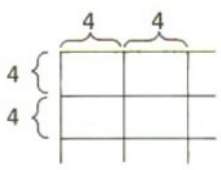
- This mode allows you to draw various graphic figures using a 256 x 192 point configuration per screen.
- Once the computer returns to the command mode at the completion or abortion of program execution, the Text mode will be automatically selected.
- Text mode statements (such as PRINT) or the INPUT statement are not usable in this mode.
- When you wish to display characters or text symbols on the graphic screen, use the OPEN statement to open a file on the graphic screen ("GRP:"), then use the PRINT# statement to print the desired characters or symbols. In this case, the top left corner of the first character corresponds to the last reference point (LP) on the graphic screen, which is specifiable with the PRESET statement.
- Up to two colours are specifiable for each 8-dot area at a time. This means that only two colours can be specified in the area between (0, 0) and (8, 0). If a third colour is specified within this area, all points in the area will be turned into the third colour.



(4) MULTICOLOUR MODE (SCREEN 3)

- This mode allows you to draw multicolour graphics using a 64 x 48 block (4 x 4 points per block) configuration. While display operation is controlled block-by-block, display location is specified by point.
- Once the computer returns to the command mode at the completion or abortion of program execution, Text mode will be automatically selected.
- Text mode statements (PRINT, etc.) or the INPUTstatement are not usable in this mode.
- When you wish to display characters or text symbols on the multicolour graphic screen, use the OPEN statement to open a file on the graphic screen ("GRP"), then use the PRINT# statement to input the desired characters or symbols into the file.
- Up to 16 colours are specifiable for each block.





Up to 16 colours can be specified for each block comprised of 4 x 4 points.

3. SPRITE SCREEN

The sprite screen allows you to display and move userdefined sprite figures on the screen. First use the SPRITE$ variable to define the desired sprite figure, then use the PUT SPRITE statement to display it on the screen.

(1) SPRITE FIGURE SIZE

Each sprite figure can be specified with an 8 x 8 or 16 x 16 dot matrix. When it is displayed on the screen, the height and width of each figure can be expanded by two in each direction. The figure size and expansion attribute are specified with the SCREEN statement.

(2) NUMBER OF DEFINABLE SPRITE FIGURES

- The number of definable sprite figures depends on the figure sizes.

Figure size	Definable figures	Figure size on display
8 x 8	256 types	8 x 8 points
		16 x 16 points
16 x 16	64 types	16 x 16 points
		32 x 32 points

- Up to 32 sprite figures can be displayed per screen, and up to 4 figures can be displayed on each row. If five or more sprite figures are specified per row, the 5th and subsequent figures will not appear on the screen.

(3) SPRITE INTERRUPT

If two sprite figures conflict (overlap) with each other, it causes an interrupt to the CPU. You need not check for conflict of figures in your program.
The Sprite interrupt is made available by:

- Specifying the first line of the interrupt service routine to which control is to be passed when a sprite interrupt occurs:

  ON SPRITE GOSUB

- Specifying whether the sprite interrupt is to be enabled, disabled, or held:

SPRITE ON	Enables interrupt
SPRITE OFF	Disables interrupt
SPRITE STOP	Holds interrupt

  If a conflict of sprite figures occurs after the SPRITE ON statement has been executed, it causes a sprite interrupt. Control is passed to the interrupt service routine whose first line number was specified by the ON SPRITE GOSUB statement.

4. COLOUR CODES

Colour codes are use to specify display colours. Numbers 0 through 15 are assigned to display colours as follows:

0 . . . . . Clear	8 . . . . . Red
1 . . . . . Black	9 . . . . . Bright red
2 . . . . . Green	10 . . . . . Yellow
3 . . . . . Light green	11 . . . . . Light yellow
4 . . . . . Dark blue	12 . . . . . Dark green
5 . . . . . Light blue	13 . . . . . Purple
6 . . . . . Deep red	14 . . . . . Grey
7 . . . . . Sky blue	15 . . . . . White

5. HOW TO SPECIFY COORDINATES

Screen coordinates specified in graphic statements (CIRCLE, LINE, PAINT, PSET, PRESET, and POINT) or the PUT SPRITE statement include absolute coordinates and relative coordinates.

Absolute coordinate

(x, y)
Specifies an absolute point on the screen referred to by the x and y values.

Relative coordinate

STEP (x, y)
Specifies a point on the screen which is apart from the last reference point LP by the lengths specified by the x and y values of each coordinate. The LP refers to the last coordinate point specified by a graphic statement, and has an initial value of (0, 0).

VI SOUND FEATURES

1. PROGRAMMABLE SOUND GENERATOR (PSG)

The PSG is a simplified music synthesizer designed to let you output tones to the internal speaker by using the PLAY or SOUND statement.

FEATURES OF THE PSG

1) Contains three independent audio channels (A, B, C) to simultaneously output up to three different tones. This allows you to play three-tone cords.
2) Capable of delivering a noise tone to each channel, as well as the musical tones, which lets you produce sound effects for your video games.
3) The "envelope" feature is available to vary the output tone level along the time axis. It allows you to add colour or beat to output tones.

2. MUSIC PLAY USING THE PLAY STATEMENT

The PLAY statement uses music macro commands to let the computer play music.

PLAY string expression A, string expression B, string expression C

String expressions A, B, and C represent music macro commands for channels A, B, and C, respectively. For details on the PLAY statement format, refer to Chapter 2, "LANGUAGE DESCRIPTION."
The following music macro commands are available:

(1) MACRO COMMANDS TO SPECIFY NOTES

A to G — These commands specify the seven whole notes in an octave scale in the order C, D, E, F, G, A, and B.

+ − — Used to specify the pitch of a note; to sharp or flat a note. To sharp a note, use "#" or "+", such as A# or A+; to flat a note use "−", such as A−.

Example: PLAY "F#", "G", "C"

O integer — This command specifies which octave out of eight available octaves is to be used, by using an integer between 1 and 8. This is specified in front of the notes (A ~ G), such as O5. The default value is O4. Once this command is specified, it remains valid until another octave command is subsequently specified. This command is valid only on the channel for which it is specified.

Example: PLAY "O4CDEFGABO5CDEFGAB"

N integer — This command specifies a specific pitch in the full eight octaves, by using an integer between 0 and 96. N1 specifies O1C#, and N95 specifies O8B. N0 denotes a rest. Every integer increment raises the pitch a half tone.

Example: PLAY "N36N38N40N41N43N45N47N48"

(2) MACRO COMMAND TO SPECIFY DURATION

L integer

This command is used to specify the duration (1/integer) of a note and for all the notes that follow. The valid range for the integer is from 1 to 64. The command default is L4.

L1	Whole note
L2	Half note
L4	Quarter note
L8	Eight note
L64	Sixty-fourth note

Once a duration command is specified, it remains valid until another duration command is subsequently specified. The duration command is valid only on the channel for which it is specified.

Example: PLAY "L1CDEFGAB"

When you wish to specify a specific note for a specific duration or rest, place the pertinent integer just following the desired note command, such as A16.

Example: PLAY "C1DEFGAB"

If a duration or rest command is followed by a period (.), the corresponding note is played 1.5 times as long. If two consecutive periods (..) are specified, the preceding note is played 2.25 times as long; if three consecutive periods (...) are specified, the note is played 3.375 times as long.

Example: PLAY "L1C.", "C1.", "C1C2"

(3) MACRO COMMAND TO SPECIFY REST

R integer

This command specifies the duration (1/integer) of a rest. If only R is specified, it is assumed to be R4.

R1	Whole rest
R2	Half rest
R4	Quarter rest
R8	Eighth rest
R64	Sixty-fourth rest

Example: PLAY "CDEF1FGABRCDEFGAB"

(4) MACRO COMMAND TO SPECIFY TEMPO

T integer

This command specifies the number of quarter notes to be played in one minute, for all the notes that follow. The valid range for the integer is from 32 to 255. The command default is T120.
Once a tempo is specified, it remains valid until another tempo is subsequently specified. This command is valid only on the channel for which it is specified.

Example: PLAY "T240CDEFGABT60CDEFGAB"

(5) MACRO COMMAND TO SPECIFY LOUDNESS

V integer — This command specifies the loudness of the tones that follow. The valid range for the integer is from 0 to 15, with the default value being 8. Once this command is specified, it remains valid until another loudness command is subsequently specified. The loudness command is valid only on the channel for which it is specified, and causes the envelope command, if specified, to be invalid.

Example: PLAY "V15CDEFGABV8CDEFGAB"

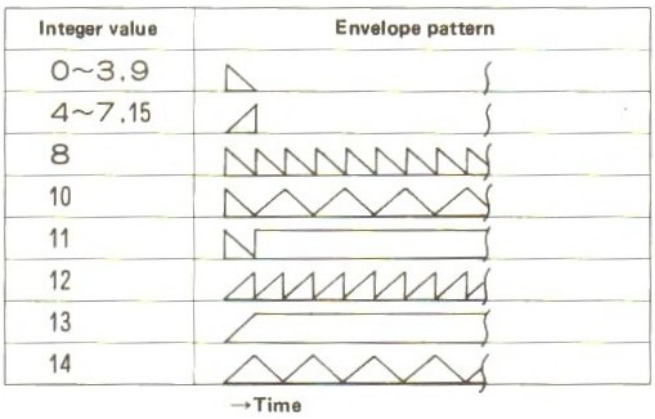
(6) MACRO COMMANDS TO SPECIFY TONAL COLOURS

The Envelope commands include the S and M command. The S command is used to specify the waveform, or envelope, according to which the loudness of the note is varied. The M command is used to specify the period of the envelope. You can control the tonal colour of your music by combining these commands.

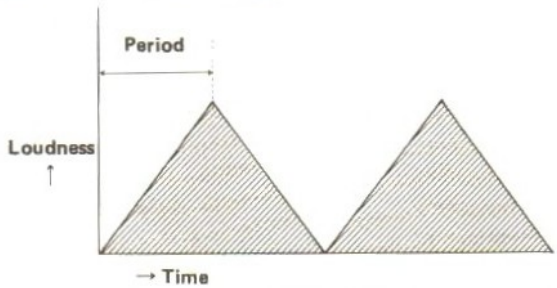
Once the Envelope command is specified, the Loudness command, V, is made invalid.

The Envelope command is valid on all three of the audio channels. This means only one envelope pattern and period can be specified for all three of the channels at a time.

S integer — This command specifies an envelope pattern. The valid range of the integer is from 0 to 15, with which the following eight envelope patterns can be specified.:



M integer — This command specifies the period of an envelope. The valid range of the integer is from 1 to 65535.



Example: PLAY "S0M10000CDEFGAB"

(7) MUSIC MACRO COMMANDS ASSIGNED TO VARIABLES

X Character variable; This variable causes the PLAY statement to play music according to the music macro commands assigned to this variable. It must be followed by a semicolon (;).

Example: A$="CDEFG" : B$="AB" : PLAY"XA$;XB$"

= numeric variable This variable may be used in place of an integer specified in a music macro command. It must be followed by a semicolon (;).

```
Example:  10  For J=3 TO 6
          20  PLAY"O=J;CDEFGAB"
          30  NEXT
          40  END
```

[DRILL] Let's play the first four bars of the "Wild Roses" by Schubert:


```
10 PLAY "O4T60S0M10000", "O4T60S0", "O4T60S0"
20 PLAY "L8BBBB", "L8GGGG", "L8DDDD"                          1st bar
30 PLAY "L16O5DCCO4BA4", L16BAAGE4", "L16F+EEDC4"             2nd bar
40 PLAY "L8AABO5C", "L8F+F+GA", "L8DDEF+"                     3rd bar
50 PLAY "O5D4GR8", "A4O5DR8", "F+4BR8"                        4th bar
60 END
```

10 PLAY "O4T60S0M10000"	O4	Specifies the 4th octave.
	T60	Sets the tempo to 60 quarter notes per minute.
	S0M10000	Specifies the envelope to simulate a piano sound.
20 PLAY "L8BBBB"	L8	Specifies eighth notes.
	BBBB	Plays the B note four times.
30 PLAY "L16O5DCCCO4BA4"	L16	Specifies sixteenth notes.
	O5	Specifies the 5th octave.
	DCCC	Plays D, C, C, and C notes.
	O4	Specifies the 4th octave.
	A4	Plays a quarter note of A pitch.

3. SOUND EFFECTS USING THE SOUND STATEMENT

The SOUND statement lets you load values into the registers of the programmable sound generator (PSG). Appropriate combinations of several SOUND statements allows the PSG to create various sound effects which are not available with the PLAY statement alone.

SOUND register number, integer expression

This statement causes the value of the integer expression to be loaded into the specified PSG register. For more details on the SOUND statement format, refer to Chapter 2, "LANGUAGE DESCRIPTION".

(1) PSG REGISTERS

The PSG contains 16 registers, of which 14 registers are available to the user.

Register No.	Function	Bit							
		b7	b6	b5	b4	b3	b2	b1	b0
0	Frequency on CH.A	FT (A)							
1						CT (A)			
2	Frequency on CH.B	FT (B)							
3						CT (B)			
4	Frequency on CH.C	FT (C)							
5						CT (C)			
6	Noise frequency				NP				
7	Output channel select			Noise			Tone		
		1	O	C	B	A	C	B	A
8	Loudness on CH.A				M	L (A)			
9	Loudness on CH.B				M	L (B)			
10	Loudness on CH.C				M	L (C)			
11	Envelope period	FT (E)							
12		CT (E)							
13	Envelope pattern					EP			

(2) HOW TO USE THE PSG?

The PSG can produce up to three tone signals and one noise signal at a time. It outputs them to the internal speaker through the three audio channels (A, B, C) to create various sound effects.

The tone for each channel is specified by loading the desired pitch, or frequency, data into register 0 through 5.

Noise frequency data is loaded into register 6.

Register 7 is used to select one of the three tone channels to be output, and determine whether the output channel delivers a tone, a noise or a combination of tone and noise. While there is only one noise source available, it may be output to any desired channel together with the tone signals.

Registers 8, 9, and 10 are used to set the loudness levels on the three channels. Either a constant loudness or loudness varying according to a specific envelope pattern can be selected with these registers. When an envelope is specified, the envelope pattern is loaded into register 13, and the envelope period is loaded into registers 11 and 12.

1) Tone frequency setup

To set up a tone frequency, first determine the TP value (up to 12 bits) obtained from the following formula, then divide the TP value into high order 4 bits (CT) and low order 8 bits (FT), as follows:

```
TP=1789772.5/(16*F)
CT=TP \ 256              High order 4 bits of TP
FT=TP MOD 256            Low order 8 bits of TP
```

The FT and CT for channels A, B, and C are loaded into registers 0 and 1, 2 and 3, and 4 and 5, respectively. In registers 1, 3, and 5, only the low order 4 bits are meaningful.

2) Noise frequency setup

To set up a noise frequency (in Hertz), load a five-bit value, NP, determined from the following formula, into register 6:

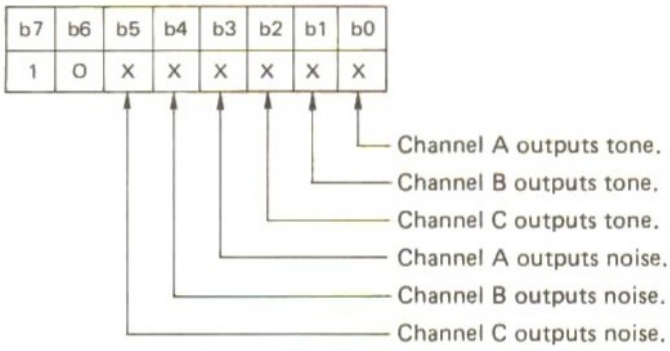
NP=1789772.5/(16*F)

Only the low order five bits of register 6 are meaningful, with the high order three bits meaningless.

3) Channel select

To select the output channel, set a zero into the bit of register 7 that corresponds to that output channel.

Register 7 bit configuration ..... Set 0 into the bit corresponding to the output channel.



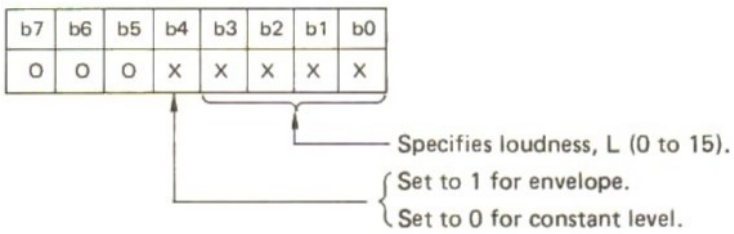
The most significant 2 bits of this register are used to specify the input/output direction of the general-purpose I/O ports, which have nothing to do with the sound output function.

4) Loudness setup

The loudness levels on channels A, B, and C are set up with registers 8, 9, and 10, respectively.

Whether a constant loudness or varying (envelope) loudness is to be used can be specified for each channel by using bit 4 on each register. The most significant 3 bits of each register have no meaning.

Bit configuration on registers 8, 9, and 10.



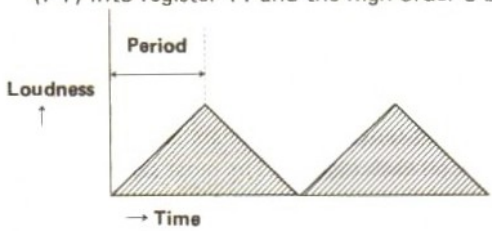
5) Specifying the envelope pattern and period

The tone output on the channel for which an envelope is specified by registers 8, 9, or 10 varies at the period specified by registers 11 and 12, and according to the envelope pattern specified by register 13. The following eight envelope patterns are specifiable with the least significant 4 bits of register 13.

4 LSB's				Envelope pattern
b3	b2	b1	b0	
0	0	×	×	
0	1	×	×	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

→Time

The envelope period, T (in sec.), can be specified by loading the value of TP (up to 16 bits) determined from the following formula, into registers 11 and 12: low-order 8 bits (FT) into register 11 and the high-order 8 bits (CT) into register 12.




```
TP= 1789772.5*T/256
CT=TP\256                    High order 8 bits
FT=TP MODE 256               Low order 8 bits
```

[DRILL] Create a gun shot sound.

```
10 SOUND  6, 15              Set the noise frequency.
20 SOUND  7, &B10000111      Specifies channels A, B, and C for noise output.
20 SOUND  8, &B00010000      Specifies the envelope for channel A.
40 SOUND  9, &B00010000      Specifies the envelope for channel B.
50 SOUND 10, &B00010000      Specifies the envelope for channel C.
60 SOUND 11, 0: SOUND 12, 16 Specifies the envelope period.
70 SOUND 13, 0               Specifies the envelope pattern.
80 END
```

Ⅶ FILE

A file is a set of related records, data, or program steps usually stored on a storage medium (such as cassette tape). Each file has its own name to discriminate it from all other files. The concept of a file may be applicable to various input/output devices and their media, including cassette tape.

1. SPECIFYING A FILE

A file is specified with a device name and a file name as follows. Program or data is input to or output from the specified file on the specified device.

"DEVICE NAME FILE NAME"

- A file may also be specified with character variables or character expressions, as well as character strings enclosed in quotation marks.
- The device or file name is defaultable.

(1) Device name

The device name must specify the input/output device used for file access.

I/O device	Device name	Usable mode	
		INPUT	OUTPUT
Cassette recorder	CAS :	○	○
Text mode screen	CRT :	×	○
Graphic mode screen	GRP :	×	○
Printer	LPT :	×	○

- The display screen can also be treated as an output device, in which case the screen is used chiefly for displaying characters or text symbols in Graphic mode.
- The device name may be specified with either uppercase or lowercase characters.
- The CSAVE or CLOAD statement requires no device name specification.

(2) File name

The file name specification is needed when accessing a cassette tape file. No file name is needed for accessing any file other than a cassette tape file.

- A file name must be a string of no more than six characters.
  If a file name contains less than six characters, the remaining locations are filled with blanks.
  If a file name exceeds six characters, the seventh and all subsequent characters are ignored.
- A file name should not contain a colon (:) or the numeric characters 0 and 255 (&HFF).

2. PROGRAM FILE

The following commands are used for saving or loading program files:

CSAVE	Saves a program to a cassette tape file.
CLOAD	Loads a program, which was saved with the CSAVE command, from a cassette tape file.
SAVE	Saves a program of ASCII form to a file on a specified device.
LOAD	Loads a program of ASCII form from a file on a specified device.
MERGE	Merges an ASCII program file with the program currently in memory.
BSAVE	Saves a machine code program to a specified device.
BLOAD	Loads a machine code program from a specified device.

3. DATA FILE

(1) Open a File

When accessing a data file, you must open the file in advance by using an OPEN statement. The OPEN statement specifies the device name, file name to be opened, input/output direction, and file number. Internally the OPEN statement causes a file control block (to be used for I/O operations) to be set aside in memory.

(2) File number

A file number represents the device and file names specified in the OPEN statement. The PRINT# or INPUT# statement accesses the file whose file number is specified following these statements.

The available range of file number is specified by the MAXFILES statement. The initial value of a file number is one (1).

(3) Closing a file

When I/O access to a file is completed, you must close the file by using the CLOSE statement. If the file is left open, you cannot open another file using the same file number.

VIII INTERRUPTS

The purpose of an interrupt is to tell the computer's CPU that it must suspend whatever it is doing, process the event or data being input, then continue its suspended operations. If an interrupt signal is not available, the main program will always have to check to see if an event or external logic is requesting service. With an interrupt signal, however, you have only to specify interrupt service routine names and interrupt enable statements at the beginning of the main program. This eliminates the need for a service request check program and speeds execution of the main program.

Possible Causes of Interrupts Interrupt Service Statements

Error interrupt . . . . . ON ERROR GOTO

Function key interrupt . . . . . ON KEY GOSUB

Stop key interrupt . . . . . ON STOP GOSUB

Sprite conflict interrupt . . . . . ON SPRITE GOSUB

Joystick trigger interrupt . . . . . ON STRIG GOSUB

(Space bar interrupt)

Interval timer interrupt . . . . . ON INTERVAL GOSUB

The interrupt priority order is from the top to the bottom of this list.

Example: In the following program, control branches to line 1000 when the Space bar is pressed:

When using interrupts (Program)

```
10   ON STRIG GOSUB 1000
20   STRIG (0) ON
30   TIME=0
40   FOR I=1 TO 5000
50   PRINT T, TIME-T
60   T=TIME
70   NEXT
80   END
1000 PRINT "SPACE ON"
1010 RETURN
```

When not using interrupts (Program)

```
10   TIME=0
20   FOR I=TO 5000
30   PRINT T, TIME-T
40   T=TIME
50   K$=INKEY$
60   IF K$=" " THEN GOSUB 1000
70   NEXT
80   END
1000 PRINT "SPACE ON"
1010 RETURN
```

IX MACHINE LANGUAGE

(1) DEVELOPMENT OF A MACHINE-LANGUAGE PROGRAM

The HX-10 computer uses a Z80A processor for its CPU. So you must uses Z80 machine code for your machine language programs. For details on machine code, read the books written about the Z80 Machine Language.

1) Use the CLEAR statement to set aside a memory area to be used for the machine code program.
2) Create your machine language program by using the POKE and PEEK statements.
3) To save your machine language program on cassette tape, use the BSAVE statement. To load it into memory, use the BLOAD statement.
4) Your machine language program can be run with the DEFUSR statement and the USR function.

Note: A small bug in your machine code program can make the system unrecoverable from an error state, and you will have to temporarily turn off the system before continuing. It is strongly advisable that you save your machine code program on a cassette before executing it.

(2) ARGUMENT TRANSFER USING THE USR FUNCTION

USR number, argument

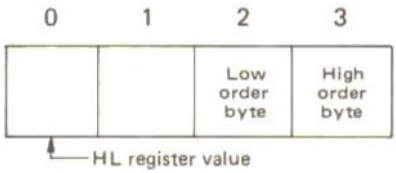
1) Data is transferred between a BASIC program and machine language program with arguments. A machine language program uses the A and HL or DE registers to read argument data.
2) The A register value is 2, 3, 4, or 8 depending on the argument type.
When transferring the result of machine-language program execution to a BASIC program, the result must have the same format as the argument, and if it is a character string, must have the same string length as the argument. The resultant data must have the same address as that of the operand data which was transferred from the BASIC program.

A register value	Argument type
2	Integer
3	Character string
4	Single-precision real number
8	Double-precision real number

3) The HL or DE register value specifies the address of the data to be transferred. The address specification format differs depending on argument type:

- **Integer**

  An integer is represented in two-byte binary form, and is stored in memory in the order low-order byte and high-order byte, starting with the address specified by (HL register value + 2).

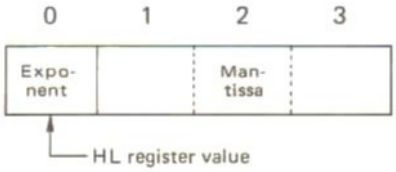


- **Single-precision real number**

  A single-precision real number is represented by one byte for the exponent and three bytes for the mantissa (4 bytes in all), and is stored in memory in the order exponent and mantissa, starting with the address specified by the HL register value.

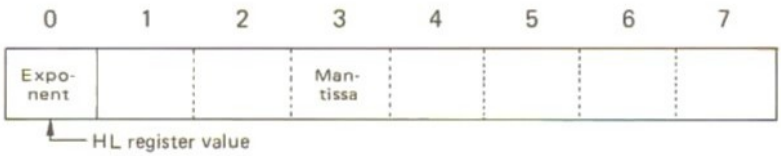
  The MSB of the exponent part specifies the sign (0 for positive, 1 for negative) of the number, and the remaining seven bits represent the exponent, from E+62 to E–64.

  The mantissa is represented by a six-digit binary coded decimal.



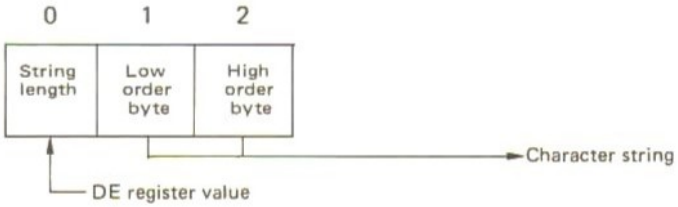
- **Double-precision real number:**

  A double-precision real number is represented by one byte for the exponent and seven bytes for the mantissa (8 bytes in all), and is stored in memory in the order of exponent and mantissa, starting with the address specified by the HL register value.



- **Character string**

  For a character string, the length and the low- and high- order bytes of the address where the character string is located are stored in memory, in the order in which they are written, starting with the address identified by the DE register value.



CHAPTER 2

LANGUAGE DESCRIPTION

EXPLANATORY NOTES ON FORMAT

Square brackets [ ] — Denote an optional item.

Example

AUTO [beginning line number] [, increment]

- AUTO
- AUTO beginning line number
- AUTO ,increment
- AUTO beginning line number, increment

..... repetition — Denotes any number of repetitions within one line.

Example

Constant [, constant···]

- Constant
- Constant, constant
- Constant, constant, constant
- etc.

Integer expression — When "integer expression" is specified, a numeric expression (including a numeric variable or constant) may be specified. If a double- or single-precision real number is contained in the expression, the value of the expression is converted to an integer before execution, however.

Example

CHR$ (integer expression)

A$ = CHR$ (65.23) "A" is assigned to variable A$.
A$ = CHR$ (X) Double precision real type variables are also valid.

ABS (Absolute) — Function

Function The ABS function returns to the program the absolute value of an expression.

Format ABS (numeric expression)

Descriptions
1) The ABS function returns the absolute value of the numeric expression that follows it.
2) The result is always returned as a double-precision real number, regardless of the type of the numeric expression.

Example
```
10 A = — 1 : B% = 1
20 AA = ABS (A) : BB = ABS (B%)
30 PRINT AA, BB
40 END
```

ASC (ASCII) — Function

Function The ASC function returns the ASCII code for the character that follows it.

Format ASC (string expression)

Descriptions
1) The ASC function returns to the program the ASCII code for the first character of the string expression that follows it. The second and all remaining characters in the expression are ignored.
2) If the string expression is a null string (" "), an error will result (Illegal function call).
3) When the string expression is a graphic symbol, the ASCII code for the graphic character header (&H01) is returned.

Example
```
10 A$ = "ABC"
20 PRINT ASC (A$), ASC("D")
30 END
```

ATN (Arc Tangent) — Function

Function This function returns the arctangent of a number.

Format ATN (numeric expression)

Descriptions
1) The ATN function returns the arctangent of the numeric expression that follows it.
2) The result is returned in radians, between $-\pi/2$ and $\pi/2$.
3) The result is always returned as a double-precision real number, regardless of the type of the numeric expression.

AUTO Command

Function This command generates an automatic line numbering sequence.

Format AUTO [first line number] [, increment]

Descriptions

1) When this command is input and the RETURN key is pressed, the first line number (10 when both first line number and increment are omitted) appears on the screen. Each time you enter a program line and press the RETURN key, a new line number with the specified increment appears in the next line position. It allows you to avoid having to type a line number for each program line.
2) The line number must be an integer between 0 and 65529, and the increment must be a positive integer.
3) When the first line number specification is omitted, a zero is assumed for it. When the increment is omitted, a ten(10) is assumed for it. When both the first line number and increment options are omitted, a 10 is assumed for both of them.
4) To clear the AUTO command function, simultaneously press the CTRL and STOP keys. The system returns to command mode, with the prompt "Ok" appearing on the screen.
5) If a line number which already exists in the program currently in memory is generated, an asterisk (*) will appear following the line number.

   If you type a program line following the asterisk, and press the RETURN key, the old program line in memory will be replaced with the new program line just typed in. If you just press the RETURN key without entering a program line following the asterisk, the old program line in memory will remain as it is.
6) The screen editor functions (edit functions using the cursor) are also available when the AUTO command is active.

Example AUTO 1000, 10

BASE — Function

Function The BASE function returns the first address of the tables in the VRAM.

Format BASE (integer expression)

Descriptions

1) This function returns the first address of the table (within the VRAM) specified by the integer expression that follows it.
2) The allowable range of the integer expression is from 0 to 19. The specifiable tables are listed:

Screen mode / Table name	40 x 24 Text	32 x 24 Text	High-resolution graphic	Multicolour
Name table	0	5	10	15
Colour table	Not used	6	11	Not used
Pattern generator table	2	7	12	17
Sprite attribute table	Not used	8	13	18
Sprite pattern table	Not used	9	14	19

BEEP — Statement

Function The BEEP statement causes an internal buzzer to sound.

Format BEEP

Description This statement outputs a "pip" tone to the internal speaker for approximately 0.04 sec.

BIN$ (Binary $) — Function

Function This function converts a numeric value into a binary character string.

Format BIN$ (integer expression)

Description The BIN$ function converts the integer expression that follows it into a character string of binary notation.

Example

```
10 A$ = BIN$ (16)      Character string "10000" is assigned to variable A$.
20 PRINT A$
30 END
```

BLOAD Command

Function This command causes a memory image file to be loaded into memory.

Format BLOAD "device name [file name]"[, R] [, offset]

Descriptions

1) The BLOAD command is used to load a machine language program or data file (which was saved with the BSAVE command) into memory.
2) When the file name is omitted for cassette tape files, the first file on the tape is loaded.
3) When the R option is specified, execution of the loaded machine language program is initiated from the execution start address specified in the B SAVE command, immediately after the program is loaded.
4) When an offset is specified, program or data is loaded into an address area whose first and last addresses are the sum of the offset and the addresses specified in the BSAVE command.
5) Since the BLOAD command causes a machine code program or data to be loaded into any location in memory, endless execution of the BLOAD command, or program runaway, may occur if the program or data is loaded into the work area or file control block. Pay special note of the offset value and the addresses specified in the BSAVE command.

Example

```
BLOAD"CAS:SAMPLE",&H1000          Only offset is specified.
BLOAD"CAS:SAMPLE",R               Only R option is specified.
BLOAD"CAS:SAMPLE",R,&H1000        Both offset and R option
                                  are specified.
```

BSAVE Command

Function The BSAVE command saves portions of the computer's memory on a specified device.

Format BSAVE "device name [file name]", first address, last address [,execution start address]

Descriptions

1) This command saves the memory image contents of the portion of the memory between the first and last addresses specified in it.
2) If the R option is specified in the BLOAD command, execution of the program loaded by the BLOAD command is automatically initiated from the execution start address specified in this command. When the execution start address is omitted, the first address is assumed for it.

Example

```
BSAVE"CAS:SAMPLE",&HD000,&HD1000
BSAVE"CAS:SAMPLE",&HD000,&HD1000,&HD00A
```

CALL — Statement

Function This statement calls an extended statement.

Formats
CALL extended statement name
CALL extended statement name (argument, [,argument···])
_ extended statement name [(argument [,argument···])]

Descriptions
1) The CALL statement calls an extended statement, written in machine language, from an extended ROM cartridge.
2) An underscore (_) is usable in place of CALL.
3) For details on extended statements, refer to the MSX software specifications document.

Examples
```
CALL SAMPLE
CALL SAMPLE ("A","123","XX")
```

CDBL (Convert to Double) — Function

Function This function converts single or double precision real number that follows it to a double precision real number.

Format CDBL (numeric expression)

Descriptions
1) The CDBL function converts the value (integer or single-precision real number) of the expression that follows it to a double precision real number.
2) The number of significant figures of the values does not change before or after conversion.

Example
```
A#=CDBL(B%)
```

CHR$ (Character $) — Function

Function This function converts an ASCII code to its character equivalent.

Format CHR$(integer expression)

Descriptions
1) The CHR$ function returns the character, symbol, or control code for the ASCII code specified by the integer expression that follows it.
2) For the ASCII code list, refer to Chapter 3, Section 1, "Character Code Table."

Example
```
A$=CHR$(&H41)    Character "A" is assigned to the variable A$.
```

CINT (Convert to Integer) — Function

Function This function converts a numeric expression to an integer.

Format CINT (numeric expression)

Descriptions

1) The CINT function converts the value of the numeric expression that follows it to an integer, by truncating the decimal places of the value.
2) If the result of the conversion is outside the range from −32768 to +32767, an error will result.

Example

```
A% = CINT(B#)
```

CIRCLE

Statement

Function

This statement is used to draw circles or ellipses on a graphic screen.

Formats

CIRCLE (X coordinate, Y coordinate), radius[, colour code]
CIRCLE (X coordinate, Y coordinate), radius,[colour code], begin angle
CIRCLE (X coordinate, Y coordinate), radius,[colour code], [begin angle], end angle
CIRCLE (X coordinate, Y coordinate), radius,[colour code], [begin angle], [end angle], ratio of Y radius to X radius

Coordinates (X coordinate, Y coordinate) may also be specified with the relative coordinate specification using STEP (X coordinate, Y coordinate).

Descriptions

1) The CIRCLE statement causes a circle, arc, or ellipse to be drawn on a graphic screen, with its center located at the specified coordinates, and using the colour specified by the colour code.
2) The colour code must be an integer from 0 to 15.
   When the colour code is omitted, the colour specified by the COLOR statement is used for drawing.
3) The begin and end angles must be specified by numeric expressions in radians, ranging from -2π to 2π.
   When the begin and end angles are omitted, 0 and 2π are assumed for them, respectively, and causes a circle to be drawn on the screen.
   When negative values are specified for the begin and end angles, the absolute value of the angles are used for drawing, and a fan shape (with the center of the arc linked to each end of the arc by straight lines) will be drawn on the screen.
4) The ratio of Y radius to X radius must be specified with a numeric expression.
   When the ratio specification is omitted, 1.0 is assumed for it, and causes an arc of the specified radius to be drawn on the screen.
   When the ratio, radius along the Y axis/radius along the X axis, is specified, an ellipse with the specified ellipticity is drawn on the screen.
   The radius specified in the statement refers to the larger of the Y and X radii.
   When the specified ratio is smaller than one, the specified radius refers to the Y radius.
   When the specified ratio is larger than one, the specified radius refers to the X radius.
5) If the coordinates are specified by using the relative coordinate specification, STEP(X coordinate, Y coordinate), the center of a circle or arc is given by the distance from the last reference point (LP).
   When the CIRCLE statement is executed, the LP is set to the center coordinates.

Example

```
10 COLOR 15, 5 : SCREEN 2
20 P = 3.1415927
30 CIRCLE (20, 20), 20, 1                Draws a circle.
40 CIRCLE (60, 60), 20,, 0, P            Draws a half circle.
50 CIRCLE (100, 100), 20, 1, -P/2, -P    Draws a fan figure.
60 CIRCLE (140, 140), 20,,,, 2           Draws an ellipse.
70 CIRCLE (180, 180), 20,,,, 1/2
80 GOTO 80
```

CLEAR — Statement

Function

This statement initializes all variables and sets the size of the user's area in memory.

Format

CLEAR [string area size [, upper limit address of memory]]

Descriptions

1) The CLEAR statement frees all memory used for data without erasing the program currently in memory.
   It sets all numeric variables to zero and all string variables to null (" ").
   All open files are closed.
   The contents of the statements which begin with a DEF (DEF FN, DEF USR, DEFINT, DEFSNG, DEFDBL, DEFSTR, etc.) are all made invalid.
   All definitions of arrays are cleared.
   FOR NEXT loops are discontinued.
   Control is not returned from subroutines by the RETURN statement.
2) The size of the string area (in which character strings assigned to string variables are stored) must be specified by an integer expression, in bytes.
   The initial default size is 200 bytes.
3) The upper limit address of the user's area in memory must be specified by an integer expression, in bytes. For mapping in the memory, see Chapter 3, Section 3, "Memory Map".
   The area between the specified upper limit address and &HF380 is not accessible to the BASIC program for its program or data area, and the machine code program written in this area will not be destroyed.
   The initial default value of the upper limit is &HF380, which is equal to the maximum available address.

Example

```
10 CLEAR 1000, &HE000
```

CLOAD — Command

Function This command allows the user to load a program file from cassette tape into memory.

Format CLOAD ["file name"]

Description

1) The CLOAD command is used to load a specified program file from a cassette tape into the computer's memory.
   When the specified file is found, the computer displays a message "Found: file name" on its screen, and starts the loading operation. When the loading operation is completed, the prompt "Ok" appears on the screen.
   Each time a file other than the specified one is found, the computer will display "Skip: file name."
2) The file name must be six or less alphanumeric characters. If seven or more characters are used for a file name, the seventh and all remaining characters are ignored.
   When the file name is omitted, the file which is first found will be loaded.
3) When the CLOAD command is executed, all programs and variables previously in memory are cleared, and all open files are closed.
4) The data transfer rate for the CLOAD command need not be specified. It is automatically set to the rate at which the save operation was performed.

Examples

```
CLOAD"SAMPLE"
CLOAD
```

CLOAD? (Verify) — Command

Function This command allows the user to compare a program on cassette tape with one in memory, for verification.

Format CLOAD? ["file name"]

Descriptions

1) The CLOAD? command is used to verify that the program currently in memory agrees with one in a cassette file, by comparing them while reading from the cassette file. If the two programs completely agree with each other, "Ok" will appear on the screen. If not "Verify error" will appear on it.
2) This command is generally used immediately after the CSAVE command is executed, to verify that the program was properly saved on cassette tape.

Examples

```
CLOAD?
CLOAD?"SAMPLE"
```

CLOSE — Statement

Function This statement causes all, or specified, open devices and files to be closed.

Format CLOSE [[#] file number [,[#] file number ···]]

Descriptions

1) The CLOSE statement causes the file specified by the file number to be closed.
   The file number used for file closing may be used for specifying another file to be opened.
2) More than one file can be closed at a time by specifying their file numbers in a single CLOSE statement.
3) When the file number is omitted, all open files are closed.
4) If the CLOSE statement is executed for a file which was opened for data output, all data remaining in the buffer is output to that file.
   To properly complete an output operation to a file, the file must be closed.
5) The END, RUN, NEW, or CLEAR statements also cause open files to be closed.

Examples

```
CLOSE #1
CLOSE 1, 3
```

CLS (Clear Screen) — Statement

Function This statement clears the screen.

Format CLS

Descriptions

1) The CLS statement clears all characters and graphic figures from the screen, except for sprite figures.
2) On a text mode screen, the CLS statement does not clear the function indicators in the bottom area of the screen, and causes the cursor to be positioned at the home position (0, 0).
3) On a graphic mode screen, execution of the CLS statement causes the background colour to be changed to the colour specified by the COLOR statement.
   The location of the last reference point (LP) does not change before or after the execution of this statement.

Example

```
CLS
```

COLOR Statement

Function This statement specifies display colours.

Formats
COLOR foreground colour [, background colour]
COLOR [foreground colour] , background colour
COLOR [foreground colour] , [background colour] , border colour

Descriptions

1) The COLOR statement is used to specify foreground colour (for characters or graphics), background colour, and/or border colour on the screen.
2) Foreground colour
   On a text mode screen the colour for characters on the screen is specified by the foreground colour. Once the COLOR statement is executed, all the characters currently on the screen turn the colour specified by the foreground colour.
   On a graphic mode screen, the foreground colour specifies the colour of each graphic figure. This colour specification is valid when the colour code is omitted in Graphic statements (PSET, LINE, CIRCLE, DRAW, PAINT, etc.).
3) Background colour
   On a text mode screen, the execution of the COLOR statement immediately affects the background colour.
   On a graphic mode screen, the background colour is affected when a CLS statement is executed after the COLOR statement. The COLOR statement affects the background colour before a CLS statement is executed if the colour code is omitted in the PRESET statement.
4) Border colour
   Border colour specifies the colour for the border area (where no character or graphic figure can be displayed) outside the background.
5) Colour codes are used for specifying the foreground, background, and border colours. The power-on default values of colour codes are 15, 4, and 4, respectively.

Colour codes

0	Clear	8	Red
1	Black	9	Bright red
2	Green	10	Yellow
3	Light green	11	Light yellow
4	Dark blue	12	Dark green
5	Light blue	13	Purple
6	Dark red	14	Grey
7	Sky blue	15	White

Example
```
COLOR 10, 15, 1
```

CONT (Continue) — Command

Function This command is used to resume execution of a program after a break.

Format CONT

Descriptions

1) The CONT command is used to resume execution of a program which was broken by simultaneous operation of the CTRL and STOP keys, execution of a STOP or END statement, or an error generation.
2) In general, this command is used in program debugging. After breaking program execution with a STOP statement, you can check the values of variables, by executing a PRINT command in Direct mode, or modify them by using the LET statement, before resuming program execution with the CONT command.
3) If the program contents have been modified during a break period, the CONT command will not resume program execution.

Example CONT

COS (Cosine) — Function

Function This function returns the trigonometric cosine of a number.

Format COS (numeric expression)

Descriptions

1) The COS function returns the trigonometric cosine value of the numeric expression that follows it.
2) The value of the numeric expression must be in radians.
3) The result is always a double-precision real number regardless of the numeric expression type.

Example A = COS(1.73)

CSAVE — Command

Function This command is used to save a program file on a cassette tape.

Format CSAVE "file name" [, baud rate]

Descriptions

1) The CSAVE command causes programs to be transferred from memory to cassette tape.
   Saved programs can be reloaded into memory with the CLOAD command.
   Immediately after saving a program, use the CLOAD? command to verify that the program has been properly saved.
2) The file name must be six or less alphanumeric characters. If it exceeds six characters, the seventh and all remaining characters are ignored.
3) The baud rate may be used to specify the data transfer rate, with an integer of 1 or 2.
   When 1 is specified: 1200 baud
   When 2 is specified: 2400 baud
   When the baud rate is omitted, the baud rate previously specified in the SCREEN statement or CSAVE command is used. The initial default value of the baud rate is 1200 baud.

Examples

```
CSAVE"SAMPLE"
CSAVE"SAMPLE", 2
```

CSNG (Convert to Single) — Function

Function This function converts an integer or double precision real number to a single precision real number.

Format CSNG (numeric expression)

Descriptions

1) The CSNG function converts the numeric expression that follows it to a single precision real number.
2) When the value of the numeric expression contains seven or more significant figures, the seventh significant figure is rounded to the nearest whole number.

CSRLIN (Cursor Line) — Function

Function This function returns the value of the vertical coordinate of the screen cursor.

Format CSRLIN

Descriptions

1) CSRLIN returns the row number of a text mode screen on which the cursor is currently located.
2) The result is always an integer between 0 and 23, 0 being assigned to the first row.

Example

```
PRINT CSRLIN
```

DATA Statement

Function This statement stores numeric and string constants (to be accessed by a READ statement) within a program.

Format DATA constant [, constant····]

Descriptions

1) The DATA statement stores numeric and string constants within a program. These constants are accessed by a READ statement within the same program.
2) The DATA statement is a non-executable statement, and may be located on any program line.
3) More than one constant is specifiable in a DATA statement by separating them with commas (,), as long as the program line containing the DATA statement does not exceed 255 characters.
4) Constants specified in a DATA statement may be either numeric or string constants.
   Numeric constants may be of any type.
   Quotation marks (") may be omitted from string constants when placed in a DATA statement. It is not allowed, however, to omit quotation marks when the string contains commas (,), colons (:), or semicolons (;) within it, or when there are one or more blanks at the first or last position of the string.
5) The type of the constants placed in a DATA statement must match the type of variables placed in the corresponding READ statement.
   If numeric constants are read by string variables, the numeric constants are regarded as string constants.
   If string constants are read by numeric variables, an error will result.
   The type of numeric constants (integer, single-precision real number, double-precision real number) need not match that of numeric variables to which the former are assigned. Type conversion occurs when constants are read into variables.
6) More than one DATA statement may be used in a program.
   The READ statement reads constants beginning with those stored in the DATA statement which is on the line with the smallest line number.
   The RESTORE statement allows you to read constants from a DATA statement on the specified line.

Example DATA 1.23, &HE9,SAMPLE, "X : X"

DEF FN (Define Function) — Statement

Function This statement defines and names a user-defined function.

Format DEF FN function name [(argument [, argument···])] = expression of the function

Descriptions

1) The DEF FN function assigns the expression of the function specified on the right part to the function name specified on the left part.
   When the "FN function name" is called, the defined expression of the function is operated to return the result.
2) The function name and its type are specified in the same way as a variable name and its type.
   The type of the function name must match the type of the expression of the function.
3) The variable names assigned to arguments correspond to the variables with the same names used in the expression of the function.
   These variables are not affected by manipulation of variables with the same names used in other portions of the same program.
   If there is a variable name in the expression of the function which does not correspond to a variable assigned to an argument, the value of the variable with the same name in other portions of the program is assigned to the variable for the argument.
4) This function is not usable in Direct mode.

Example

```
10 DEF FNS(X, Y) = SQR(X*X + Y*Y)
20 A = FNS(4, 3) : PRINT A
30 END
```

DEF USR (Define User) — Statement

Function This statement defines the first address of a machine-code subroutine.

Format DEF USR [number] = first address

Descriptions

1) The DEF USR statement defines the first address of a machine code subroutine called by the USR function.
2) The number may range from 0 to 9, which allows for specification of up to ten machine-code subroutines.
   When the number is omitted, zero is assumed for it.

Example

```
DEF USR = &HF100
```

DEFDBL (Define Double) Statement

Function This statement is used to declare that the variables that follow it are double-precision real type variables.

Formats

DEFDBL alphabetic character-alphabetic character [, alphabetic character-alphabetic character···]

DEFDBL alphabetic character [, alphabetic character···]

Descriptions

1) The DEFDBL statement is used to declare that the variables whose names begin with the specified alphabetic character or with a character in the specified range of alphabetic characters and have no type declaration symbols following them, are double-precision real type variables.

Alphabetic character-alphabetic character:

The variables whose names begin with a character in the specified range of alphabetic characters are defined as double-precision real type.

Alphabetic character

The variables whose names begin with the specified alphabetic character are defined as double-precision real type.

2) Variable names with type declaration symbols (%, !, #, or $) have the types specified by those symbols.

Example

DEFDBL A, D-F Variables AB, D, E1, and so forth are defined as double-precision real type.

DEFINT (Define Integer) Statement

Function This statement is used to declare integer type variables.

Formats

DEFINT alphabetic character-alphabetic character [, alphabetic character-alphabetic character···]

DEFINT alphabetic character [, alphabetic character···]

Descriptions

1) The DEFINT statement is used to declare that the variables whose names begin with the specified character or with a character in the specified range of characters and have no type declaration symbols, are integer type variables.

Alphabetic character-alphabetic character:

The variables whose names begin with a character in the specified range of alphabetic characters are defined as integer type.

Alphabetic character:

The variables whose names begin with the specified alphabetic character are defined as integer type.

2) Variable names with type declaration symbols (%, !, #, or $) have the types specified by those symbols.

Examples

DEFINT A, D–F Variables AB, D, E1, and so forth are defined as integer type.

DEFSNG (Define Single) Statement

Function This statement is used to declare single-precision real type variables.

Formats

DEFSNG alphabetic character-alphabetic character [, alphabetic character-alphabetic character···]

DEFSNG alphabetic character [, alphabetic character···]

Descriptions

1) The DEFSNG statement is used to declare that the variables whose names begin with the specified character or with a character in the specified range of characters and have no type declaration symbols, are single-precision real type.

Alphabetic character-alphabetic character:

The variables whose names begin with a character in the specified range of alphabetic characters are defined as single-precision real type.

Alphabetic character:

The variables whose names begin with the specified alphabetic character are defined as single-precision real type.

2) Variables with type declaration symbols (%, !, #, or $) have the types specified by those symbols.

Example

DEFSNG A, D–F Variables AB, D, E1, and so forth are defined as single-precision real type.

DEFSTR (Define String) Statement

Function This statement is used to declare string variables.

Formats

DEFSTR alphabetic character-alphabetic character [, alphabetic character-alphabetic character···]

DEFSTR alphabetic character [, alphabetic character···]

Descriptions

1) The DEFSTR statement is used to declare that the variables whose names begin with the specified character or with a character in the specified range of characters and have no type declaration symbols, are string variables.

Alphabetic character-alphabetic character:

The variables whose names begin with a character in the specified range of alphabetic characters are defined as string variables.

Alphabetic character:

The variables whose names begin with the specified alphabetic character are defined as string variables.

2) Variables with type declaration symbols (%, !, #, or $) have the types specified by those symbols.

Example DEFSTR A, D–F Variables AB, D, E1, and so forth are defined as string variables.

DELETE Command

Function This command erases program lines.

Formats
DELETE line number
DELETE [begin line number] – end line number

Descriptions

1) The DELETE command deletes the specified program line or specified range of program lines.
   When a line number is specified following this command, only the specified line is deleted.
   When the begin and end line numbers are specified, the program lines between and including the begin and end line numbers are deleted.
   When only an end line number (– end line number) is specified, all program lines between and including the first program line and the specified end program line are deleted.
2) If a period (.) is used instead of a line number, the last execution line is specified. The last execution line refers to the program line which was last executed by BASIC. When program execution stopped due to an error statement, the last execution line is the program line on which the error occurred.
   After the LIST or LLIST command is executed, the last execution line refers to the line specified last.
3) If the DELETE command is used in program mode, the system returns to the Command mode after executing it.

Examples

DELETE 10	Deletes program line 10
DELETE 10–50	Deletes program lines between and including 10 and 50.
DELETE –50	Deletes all the program lines from the first line to line 50.

DIM (Dimension) Statement

Function

This statement specifies the maximum values for array variable subscripts and allocates appropriate storage.

Formats

DIM variable name (max. value of subscript [, max. value of subscript···])
DIM variable name (max. value of subscript [, max. value of subscript···])
[, variable name (max. value of subscript [, max. value of subscript···])
···]

Descriptions

1) The DIM statement specifies the maximum values for array variable subscripts and allocates appropriate storage for data assigned to arrange variables.
2) Array variables specified in a DIM statement may be subscripted from zero to the maximum values of subscripts specified in the statement.
When an array not defined in the DIM statement is used, the subscripts that can be used are from 0 to 10.
3) The maximum value of subscripts may be specified with an integer expression in a range between 0 and the maximum available memory location.
If the maximum value of subscripts exceeds the maximum available memory capacity, an error will occur when the DIM statement is executed for numeric arrays. For string arrays, an error will occur when the DIM statement is executed or when the character string transferred to array variables exceeds the available string area in memory.
For the memory areas available to variables, refer to Chapter 1, Section 3, "CONSTANTS AND VARIABLES."
The size of the unused memory area is returned by the FRE function.
4) up to 255 subscripts (dimensions) are specifiable.
5) It is not possible to redefine an array which is already defined in a DIM statement.
6) All variables, immediately after defined in the Dim statement, have a value of zero for numeric variables, and null (" ") for string variables.
7) Arrays can be deleted by the ERASE or CLEAR statement.
When memory area runs short, unnecessary arrays may be deleted with these statements. When initializing array variables or updating the maximum value of subscripts, temporarily delete the pertinent array, then redefine it with the DIM statement.

Examples

```
DIM A (25, 3)
DIM A(3), B(5, 8, 7)
```

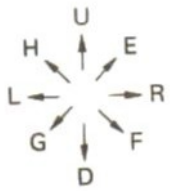
DRAW Statement

Function This statement is used to draw figures on a graphic screen.

Format DRAW string expression

Descriptions

1) The DRAW statement lets the computer draw various graphic figures on the screen using graphic macro commands specified by the string expression. Graphic macro commands are chiefly used for shifting a reference point, which functions like a paint brush.
2) The string expression consists of one or more graphic macro commands.
3) Graphic macro commands include the following:



U distance
- Shifts the reference point (RP) the specified distance upward.

D distance
- Shifts the RP the specified distance downward.

L distance
- Shifts the RP the specified distance to the left.

R distance
- Shifts the RP the specified distance to the right.

E distance
- Shifts the RP diagonally the specified distance to the upper right direction.

F distance
- Shifts the RP diagonally the specified distance to the lower right direction.

G distance
- Shifts the RP diagonally the specified distance to the lower left direction.

H distance
- Shifts the RP diagonally the specified distance to the upper left direction.

M horizontal coordinate, vertical coordinate
- Shifts the RP to the specified coordinates.
- If the horizontal and vertical coordinates are signed (with + or –), they specify relative coordinates.

B
- Shifts the RP without drawing its locus.

N
- Shifts the RP while drawing its locus.
- After being shifted, the RP designates a start point for another locus.

A angle
- Rotates the figures drawn by the U, D, L, R, E, F, G, H, or M (for relative coordinates only) commands in 90 deg. increments.
- The angle is specified with an integer from 0 to 3: 0 for 0 deg., 1 for 90 deg., 2 for 180 deg., and 3 for 270 deg.

C colour code
- Draws figures using the colour specified by the colour code.

S scale factor
- The distance specified in the U, D, L, R, E, F, G, H, and M commands is multiplied by the scale factor value.
- The scale factor can only be an integer from 1 to 255.

X string variable
- Causes the computer to execute the graphic macro commands assigned to the string variable.

= numeric variable
- Allows the use of the numeric variable for distance, angle, colour code, scale factor, and so forth.

END — Statement

Function This statement is used within a program to stop program execution.

Format END

Description The END statement stops program execution, closes open files, and returns the system to Command mode.

EOF (End of File) — Function

Function This function returns a value indicating whether the end of a file has been reached or not.

Format EOF (file number)

Descriptions

1) The EOF function returns the value −1 or 0 to indicate whether or not data to be read into memory remains in the file specified by the file number.
   −1 End of file reached.
   0 End of file not reached.
2) The file specified by the file number must be opened by the OPEN statement for input mode.

ERASE — Statement

Function This statement is used to erase an array or arrays from storage.

Format ERASE array name [, array name···]

Descriptions

1) The ERASE statement causes an array or arrays to be erased from memory.
2) This statement may be used to erase unnecessary arrays from memory to increase available memory space.
3) Erased arrays can be redefined by the DIM statement.

Example

```
ERASE A, B
```

ERL (Error Line) — Function

Function This function returns the number of the program line on which an error occurred.

Format ERL

Descriptions

1) The ERL function is chiefly used in an error service routine specified in the ON ERROR GOTO statement.
2) In an IF statement, this function must be placed on the left side of a relational operator (with a line number placed on the right side of the operator.) The line number in the IF statement is automatically renumbered when the RENUM command is executed, provided the line number is on the right side of the relational operator.
3) If an error occurred when the system is in Direct mode, the function returns a value of 65535.

Examples

```
PRINT ERL
IF ERL = 120 THEN PRINT"ERROR"
```

ERR (Error Number) — Function

Function This function returns an error code.

Format ERR

Descriptions

1) The ERR function is chiefly used in an error service routine specified in the ON ERROR GOTO statement.
2) For error codes, refer to Chapter 3, Section 6, "Error Code Table."

Example

```
PRINT ERR
IF ERR = 4 THEN PRINT"ERROR 4"
```

ERROR — Statement

Function This statement causes the system to generate an error deliberately.

Format ERROR error code

Description The ERROR statement is used to deliberately generate an error for error simulation.

EXP (Exponential) — Function

Function This function returns the power of "e", the base for natural logarithms.

Format EXP (numeric expression)

Descriptions

1) The EXP function returns the power of "e", the base for natural logarithms, using the value of the numeric expression as an exponent.
2) The value of the numeric expression must be in the range from −147.3654459516 to 145.0628605862.
3) The result is always a double-precision real number, regardless of the numeric expression type.

Example PRINT EXP (2.4)

FIX — Function

Function This function returns the integer part of a value.

Format FIX (numeric expression)

Descriptions

1) The FIX function returns the integer part of the value of the numeric expression, by truncating all the decimal places of the value.
2) The FIX and INT functions return different integers for a negative value.
   FIX(−1.3) returns −1
   INT(−1.3) returns −2

FOR Statement

Function

This statement executes a series of instructions within a loop formed by FOR and NEXT statements the specified number of times.

Format

FOR numeric variable = initial value TO final value [STEP increment]

Descriptions

1) The FOR statement must always be followed by a NEXT statement to complete a loop.
2) The FOR-NEXT execution loop functions as follows:
   When the FOR statement is executed, the initial value is assigned to the numeric variable, and each time the NEXT statement in the loop is encountered, the increment is added to the value of the numeric variable.
   When the value of the numeric variable equals or exceeds the final value, control proceeds with the statement following the NEXT statement. If it does not reach the final value, control returns to the statement following the FOR statement.

Example

```
10 FOR I = A TO = -3 * A STEP-1
20 PRINT I
30 NEXT I
40 END
```

3) The initial value, final value and increment are specified with numeric expressions.
   The increment may be a negative value. If it is, control proceeds with the statement following the NEXT statement when the value of the numeric variable is equal to or less than the final value.

Example

```
10 A = 2
20 FOR I = A TO =-3*A STEP - 1
30 PRINT I
40 NEXT I
50 END
```

   If zero is specified for the increment, the FOR-NEXT loop becomes a permanent loop.
   When "STEP increment" is omitted, +1 is assumed for the increment.

Example

```
10 FOR I = 1 TO 10
20 PRINT I
30 NEXT I
40 END
```

4) The FOR-NEXT loop is executed only once in the following cases.
   The increment is a positive value, and the initial value is larger than the final value.
   The increment is a negative value, and the initial value is a smaller than the final value.
5) The numeric variable must be a simple variable. If an array is specified for the numeric variable, an error will result.

6) One FOR-NEXT loop may include another FOR-NEXT loop (nesting). The larger FOR-NEXT loop must completely include the smaller FOR-NEXT loop. If part of the smaller loop is outside the larger loop, an error will result.

Example

Complete nesting

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 3
30 PRINT I, J
40 NEXT J
50 NEXT I
60 END
```

Crossed loop (error)

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 3
30 PRINT I, J
40 NEXT I
50 NEXT J
60 END
```

7) If a CLEAR or MAXFILES statement is executed inside a FOR-NEXT loop, the loop will be discontinued.

FRE — Function

Function This function returns the size of the unused memory area.

Format

FRE (numeric expression)
FRE (string expression)

Descriptions

1) The numeric or string expression is a dummy, and may have any value. The result is returned in bytes.
2) FRE (numeric expression).
   Returns the size of the free area within the user area. The free area is a portion of the user area not used for storing a program or data.
3) FRE (string expression):
   Returns the size of the unused area within the string area. When executing the FRE (string expression), delete unnecessary character strings from the string area to expand the available string area.
4) For memory mapping, refer to Chapter 3, Section 3, "Memory Map."

Examples

```
PRINT FRE (0)
PRINT FRE (" ")
```

GOSUB

Statement

Function This statement calls a subroutine.

Format GOSUB line number

Descriptions This statement allows a branch to the subroutine whose first line is specified by the line number that follows it.
A RETURN statement placed in the subroutine causes control to be returned to the statement following the GOSUB statement in the main program.

<**Subroutine**>

1) A subroutine is an independent set of instructions which is always terminated with a RETURN statement.
2) Subroutines can be called from any location in the main program any number of times to perform the same operation repeatedly.
3) One subroutine can call another subroutine. This is called multiple subroutines in nesting.
   Each time a subroutine is nested, the corresponding memory area (stack) is used. Nesting is allowed as long as stack area is available.
4) More than one RETURN statement may exist in a subroutine.
5) If a CLEAR or MAXFILES statement is executed in a subroutine, the RETURN statement in the subroutine will not be able to return control to the main program.

Example

```
10  GOSUB 100
20  I = 0 : GOSUB 200
30  I = 1 : GOSUB 200
40  END
100 ' Subroutine
110 PRINT "SUB100"
120 RETURN
200 ' Subroutine
210 PRINT "SUB200"
220 IF I = 1 THEN RETURN
230 GOSUB 100
240 RETURN
```

GOTO — Statement

Function This statement creates a branch to the specified line, where execution continues.

Format
GOTO line number
GO TO line number (up to one space allowed between GO and TO.)

Description
1) The GOTO statement creates a branch to the program line specified, where execution will continue.
2) If a GOTO statement is executed in Direct mode, program execution will be initiated from the specified line.
Unlike the RUN command, the GOTO statement does not initialize variables nor close files.

Example
```
10  GOTO 100
20  PRINT "20"
100 PRINT "100"
110 END
```

HEX$ (Hex $) — Function

Function This function converts the value following it to a character string of hexadecimal notation.

Format HEX$ (integer expression)

Description The HEX$ function converts the value of the integer expression to a character string of hex notation.

Example
```
A$ = HEX$(41)
```

IF — Statement

Function This statement is used to make conditional decisions.

Formats
IF numeric expression THEN statement [:statement···] [ELSE statement [:statement···]]
IF numeric expression THEN statement [:statement···] [ELSE line number]
IF numeric expression THEN line number [ELSE statement [:statement···]]
IF numeric expression THEN line number [ELSE line number]
IF numeric expression GOTO line number [ELSE statement [:statement···]]
IF numeric expression GOTO line number [ELSE line number]

Descriptions
1) The IF statement decides the route of programming flow based upon the results of a numeric expression. When the numeric expression is a relational or logical expression, decision is made based upon true or false.
2) When the value of the numeric expression is other than zero or the condition in the relational expression is satisfied (true), the statement that follows the THEN is executed, or the program branches to the line number that follows the THEN or GOTO.

Example

```
10  A = 1 : B = -1
20  IF A = -1 THEN PRINT"A=1"
30  IF B THEN PRINT "B<>0"
40  END
```

3) When the value of the numeric expression is zero or the condition given by the relational expression is not satisfied (false), the statement that follows ELSE is executed or the program branches to the line number specified in the ELSE clause.
   When there is no ELSE clause, control is passed to the next line.

Examples

```
10  A = 1 : B = 0
20  IF A = 0 THEN PRINT"A=0" ELSE PRINT"A<>0"
30  IF B THEN PRINT "B<>0"
40  PRINT "40"
50  END
```

4) The THEN or ELSE clause in an IF statement may contain another IF statement (nesting).
   The number of THEN clauses may differ from the number of ELSE clauses. Each ELSE clause is paired with the nearest THEN clause.

Example

```
IF A$ ="X" THEN IF B$ = "Y" THEN PRINT"Y"ELSE PRINT"X" ELSE 100
```

5) In relational expressions, relational operators (=, <>, <. >, =<, or =>) are used to compare the magnitudes of numeric values or character strings. In logical expressions, logical operators (AND, OR, NOT, etc.) are used to logically correlate more than one relational expression.
   For details on relational and logical expressions, refer to Chapter 1, Section 4, "OPERATIONS."

Example

```
10   INPUT "Y/N/E ?";A$
20   IF A$ = "E" THEN 40
30   IF A$<>"Y" OR A$<>"N" THEN PRINT "Input Miss=".
     ;A$:GOTO 10 ELSE 100
40   PRINT "End"
50   END
100  IF A$="Y" THEN PRINT "Yes":GOTO 10
110  IF A$="N" THEN PRINT"No":GOTO 10
```

INKEY$ Function

Function This function is used to read a character from the keyboard.

Format INKEY$

Descriptions

1) The INKEY$ function returns the character of the depressed key on the keyboard.
   If no key is depressed, it returns a null string (" ").
   Each time a key is pressed, the character of the key is transferred to the computer's keyboard buffer. When the INKEY$ function is executed, it reads the last transferred keyboard character from the buffer.
   Therefore the INKEY$ function may read a character which was transferred to the buffer prior to the execution of the function.
2) If no key has been pressed, the INKEY$, when executed, does not wait for key entry, but passes control to the next statement.
   If you want the computer to wait for a key entry, you have to make a program which tests for a key entry.
3) The INEKY$ function does not display the character of the pressed key on the screen.

Example

```
10  IF INKEY$<>"" THEN 10      Clears the keyboard buffer.
20  K$=INKEY$
30  IF K$="" THEN 20           Waited for a key entry.
40  PRINT " KEY = ";K$
50  IF K$="E" THEN 100
60  GOTO 20
100 END
```

INP (Input) Function

Function This function is used to read data from an I/O port.

Format INP (port address)

Descriptions

1) The INP function reads one byte of data from the port with the specified port address.
2) The port address must be an integer ranging from 0 to 255.
3) For details on port addresses, refer to Chapter 3, Section 5, "I/O Map."

Example

```
10 A = INP(&HA8)
20 A$ = BIN$(A)
30 PRINT RIGHT$("0000000" +A$, 8)
40 END
```

INPUT Statement

Function

This statement is used to input numbers or strings from the keyboard into variables.

Format

INPUT["phrase";] variable [, variable···]

Descriptions

1) The INPUT statement, when executed, prints a phrase, question mark (?) and a blank on the screen, and waits for data (numbers or strings) entry from the keyboard.
2) When information is entered from the keyboard, it is displayed following the blank. When the RETURN key is pressed, the information is transferred to the specified variable.
   The entered information may be edited with the screen edit keys before the RETURN key is pressed.
3) If only the RETURN key is pressed, with no information entered from the keyboard, the value of the variable remains the same as it was before the INPUT statement was executed.
   If the variable is first used in the INPUT statement, however, operation of the RETURN key will assign zero to numeric variables, and a null string (" ") to string variables.
4) Blanks preceding and/or succeeding entered information are ignored.
5) When the phrase (including quotes is omitted, only a question mark (?) and a blank will appear when the INPUT statement is executed.
6) When you wish to input more than one data item into corresponding variables, each data item must be separated by a comma (,).
   When the number of input data items does not match the number of specified variables, the following will happen:
   * When the number of data items is less than the number of variables:
     Two question marks (??) will appear on the screen, prompting continued data entry.
   * When the number of data items exceeds the number of variables:
     The message "Extra ignored" will appear on the screen, signifying that the extra data is ignored, and execution will continue.
7) If a character string is transferred to a numeric variable, the message "Redo from start" will appear on the screen, and the system waits for data reentry from the keyboard.
8) Quotation marks (") may be omitted from a character string to be input to a string variable.
   However, if the character string contains at least one comma, or has blanks at the first or last character position or positions of the string, the string must be enclosed in quotes.
9) If the INPUT statement is executed while in Graphic mode, the system will automatically return to Text mode.

Example

```
10 INPUT "ABC";A$
20 PRINT A$
30 INPUT B
40 PRINT B
50 INPUT C, D, E$
60 PRINT C;D;E$
70 END
```

INPUT#

Statement

Function This statement reads data items from a file and assigns them to program variables.

Format INPUT# file number, variable [, variable ···]

Descriptions

1) The INPUT# statement reads data item (numeric values or character strings) from the file specified by the file number, and assigns them to the specified variables.
2) The file specified by the file number must first be opened by an OPEN statement for input mode.
3) The data items readable with the INPUT# statement are those recorded in the file by the PRINT# statement.
4) The variable types must match the corresponding data item types.

Example

```
INPUT# 1, A, B
```

INPUT$

Function

Function This function returns a string of a specified number of characters read from the keyboard or from a specified file.

Formats

INPUT$ (integer expression, [#] file number)
INPUT$ (integer expression)

Descriptions

1) When INPUT$ (integer expression, [#] file number) is specified:
   The INPUT$ function returns a character string whose length is specified by the integer expression, and which is read from the file specified by the file number.
2) When INPUT$ (integer expression) is specified:
   The function returns a character string whose length is specified by the integer expression, and which is read from the keyboard.
   Once the specified string length is returned, execution proceeds with the next statement without requiring RETURN key operation.
   The character string entered from the keyboard will not appear on the screen.
3) This function can read any character code except the CTRL and STOP key codes. Included in the readable character codes are the CR (&H0D) or LF (&H0A) codes.

Example

```
10 A$ = INPUT$(2)
20 PRINT A$
30 END
```

INSTR (Instring) Function

Function This function returns the location of a specified character in a specified string.

Format INSTR ([integer expression,] string expression 1, string expression 2)

Descriptions

1) The INSTR function searches for the string specified by string expression 2 within the string specified by string expression 1 and returns the location of the first character of the former string.
2) The integer expression specifies the first character location where searching will begin.
   The value of the integer expression must be an integer ranging from 1 to 255. When it is omitted, searching starts with the first character location on string 1.
3) If string 2 was not found in string 1 of the value of the integer expression exceeds the number of characters contained in string 1, the INSTR function will return zero.

Example

```
10 A$ = "ABCDEFGABC"
20 B$ = "FG"
30 A = INSTR(A$, B$)
40 B = INSTR(3, A$, "ABC")
50 PRINT A, B
60 END
```

INT (Integer) Function

Function This function returns an integer which does not exceed the value of a numeric expression.

Format INT (numeric expression)

Descriptions

1) The INT function returns an integer which does not exceed the value of the numeric expression that follows it, by truncating all decimal places of the expression value.
2) For negative values, the INT and FIX functions return different integers. For positive values, they return the same integer, however.
   A = INT(−1.23) −2 is assigned to variable A.
   A = FIX(−1.23) −1 is assigned to variable A.

INTERVAL ON — Statement

Function This statement enables interrupts from an interval time.

Format INTERVAL ON

Description Once the INTERVAL ON statement is executed, the interrupt service routine specified in the ON INTERVAL GOSUB statement is executed each time the interval specified in the same statement elapses.

INTERVAL OFF — Statement

Function This statement disables interrupts from an interval timer.

Format INTERVAL OFF

Description Once the INTERVAL OFF statement is executed, interrupts from an interval timer are disabled (the interrupt service routine specified in the ON INTERVAL GOSUB statement will not be executed when the interval specified in the same statement has elapsed).

INTERVAL STOP — Statement

Function This statement holds up an interrupt from an interval timer.

Format INTERVAL STOP

Descriptions

1) The INTERVAL STOP statement holds up interval timer interrupts (whose interval is specified in the ON INTERVAL GOSUB statement) until the INTERVAL ON statement is executed.
   Once a specified interval elapses after the INTERVAL STOP statement has been executed, a subsequent execution of the INTERVAL ON statement initiates the execution of the interrupt service routine.
2) The INTERVAL STOP statement holds up the execution of the INTERVAL ON statement. If no INTERVAL ON statement has been executed, interrupts are not held up but ignored.

KEY — Command

Function This command designates a character string to any of the function keys.

Format KEY integer expression, string expression.

Descriptions

1) The KEY command assigns the contents of the character string expression to the function key specified by the integer expression.
2) The value of the integer expression must be an integer from 1 to 10, which corresponds to the function key numbers.
3) Up to 15 characters may be used for a string; all extra characters will be ignored.
   When placing a control code in the string expression, it must be preceded by a plus (+) sign and a CHR$ function.

Example

```
KEY 2, "SCREEN 0"+CHR$ (&H0D)
```

KEY LIST — Command

Function This command lists all of the function key designations on the screen.

Format KEY LIST

Description The KEY LIST command lists all of the character strings assigned to the function keys, on a text mode screen.

KEY ON — Statement

Function This statement prints the function key designations at the bottom of the screen.

Format KEY ON

Descriptions

1) The KEY ON statement is used to print the first five characters of strings assigned to function keys, at the bottom of the screen.
2) Normally the key designations for function keys 1 through 5 are printed; when the SHIFT key is pressed and held, those for function keys 6 through 10 are printed.

KEY OFF — Statement

Function This statement turns off the function key designation printout at the bottom of the screen.

Format KEY OFF

Description The KEY OFF statement is used to erase the function key destinations at the bottom of the screen.

KEY (n) ON — Statement

Function This statement is used to enable function key interrupts.

Format KEY (integer expression) ON

Descriptions

1) If the function key specified by the integer expression is pressed after the KEY(n) ON statement has been executed, the program branches to the interrupt service routine specified in the ON KEY GOSUB statement.
2) When KEY(n) ON is valid, all function key designations are ignored.

KEY (n) OFF — Statement

Function This statement is used to disable function key interrupts.

Format KEY (integer expression) OFF

Description Once the KEY(n) OFF statement is executed, no interrupt will occur when the function key specified by the integer expression is pressed.

KEY (n) STOP — Statement

Function This statement is used to hold up function key interrupts.

Format KEY (integer expression) STOP

Descriptions

1) The KEY(n) STOP statement is used to hold up an interrupt caused by pressing the function key specified by the integer expression, until a KEY(n) ON statement is subsequently executed.
   If the specified function key is pressed after the KEY(n) STOP statement has been executed, the interrupt service routine will be executed when the KEY(n) ON statement is subsequently executed.
2) The KEY(n) STOP statement is used to hold up execution of the KEY(N) ON statement. When no KEY(n) ON statement has been executed, interrupts are not held up but ignored.
3) Once the KEY(n) STOP statement is executed, the key designation for the specified function key will be ignored.

LEFT$ — Function

Function This function returns a specified number of characters from a character string, starting with the leftmost character.

Format LEFT$ (string expression, number of characters)

Descriptions

1) The LEFT$ function returns the specified number of characters from the character string specified by the string expression, starting with the leftmost character of the string.
2) When the specified number of characters is equal to or larger than the number of characters in the string, all the characters of the string will be returned.
3) When zero is specified for the number of characters, a null string (" ") will be returned.
4) The graphic character header (&H01) for a graphic symbol is counted as a character. This means that a graphic symbol must be counted as two characters.

Example A$ = LEFT$("ABCDEF",3) "ABC" is assigned to variable A$.

LEN (Length) — Function

Function This function returns the length of a string.

Format LEN (string expression)

Descriptions

1) The LEN function returns the total number of characters in the character string specified by the string expression that follows it.
2) A control code or blank is also counted as a character.
3) The graphic character header (&H01) for graphic symbols is also counted as a character. This means that each graphic symbol must be counted as two characters.

Example

```
10 A$ = "ABC" + CHR$(&H0D)
20 A = LEN(A$) : B = LEN("☺")
30 PRINT A, B
40 END
```

LET — Statement

Function This statement is used to assign the value of an expression to a variable.

Format LET variable = expression
Variable = expression

Descriptions

1) The LET statement assigns the value of the expression to the variable.
2) LET is not necessary.
3) It is not possible to assign the value of a string expression to a numeric variable or the value of a numeric expression to a string variable.

Example

```
10 LET A = 10
20 LET A$ = "ABC"
30 B = 20
40 PRINT A, A$, B
50 END
```

LINE — Statement

Function

This statement is used to draw straight lines or rectangles on a graphic screen.

Formats

LINE [(coordinate X_1, coordinate Y_1)] -(coordinate X_2, coordinate Y_2)[,colour code]

LINE [(coordinate X_1, coordinate Y_1)] -(coordinate X_2, coordinate Y_2),[colour code], B

LINE [(coordinate X_1, coordinate Y_1)] -(coordinate X_2, coordinate Y_2), [colour code], BF

Relative coordinate specifications, STEP (X_1, Y_1) and STEP (X_2, Y_2) are usable in place of coordinates (X_1, Y_1) and (X_2, Y_2), respectively.

Descriptions

1) The LINE statement allows you to draw straight lines or rectangles on a graphic screen, using the colour specified by the colour code.
2) When B or BF are omitted- A straight line is drawn between the two coordinate points (X_1, Y_1) and (X_2, Y_2).
   When B is specified: A rectangle, with its opposite corners located at coordinate points (X_1, Y_1) and (X_2, Y_2), is drawn on the screen.
   When BF is specified: A rectangle, with its opposite corners located at coordinate points (X_1, Y_1) and (X_2, Y_2), is drawn on the screen and the enclosed area is painted with the colour specified by the colour code.
3) When the colour code is omitted, the colour specified in the COLOR statement is assumed.
4) When the begin point (X_1, Y_1) is omitted, the last reference point (LP) is assumed for it.
   If the begin point (X_1, Y_1) is specified with relative coordinates, it refers to the LP as the origin.
   If the end point (X_2, Y_2) is specified with relative coordinates, it refers to the LP as the origin.
5) After the LINE statement is executed, the LP is located at the end point (X_2, Y_2).
6) The X coordinate point may be specified with an integer from 0 to 255, while the Y coordinate point may be specified with an integer from 0 to 191.
   If a coordinate point outside the above ranges is specified, 0 or 255 is assumed for the X coordinate point, and 0 or 191 is assumed for the Y coordinate point.

Example

```
10 SCREEN                          Draws a straight line
20 LINE (10, 10)-(50, 50), 1       Draws a rectangle.
30 LINE (60, 60)-(100, 100), , B   Draws a rectangle and paints it.
```

```
40 LINE (110, 110)-(150, 150), 1, BF   Draws a rectangle and paints it.
50 LINE STEP (10, -40)-STEP (40, 40), 1, BF
                                       Specified with relative coordinates.
60 GOTO 60
```

LINE INPUT

Statement

Function: This statement reads a character string from the keyboard and places it in a string variable.

Format: LINE INPUT ["comment";] string variable.

Descriptions:

1) The LINE INPUT statement prints the comments on the screen and waits for entry of a character string from the keyboard.
2) When a string is typed in, it is printed after the semicolon. Operation of the RETURN key assigns the string to the string variable.
   The printed string can be edited with the screen edit keys before the RETURN key is pressed.
3) If only the RETURN key is pressed, with no string typed in, a null string (" ") will be read into the variable.
4) Unlike the INPUT statement, the LINE INPUT statement prints no question mark (?) following the comment, and transfers commas (,) or quotes (") to the string variable.
5) If the LINE INPUT statement is executed on a graphic screen, correct data will not be read into the variable, and the screen will not be returned to the text mode.

Example:

```
10 LINE INPUT"ABC";A$
20 PRINT A$
30 END
```

LINE INPUT

Statement

Statement: This statement reads a character string from a file and places it in a string variable.

Format: LINE INPUT# file number; string variable.

Descriptions:

1) The LINE INPUT# statement reads a character string from the file specified by the file number. All characters upto the CR (&H0D) and LF (&H0A) codes or only the CR code are assigned to the variable.
2) Only a combination of CR and LF codes, arranged in the order in which they are written, or a CR code is regarded as a delimiter. A combination of LF and CR codes appearing in this order is not regarded as a delimiter.
3) When the characters string read from a file exceeds 254 characters, the first 254 characters are read into the first string variable, and the extra characters are assigned to a second variable.
4) Data written to a file with the PRINT# statement uses CR and LF codes as delimiters, and hence can beread with the LINE INPUT# statement.
5) ASCII files created with the SAVE command contain CR and LF codes as delimiters at the end of each line, and hence may be read as string variables.

LIST Command

Functions: This command displays program lines currently in memory.

Formats:
LIST [line number]
LIST [begin line number] – end line number.
LIST beging line number – [end line number].

Descriptions:

1) The LIST command displays all program lines between and including the begin and end line numbers on a text mode screen.
2) When only LIST is used by itself: Displays all program lines.
   When LIST line number is specified: Displays only the specified program line.
   When LIST –end line number is specified: Displays program lines from the first line of the program up to and including the end line number.
   When LIST begin line number– is specified: Displays program lines from the begin line number through the last line of the program.
3) If a period (.) is used in place of a line number, the last execution line will be displayed on the screen. The last execution line refers to the line which was last executed by BASIC. If program execution has halted due to an error statement, the last execution line is the line on which the error occurred. When the LIST or LLIST statement is executed, the last execution line refers to the line which was last specified.
4) To temporarily stop a list operation, press the STOP key. To resume a list operation, press the STOP key a second time.
5) To suspend a list operation and return to the command mode, simultaneously press the CTRL and STOP keys.

Example:
LIST
LIST 100
LIST 100–200

LLIST Command

Functions: This command lists program lines on a printer.

Formats:
LLIST [line number]
LLIST [begin line number]-end line number
LLIST begin line number – [end line number]

Description: The LLIST command is identical to the LIST command, with the exception that all lines specified are output to an attached printer.

Example: LLIST 100–200

LOAD

Command

Function: This command reads as ASCII form program file into memory.

Format: LOAD"device name [file name]"[, R]

Descriptions:

1) The LOAD command retrieves a program (ASCII file) which was saved by the SAVE command, and reads it into memory.
2) Execution of the LOAD command clears previous programs or variables from memory, and closes open files.
3) When the R option is specified, program execution is automatically started immediately after the program is read into memory. In this case, open files are not closed.
4) When the file name is omitted, the program file which is first found will be read.

Example:

```
LOAD"CAS:SAMPLE"
LOAD"CAS:SAMPLE", R
```

LOCATE

Statement

Function: This statement positions the cursor to the point specified on the screen.

Format:

LOCATE column position
LOCATE [column position], row position
LOCATE [column position], [row position], cursor switch

Descriptions:

1) The LOCATE statement positions the cursor at the specified position on a text mode screen.
2) Column positions range from 0 to 39, 0 being assigned to the leftmost column.
   If a column position exceeding the maximum screen width specified by the SCREEN or WIDTH statement is specified, the position identified by the maximum screen width will be assumed.
   When the column position is omitted, the previous column position is assumed.
3) Row positions range from 0 to 23, 0 being assigned to the top row.
   If a value exceeding the last row position is specified, the last row position will be assumed.
   The last row position is 22 when the function key designations are printed on the screen, and is 23 when they are not on the screen.
   When the row position is omitted, the current row position is assumed.
4) The cursor switch, specified with a numeric expression having a value of 1 or 0, is used to turn the screen cursor on and off.
   When the switch value is zero, no cursor appears on the screen except when the system waits for key entry. When the switch value is one, the cursor always appears on the screen.
   The initial default value is 0.

Example:

```
10 LOCATE 10, 10:INPUT A$
20 LOCATE 10 : PRINT A$
30 END
```

LOG (Logarithm) — Function

Function: This function returns the natural logarithm of a numeric expression.

Format: LOG (numeric expression)

Descriptions:
1) The LOG function returns the natural logarithm (using "e" as its base) of the numeric expression that follows it.
2) The value of the numeric expression must be larger than zero.
3) The result is always returned as a double-precision real number regardless of the numeric expression type.

Example:
```
A=LOG (1.23)
```

LPOS (Printer Head Position) — Function

Function: This function returns the printer head position.

Format: LPOS (expression)

Descriptions:
1) The LPOS function returns the head position of the printer buffer in memory. It does not refer to the physical head position.
2) The expression is a dummy, and may have any value.

Example:
```
A=LPOS (X)
```

LPRINT — Statement

Function: This statement outputs data (numeric values or character strings) to the printer.

Formats:
```
LPRINT
LPRINT expression [;expression ··· ] [ ; ]
LPRINT expression [,expression ···] [ , ]
```

Descriptions:
1) The LPRINT statement outputs the numbers or strings derived from the expressions to an attached printer.
2) When specifying more than one expression in an LPRINT statement, separate them with commas (,) or semicolons (;).
3) When only LPRINT is used, with no expression specified, only a line feed operation will occur when the statement is executed.
4) If the LPRINT statement is not followed by a comma or semicolon, the output data will be followed by a CR code (&H0D) and a LF code (&H0A).
5) The handling of commas, semicolons and data is identical to the PRINT statement, except that the data is sent to the printer.

Example:
```
10 LPRINT "ABC";123;
20 LPRINT "CDE"
30 END
```

LPRINT USING

Statement

Function: This statement is used to output numbers or strings to the printer using a specific format.

Format:
LPRINT USING format string; expression [;expression ···] [;]
LPRINT USING format string; expression [,expression ···] [,]

Descriptions:

1) The LPRINT USING statement outputs the numbers or string derived from the expressions to an attached printer, using a specific format.
2) When specifying more than one expression in an LPRINT USING statement, separate them with commas (,) or semicolons (;).
3) If the LPRINT USING statement is not followed by a comma or semicolon, the output data is terminated with a CR code (&H0D) and a LF code (&H0A).
4) The handling of commas, semicolons and data is identical to the PRINT statement, except that the data is sent to the printer.
5) Format specification is identical to that for the PRINT USING statement. For more details refer to the PRINT USING statement.

Example:

```
10 LPRINT USING "\\ #### ";"ABC";123;
20 LPRINT USING "\\"; "CDE"
30 END
```

MAXFILES

Statement

Function: This statement specifies the maximum number of files.

Format: MAXFILES=integer expression

Descriptions:

1) The MAXFILES statement is used to specify the maximum number of files, as used in an OPEN statement, using an integer expression.
   All files up to the specified maximum number can be opened at a time.
2) The number that can be specified with the integer expression ranges from 0 to 15.
   The initial value of the integer expression is one. If the value of the integer expression is zero, no file can be opened with an OPEN statement.
3) Similar to the CLEAR statement, this statement initializes all variables.
   Numeric variables are initialized to zero; string variables are initialized to null (" ").
   All open files are closed.
   All of the contents defined in the statements beginning with DEF (DEF FN, DEF USR, DEFINT, DEFSNG, DEFDBL, and DEFSTR) are made invalid.
   All arrays are cleared.
   FOR NEXT loops are discontinued.
   Control is not returned from a subroutine by the RETURN statement.
4) When the MAXFILES statement is executed, a file control block specified by (the value of the integer expression + 1) is set aside in memory.
   A space of 267 bytes is set aside for each file control block.

Example: MAXFILES=10

MERGE

Command

Function: This command reads an ASCII program file and merges it with the program currently in memory.

Format: MERGE "device name [file name]"

Descriptions:

1) The MERGE command reads an ASCII program file which was recorded by the SAVE command, and merges it with the program currently in memory.
2) When the file name is omitted, the ASCII program file which is first found will be read.
3) If the program in a file contains the same line number as that in the program currently in memory, the program line in the program file replaces the one in memory.

Example: MERGE "CAS:SAMPLE"

MID$ Function

Function: This function returns the specified part of the given string to the screen.

Format: MID$ (string expression, position [, number of characters])

Descriptions:

1) The MID$ function returns the specified number of characters, beginning with the specified character position, from the string specified by the string expression.
2) The position is specified by an integer expression whose value is 1 through 255.
   If the specified position exceeds the number of strings a null (" ") will be returned.
3) The number of characters is specified with an integer expression whose value ranges from 0 to 255.
   When the number of characters is omitted, the character at the specified position and all the characters to the right, will be returned.
   When zero is specified for the number of characters, a null (" ") will be returned.
4) The graphic character header (&H01) in a graphic symbol is counted as a character. This means that each graphic symbol occupies two character positions.

Example:

```
10 A$="012345689ABCDEF"
20 B$=MID$ (A$, 2, 1) : C$=MID$ (A$, 4)
30 PRINT B$, C$
40 END
```

MID$ Statement

Function: This statement replaces the specified part of the given string with another string.

Format: MID$ (string variable, position [, number of characters])=string expression

Descriptions:

1) The MID$ statement is used to replace a specified number of characters in a specified character string by a string variable having the same number of characters as specified by the string expression, beginning with the specified character position. The number of characters assigned to the string variable must not change before or after replacement.
2) The character position is specified by an integer expression whose value ranges from 1 to 255.
   The character position must not exceed the number of characters assigned to the string variable.
3) The number of characters to be replaced is specified by an integer expression whose value ranges from 0 to 255.
   When the number of characters is omitted or exceeds the number of characters in the string expression, the numbers of characters to be replaced is identical to the number in the string expression.
   When the number of characters to be replaced is smaller than the number of characters in the string expression, the characters in the variable are replaced with the corresponding number of characters in the string expression beginning with the leftmost character of the expression.
4) When the value (position+number of characters−1) exceeds the number of characters in the string variable, the extra characters are ignored.
5) The graphic character header (&H01) for a graphic symbol is counted as a character. This means that each graphic symbol occupies two character positions.

Example:

```
10 A$="012345689"
20 MID$ (A$, 2, 4)="AAAAAAA"
30 PRINT A$
40 END
```

MOTOR Statement

Function: This statement is used to turn the cassette recorder's motor on and off.

Formats:

```
MOTOR ON
MOTOR OFF
MOTOR
```

Descriptions:

1) The MOTOR statement controls motor operation in an attached cassette recorder, which has been placed in the Play or Record mode.
2) MOTOR ON: Starts the motor.
   MOTOR OFF: Stops the motor.
   MOTOR: Stops the motor if it is turning, and starts it if it is stationary.

NEXT — Statement

Function: This statement is used with a FOR statement to form an execution loop in a program.

Format: NEXT [numeric variable [, numeric variable ···]]

Descriptions:

1) The NEXT statement is used with the FOR statement in which the same numeric variable is placed.
2) When more than one numeric variable is placed in this statement, it forms execution loops with the corresponding number of FOR statements. In this case, the numeric variables in the NEXT statement must be arranged so that the first variable corresponds to the nearest FOR statement, the second variable corresponds to the next to the nearest FOR statement, and so forth.
3) When numeric variables are omitted, the NEXT statement is paired with the nearest FOR statement.

Example:

```
10 FOR I=1 TO 5
20 FOR J=1 TO 3
30 FOR K=1 TO 2
40 PRINT I, J, K : NEXT K, J, I
50 FOR X=1 TO 4 : PRINT X
60 NEXT
70 END
```

No cross loop

NEW — Command

Function: This command deletes the program currently in memory and clears all variables.

Format: NEW

Descriptions:

1) The NEW command deletes the program currently in memory and clears all variables.
   Open files are closed.
   The contents of the statements beginning with DEF (DEF FN, DEF USR, DEFINT, DEFSNG, DEFDBL, or DEFSTR) are made invalid.
   All arrays are cleared.
2) The NEW command is often used prior to beginning a program.

OCT$ — Function

Function: This function converts a numeric value into a string of octal notation characters.

Format: OCT$ (integer expression)

Description: The OCT$ function returns a string to the screen that is equal to the octal value of the decimal integer expression that follows it.

Example:

```
10 A$=OCT$ (16)      "20" is assigned to variable A$.
20 PRINT A$
30 END
```

ON ERROR GOTO — Statement

Function: This statement is used to specify the first line number of an error service routine.

Format: ON ERROR GOTO line number

Descriptions:

1) The ON ERROR GOTO statement enables interrupts caused by errors. If an error occurs, it causes execution to branch to the error service routine whose first line number is specified by itself.
   Once this statement is executed, the BASIC system does not take care of the error (no error message appears, nor does the system return to the command mode).
2) An error service routine contains the ERR or ERL functions for error processing, and terminates with a RESUME statement.
   If an error occurs within an error service routine, no interrupt occurs, but an error message appears on the screen and the system returns to the command mode.
3) When an ON ERROR GOTO 0 statement is executed, interrupts caused by errors are disabled, and all errors are taken care of by the BASIC system.
4) Once this statement is executed, error interrupts remain enabled even after program execution is completed. This means that an interrupt caused by an error which occurred after the system returned to the command mode, will cause the specified error service routine to be executed.
   Interrupts enabled by an ON ERROR GOTO statement are not disabled until another ON ERROR GOTO statement, RUN command, or CLEAR statement is executed. To disable error interrupts within the current program, the ON ERROR GOTO 0 statement must be executed before the program is terminated.

Example: ON ERROR GOTO 1000

ON GOSUB

Statement

Function: This statement causes execution to branch to a specified subroutine.

Formats:
ON integer expression GOSUB line number [, line number ···]
ON integer expression GOSUB [[line number], ···] line number

Descriptions:
1) The ON GOSUB statement causes execution to branch to the subroutine with the specified first line number whose position in the statement is specified by the integer expression.
2) When the value of the integer expression is zero or the line number in the specified position is omitted, execution branches to no subroutine, but proceeds with the statement after the ON GOSUB statement.

Example:
ON X GOSUB 100, , 55
When X=1, the subroutine beginning with line 100 is executed.
When X=2, execution proceeds with the next statement.
When X=3, the subroutine beginning with line 55 is executed.
When X=4, execution proceeds with the next statement.

ON GOTO

Statement

Function: This statement creates a branch to a specified line.

Formats:
ON integer expression GOTO line number [, line number ···]
ON integer expression GOTO [[line number], ···] line number

Descriptions:
1) The ON GOTO statement causes execution to branch to the specified line number whose position in the statement is specified by the integer expression.
2) When the value of the integer expression is zero or the line number in the specified position is omitted, execution proceeds with the statement after the ON GOTO statement.

Example:
ON X GOTO 100, , 55
When X=1, execution branches to line 100.
When X=2, execution proceeds with the next statement.
When X=3, execution branches to line 55.
When X=4, execution proceeds with the next statement.

ON INTERVAL GOSUB Statement

Function: This statement specifies the first line of an interval timer interrupt service routine.

Format: ON INTERVAL=interval time GOSUB line number

Descriptions:

1) The ON INTERVAL GOSUB statement is used to specify the time interval at which interrupts are requested from the interval timer, and the first line number of an interval timer interrupt service routine.
   When an interval timer interrupt is enabled by the INTERVAL ON statement, the interval timer requests an interrupt at specified time intervals, causing execution to branch to the interrupt service routine whose first line number is specified in the statement.
2) The time interval can be set in 1/50 second increments, and is specified with an integer expression whose value ranges from 1 to 65535. Interval count down starts when the ON INTERVAL GOSUB statement is executed.
3) A return from the subroutine is accomplished by a RETURN statement placed in the subroutine.
   While the interrupt service routine is being executed, the system is placed in the INTERVAL STOP (interrupt hold) state. When a RETURN statement is executed, the system returns to the INTERVAL ON (interrupt enable) state.
4) While the interrupt service routine specified by any of the following statements is being executed, the system is placed in the INTERVAL STOP (interrupt hold) state until execution of the subroutine is completed:

   ON ERROR GOTO
   ON KEY GOSUB
   ON STOP GOSUB
   ON SPRITE GOSUB
   ON STRIG GOSUB

Example:

```
10 ON INTERVAL=50 GOSUB 100
20 TIME=0
30 INTERVAL ON
40 GOTO 40
100 PRINT TIME/50
110 RETURN
```

ON KEY GOSUB

Statement

Function: This statement specifies the first line number of an interrupt service routine whose execution is initiated by a function key operation.

Formats:

ON KEY GOSUB line number [, line number ···]
ON KEY GOSUB [[line number], ···] line number

Descriptions:

1) The ON KEY GOSUB statement is used to specify the first line number of an interrupt service routine whose execution is initiated by a function key operated after a KEY(n) ON statement has been executed in the same program. Execution branches to the routine with the specified first line number.
2) Line numbers in the statement are arranged in the order of function key numbers, so different line numbers may be specified for each function key. Up to ten line numbers may be specified in this statement by separating them with commas (,).
   When any line number is omitted, the corresponding function key will not cause an interrupt when operated.
3) A returning from an interrupt service routine is accomplished by a RETURN statement placed in the subroutine. While an interrupt service routine is being executed, the system is placed in the KEY(n) STOP (interrupt hold) state. When a RETURN statement is executed the system returns to the KEY(n) ON (interrupt enable) state.
4) While an interrupt service routine caused by an ON ERROR GOTO statement is being executed, the system is placed in the KEY(n) STOP (interrupt hold) state until execution of the subroutine is completed.

Example:

```
10 ON KEY GOSUB 100, , , 200
20 KEY (1) ON
30 KEY (4) ON
40 GOTO 40
100 PRINT "F1"
110 RETURN
200 PRINT "F4"
210 RETURN
```

ON SPRITE GOSUB

Statement

Function: This statement specifies the first line number of a sprite interrupt service routine.

Format: ON SPRITE GOSUB line number

Descriptions:

1) The ON SPRITE GOSUB statement is used to specify the first line number of an interrupt service routine whose execution is initiated when two sprite figures overlap on the screen.
   A sprite interrupt occurs when a sprite figure drawn with the PUT SPRITE statement (after the SPRITE ON statement has been executed) overlaps another sprite figure on the screen, causing execution to branch to the specified interrupt service routine.
2) A return from an interrupt service routine is accomplished by a RETURN statement placed in the subroutine. While an interrupt service routine is being executed, the system is placed in the SPRITE STOP (interrupt hold) state. When a RETURN statement is executed, the system returns to the SPRITE ON (interrupt enable) state.
3) While the interrupt service routine specified by any of the following statements is being executed, the system is placed in the SPRITE STOP (interrupt hold) state until execution of the subroutine is completed:

   ON ERROR GOTO ON KEY GOSUB
   ON STOP GOSUB

Example:

```
10 SCREEN 2
20 ON SPRITE GOSUB 100
30 SPRITE ON
40 SPRITE$( 0 )=STRING$ (8, 255)
50 FOR I=0 TO 300
60 PUT SPRITE 0, (I, 200–I), 1
70 PUT SPRITE 1, (200–I, I), 8, 0
80 NEXT
90 END
100 RETURN 50
```

ON STOP GOSUB

Statement

Function: This statement specifies the first line of an interrupt service routine which is initiated by an interrupt caused by the operation of the CTRL and STOP keys.

Format: ON STOP GOSUB line number

Descriptions:

1) The ON STOP GOSUB statement is used to specify the first line number of an interrupt service routine whose execution is initiated by an interrupt caused by the simultaneous operation of the CTRL and STOP keys.
   If the CTRL and STOP keys are simultaneously pressed after a STOP ON statement has been executed, an interrupt occurs causing execution to branch to an interrupt service routine with the specified first line number.
2) A return from an interrupt service routine is done by a RETURN statement placed in the routine.
   While an interrupt service routine is being executed, the system is placed in the STOP STOP (interrupt hold) state. When a RETURN statement is executed, the system returns to the STOP ON (interrupt enable) state.
3) While the interrupt service routine specified in any of the following statements is being executed, the system is placed in the STOP STOP (interrupt hold) state until execution of the routine is completed:
   ON ERROR GOTO ON KEY GOSUB
4) Erroneous execution of this statement may cause a program runaway, which can be stopped only by turning the computer off. Make sure the programming for this statement is correct.

Example:

```
10 ON STOP GOSUB 100
20 STOP ON
30 K$=INKEY$
40 IF K$="E" THEN 200
50 GOTO 30
100 PRINT "100"
110 RETURN
200 STOP OFF
210 END
```

ON STRIG GOSUB Statement

Function: This statement specifies the first line of an interrupt service routine to which execution branches when an interrupt is caused by operation of the Space bar on the keyboard or a Trigger button on a joystick.

Formats:
ON STRIG GOSUB line number [, line number ···]
ON STRIG GOSUB [[line number] , ···] line number

Descriptions:

1) The ON STRIG GOSUB statement is used to specify the first line number of an interrupt service routine to which execution branches when an interrupt is caused by operation of the Space bar on the keyboard or any of the trigger buttons on a joystick.
   If the Space bar on the keyboard or a Trigger button on a joystick is pressed after a STRIG(n) ON statement has been executed, an interrupt occurs, causing execution to branch to the interrupt service routine whose first line is specified in this statement.
2) Up to five line numbers are specifiable in this statement. Trigger numbers 0 through 4 correspond to line numbers in the order in which they are specified in this statement.
   - 0 Space bar on the keyboard.
   - 1 1st trigger button on the joystick connected to JOYSTICK socket 1.
   - 2 1st trigger button on the joystick connected to JOYSTICK socket 2.
   - 3 2nd trigger button on the joystick connected to JOYSTICK socket 1.
   - 4 2nd trigger button on the joystick connected to JOYSTICK socket 2.

   Line numbers are separated by commas (,).
   If any line number is omitted, the corresponding trigger will not cause an interrupt.
3) A returning from an interrupt service routine is accomplished by a RETURN statement placed in the routine. While an interrupt service routine is being executed, the system is placed in the STRIG(n) STOP (interrupt hold) state. When a RETURN statement is executed, the system returns to the STRIG(n) ON (interrupt enable) state.
4) While the interrupt service routine specified in any of the following statements is being executed, the system is placed in the STRIG(n) STOP (interrupt hold) state until execution of tthe routine is completed:
   ON ERROR GOTO ON KEY GOSUB
   ON STOP GOSUB ON SPRITE GOSUB

Example:

```
10 ON STRIG GOSUB 100, 200, 300
20 STRIG (0) ON : STRIG ( 1 ) ON : STRIG ( 2 ) ON
30 GOTO 30
100 PRINT "SPACE" : RETURN
200 PRINT "JOY 1 " : RETURN
300 PRINT "JOY 2 " : RETURN
```

OPEN Statement

Function: This statement is used to open a file for data I/O.

Format: OPEN "device name [file name] " [FOR mode] AS [#] file number

Descriptions:

1) The OPEN statement opens a file with the specified file name, and assigns the specified file number to it. The file, once opened, may be used for data input or output by using the INPUT# or PRINT# statements. When file access is completed, a file should be closed with the CLOSE statement.
2) The FOR clause is used to specify data input or output:

FOR INPUT	Input mode
FOR OUTPUT	Output mode

3) The file number is specified with an integer ranging from 1 to 15. It must not exceed the value specified in the MAXFILES statement.
It is not allowed to use the file number of an already opened file.
4) The OPEN statement is used for data input/output to or from cassette tape files or for printing characters or text symbols on a graphic screen.

Example:

```
10 SCREEN 3
20 OPEN "GRP: "AS 1
30 PRESET (10, 10)
40 PRINT# 1, "SAMPLE"
50 CLOSE
60 GOTO 60
```

OUT — Statement

Function: This statement is used to output one byte of data to an I/O port.

Format: OUT port address, integer expression

Descriptions:

1) The OUT statement is used to output one byte of data, specified by the integer expression, to the I/O port with the specified port address.
2) The port address is specified with an integer expression whose value ranges from 0 to 255.
3) For details on port addresses, refer to Chapter 3, Section 5, "I/O Map."
4) Erroneous execution of the OUT statement may cause a program runaway which can be stopped only by turning the system off.

PAD — Function

Function: This statement returns the status of an attached touch pad.

Format: PAD (integer expression)

Descriptions:

1) The PAD function returns the status of a touch pad (connected to one of the JOYSTICK sockets) which is specified by the integer expression.
2) The value of the integer expression specifies the JOYSTICK socket to which the touch pad is connected, and the status type to be returned.

0 – 3	Specifies JOYSTICK socket 1.
4 – 7	Specifies JOYSTICK socket 2.
0 or 4	Returns −1 when the touch pad is pressed. Returns 0 when the touch pad is not pressed.
1 or 5	Returns the X coordinate of the pressed point on the touch pad.
2 or 6	Returns the Y coordinate of the pressed point on the touch pad.
3 or 7	Returns −1 when the switch on the touch pad is pressed. Returns 0 when the switch is not pressed.

PAINT — Statement

Function: This statement is used to paint a specified area on the screen.

Formats:
PAINT (X coordinate, Y coordinate) [, paint colour [, boundary colour]]
PAINT (X coordinate, Y coordinate), [paint colour], boundary colour
Reltive coordinates STEP (X, Y) may be used in place of absolute coordinates (X, Y).

Descriptions:

1) The PAINT statement is used to paint the entire area enclosed by boundary lines (of the specified boundary colour(, and in which the specified coordinate point is located, using the specified paint colour.
   When in the high resolution graphic mode, boundary colour is not specifiable. The colour specified for the paint colour is also used for the boundary colour.
2) The boundary and paint colours are both specified with colour codes.
   When the boundary colour is omitted, the colour used for the paint colour is also used for the boundary colour.
   When the paint colour is omitted, the colour specified in the COLOR statement is used for it.
3) If relative coordinates, STEP (X, Y), are used, the coordinate point is relative to the last reference point (LP).
4) If the specified coordinate point is located in the border area on the screen or at a point where the same colour as the boundary colour is specified, painting of the specified area will not occur.

Example:

```
10 SCREEN 2
20 CIRCLE (100, 100), 50, 8
30 PAINT (70, 70), 8
40 GOTO 40
```

PDL (Paddle) — Function

Function: This function returns the current paddle status.

Format: PDL (integer expression)

Descriptions:

1) The PDL function returns the status of the paddle whose number is specified by the integer expression.
2) Up to 12 paddles can be connected to the JOYSTICK sockets on the computer. The JOYSTICK socket to be used is specified with the value of the integer expression:
   When the value is 1, 3, 5, 7, 9, or 11: JOYSTICK socket 1.
   When the value is 2, 4, 6, 8, 10, or 12: JOYSTICK socket 2.
3) The result returned is an integer from 0 to 255.

PEEK — Function

Function: This function returns the contents of a specific memory location.

Format: PEEK (address)

Descriptions:

1) The PEEK function returns the contents of the memory address specified.
2) The address must be specified with an integer expression whose value ranges from &H0 to &HFFFF (−32768 to +32767).
3) For details on the memory, refer to Chapter 3, Section 3, "Memory Map."

Example: X=PEEK (&HFFFF)

PLAY — Statement

Function: This statement is used to play music.

Format: PLAY string expression A [, string expression B [, string expression C]]

Descriptions:

1) The PLAY statement is used to play music according to the music macro commands specified by the string expressions.
2) String expressions A, B, and C specify the music macro commands for audio channels A, B, and C, respectively.
   Example: PLAY "C", "E", "G"
3) Each string expression consists of a string of one or more music macro commands.
   When the value of a string expression is null (" "), no tone will be output on the corresponding channel.
4) The music macro commands include the following.

Commands specifying pitch	A ~ G, #, +, −, O, N
Command specifying notes	L
Command specifying rest	R
Command specifying temp	T
Command specifying loudness	V
Commands specifying tonal colour	S, M

For more details, see Chapter 1, Section 6, SOUND FEATURES.

PLAY

Function

Function: This function returns values indicating whether music is being played or not.

Format: PLAY (integer expression)

Descriptions:

1) The PLAY function returns values indicating whether or not the specified audio channel is playing music using the PLAY statement.
   When the value is −1: Playing.
   When the value is 0: Not playing
2) The value of the integer expression selects the audio channel to be tested:
   0 Tests whether any of channels A, B, or C is playing or not.
   1 Tests channel A.
   2 Tests channel B.
   3 Tests channel C.

Example:

```
10 PLAY "C"
20 PRINT PLAY ( 1 )
30 GOTO 10
```

POINT

Function

Function: This function returns the colour code of a specified coordinate point.

Formats:
POINT (X coordinate, Y coordinate)
POINT STEP (X coordinate, Y coordinate)

Descriptions:

1) The POINT function returns the colour code of the specified coordinate point on a graphic screen.
2) If the coordinate point is specified with a relative coordinate specification, STEP (X, Y), the coordinate point refers to the last reference point (LP) as the origin.
   The LP remains unchanged after the POINT function is executed.

Example:

```
10 SCREEN 2
20 PSET (10, 10), 1
30 A=POINT (10, 10)
40 CIRCLE (10, 10), 50, A
50 GOTO 50
```

POKE — Statement

Function: This statement writes one byte of data into a specific memory location.

Format: POKE address, integer expression

Descriptions:

1) The POKE statement is used to write data, given by the integer expression, into the specified memory address.
2) The address must be specified by an integer expression whose value ranges from &H0 to &HFFFF (−32768 to +32767).
3) For details on memory mapping, refer to Chapter 3, Section 3, "Memory Map."
4) Erroneous execution of this statement can cause a program runaway, from which the system can be recovered only by turning it off.

Example:

```
POKE &HC100, &H1F
```

POS (Cursor Position) — Function

Function: This function returns the current column position of the screen cursor.

Format: POS (expression)

Descriptions:

1) The POS function returns the current column position of the screen cursor on a text mode screen.
   It returns zero when the cursor is located at the leftmost position on the screen.
2) The expression is a dummy, and may have any value.

Example:

```
PRINT POS ( 0 )
```

PRESET (Point Reset) — Statement

Function: This statement is used to update the colour of the specified coordinate point.

Formats:

PRESET (X, Y) [, colour code]
PRESET STEP (X, Y) [, colour code]

Descriptions:

1) The PRESET statement is used to replace the colour of the specified coordinate point with the colour specified by the colour code, on a graphic screen.
2) When the colour code is omitted, the background colour specified in the COLOR statement is specified for the coordinate point.
3) If the coordinate point is specified with a relative coordinate specification, STEP (X, Y), it refers to the last reference point (LP) as the origin.
   After the PRESET statement is executed, the LP is located at the specified coordinate point.

Example:

```
10 SCREEN 2
20 LINE (10, 10) – (100, 100), 1, BF
30 FOR I=0 TO 150
40 PRESET (I, 30)
50 NEXT
60 GOTO 60
```

PRINT Statement

Function: This statement is used to display data on the screen.

Formats:

PRINT
PRINT expression [; expression ···] [;]
PRINT expression [, expression ···] [,]
A question mar (?) is usable in place of PRINT.

Descriptions:

1) The PRINT statement is used to display data (numeric values or strings) specified by the expression on a text mode screen.
2) Data print formats are as follows:
   For a numeric expression: A printed number is preceded by a blank (for a positive number) or a negative sign (−), and is followed by another blank.
   For a string expression: The specified characters are printed left-justified.
3) When only a PRINT statement is specified, with the expression omitted, a single blank row will be printed.
4) More than one expression may be specified in a PRINT statement, by separating them with commas (,) or semicolons (;).
   Data print format differs depending on the delimiter used:
   - When commas are used as delimiters:
     Rows are divided into display fields, each 14 columns long, and every data item is printed left-justified in a separate display field. If a data item contains more than 14 characters (including a blank as the last location of a numeric data item), it continues into the next display field.
   - When semicolons are used as delimiters:
     One data item is immediately followed by another data item. For numeric data, each data item is preceded by a blank or negative sign (−) and is followed by a blank.
5) Whether a PRINT statement is followed by a comma, semicolon, or nothing affects the location of data printed by the next PRINT or PRINT USING statement:
   - When followed by nothing:
     A line feed operation occurs, at the end of the first PRINT statement, and data for the subsequent PRINT statement is printed in the next row.
   - When followed by a comma: No line feed operation occurs, and data for the subsequent PRINT statement is printed in the next display field and subsequent fields on the same row.
   - When followed by a semicolon:
     No line feed operation occurs, and data for the first PRINT statement is immediately followed by that for the next PRINT statement.

Example:

```
10 PRINT 123, "ABC"
20 PRINT 123; "ABC"; 123, 4: "A"; "B"
30 PRINT 123; 4, 5, 6, 78;
40 PRINT "ABC"
50 END
```

PRINT USING Statement

Function: This statement is used to display data on a text mode screen by using specific print formats.

Formats: PRINT USING format control string; expression [; expression ···] [;]
PRINT USING format control string expression [, expression ···] [,]
A question mark (?) may be used in place of PRINT.

Descriptions:

1) The PRINT USING statement is used to print data (numeric values or strings) given by the expressions on a text mode screen, using the print format specified by the format control string.
2) More than one expression may be specified in a PRINT USING statement, by separating them with commas (,) or semicolons (;). However, the format control string must always be separated from the following expression by a semicolon.
   Display fields are specified by the format control string, and do not depend on the delimiter type (comma or semicolon) used.
3) Whether a PRINT USING statement is followed by a comma, semicolon, or nothing affects the location of the data printed by the next PRINT or PRINT USING statement:
   - When followed by nothing:
     A line feed opration occurs, and data for the subsequent PRINT or PRINT USING statement is printed in the next row.
   - When followed by a comma or semicolon: No line feed operation occurs, and data for the first PRINT USING statement is immediately followed by that for the next PRINT USING or PRINT statement.
4) While the format control string may be specified with string variables or string constants, different control symbols must be used for different types of data to be printed:

   Control symbols for numeric data:

#	(sharp)	.	(period)
+	(plus)	−	(minus)
*	(asterisk)	£	(Pound symbol)
^	(hat)		

   Control symbols for string data:

!	(exclamation mark)	\	(back slash)
&	(ampersand)		

   If any character other than the format control symbols listed above is used in a format control string, that character will be printed either preceding or succeeding the printed data.
   Example: PRINT USING "X=### Y=###"; 12;456

5) When a numeric data item to be printed contains an integer part having a number of digits exceeding that specified by the format control string, the printed data will be preceded by a per cent (%) symbol.

Example: PRINT USING "##"; 123

(Printout): %123

Format control String for numeric data	Description
#···#	Each sharp (#) corresponds to a digit of the number to be printed. When the number of sharps exceeds that of data digits, the data is right justified when printed. Example: PRINT USING " ####"; 123,456
#···#·#···#	A period (.) specifies the position of the decimal point in printed data. The number of decimal places to be printed corresonds to the number of sharps (#) specified following the period (.). When the number of sharps for the decimal places exceeds the number of decimal places in the printed data, the remaining decimal places in the data will be filled with zeros. Example: PRINT USING "##·##"; 1.234; 5.6
#···# #···#,#···# .#···#	When a comma (,) is placed in #···#, any position of a string of sharps (#) for the integer part, the integer part of the printed data is delimited by commas (,) at three-digit intervals. Example: PRINT USING " ##########, #". 123456
+#···# +#···#,#···#	Printed data is preceded by a sign (+ or −). Example: PRINT USING " ###·##"; 12.3; −4,567
#···#+ #···#·#···#+	Printed data is followed by a sign (+ or −). Example: PRINT USING " ###·##+"; 12.3;−4,567
#···#− #···#·#···#−	Negative numbers in printed data are followed by a minus sign (−). Example: PRINT USING " ###·##−"; 12.3; −4,567
**#···# **#···#·#···#	When the integer digits of a data item to be printed are less than the number of sharps in the integer part, the remaining significant Example: PRINT USING "**#### "; 12; 123456

Format control	Description
££#···#	When a format control string is preceded by two Pound signs (££), the printed data is preceded by a single Pound sign (£).
££#···#·#···#	Example: PRINT USING "££#### "; 12; 12345
**£#···#	When a format control string is preceded by two asterisks and one Pound sign (**£), printed data is preceded by a Pound sign (£), and the first vacant digits in the integer part, if any, are filled with asterisks.
**£#···#·#···#	Example: PRINT USING "**£#### "; 12; 12345
#···#^^^^	When a format control string is followed by four hats (^), the data printed is expressed as a floating point number using an E symbol.
#···#·#···#^^^^	Example: PRINT USING " ###·###^^^^"; 1.2345678
String for string data	
!	Prints the leftmost character of string data. Example: PRINT USING " ! "; "ABCDE"
\ ⌣ \ n blanks	Prints characters whose number is equal to the number of blanks enclosed in two "back slash" marks (\ ) plus two. When the length of the string to be printed exceeds the number of specified digits, the extra characters in the string will not be printed. When it is less than the number of specified digits, the remaining digits will be filled with blanks. Example: PRINT USING " \\ "; "AB"; "ABCDEFG"; "ABC"
&	When ampersands (&) are specified, all the specified strings are printed. Example: PRINT USING "&###&###"; "ABC="; 1; "−X+"; 456

PRINT

Statement

This statement is used to output data to a specific file.

Function:

PRINT# file number
PRINT# file number, expression [; expression ···] [;]
PRINT# file number, expression [, expression ···] [,]
A question mar (?) may be used in place of PRINT.

Descriptions:

1) The PRINT# statement is used to output data (numbers or strints) specified by expressions to a file specified by the file number.
2) The file number must specify a file which was opend by an OPEN statement for output mode.
3) The output formats are identical to those for the PRINT statement:
   A numeric data item is preceded by a blank (for a positive value) or a minus sign (−) (for a negative value), and the numerals are converted to a string, which is followed by a blank.
   For a string, all the characters in the string are output.
4) More than one expression may be specified in a PRINT# statement, by separating them with commas (,) or semicolons (;).
   The data output format depends on the delimiter type used, and is identical to that for the PRINT statement, except that the PRINT# statement outputs data to a file.
   When numeric data items are separated by commas, they are put into fields, and each data item may be followed by blanks.
   When strings are separated by semicolons, they are continuously output to a file. When the file is read by an INPUT# or LINE INPUT# statement, the strings are put into a single string.
   If you wish to separate one string from another in the string data read with the INPUT# statement, separate them with commas when outputting with the PRINT# statement.
   If you wish the same for string data read with the LINE INPUT# statement, separate each string with CR (&H0D) and LF (&H0A) codes when outputting with the PRINT# statement.
5) When the PRINT# statement is followed by no comma or semicolon at its end, the output data will be followed by a CR (&H0D) and a LF (&H0A) codes.
   When a comma or semicolon is placed at the end of a PRINT# statement, no CR or LF code will be output. The output format is identical to that of the PRINT statement, except that the PRINT# statement outputs data not to the screen but to a file.

Exampels: Outputting data to a cassette tape file:

```
10 OPEN "CAS:SAMPEL" FOR OUTPUT AS 1
20 PRINT#1, 12.3; 4.56
30 PRINT#1, "ABC"; "DEF"; ","; "GH;"
40 PRINT#  "IJK"
50 CLOSE
60 END
```

Inputting data from the cassette tape file:

```
10 OPEN "CAS:SAMPLE "FOR INPUT AS 1
20 INPUT#1, A, B:PRINT A, B
30 INPUT#1, C$:PRINT C$
40 LINE INPUT#1, D$:PRINT D$
50 END
```

PRINT # USING — Statement

Function: This statement outputs data (numbers or strings) to a specified file, using a specific format.

Formats: PRINT# file number USING format control string;
expression [; expression ···] [;]

PRINT# file number USING format control string;
expression [, expression ···] [,]

A question mark (?) may be used in place of PRINT#.

Descriptions:

1) The PRINT# USING statement is used to output the data (numbers or strings) specified by the expressions to the file specified by the file number, using a specified format.
2) The data output format is identical to that for the PRINT# statement, except that the PRINT# USING statement requires format specification.
3) The format specifications are identical to that for the PRINT USING statement.

Example:

```
PRINT # 1 USING " \  \#### "; "ABC"; 123
```

PSET (Point Set) Statement

Function: This statement is used to draw a point at a specified position on the display screen.

Formats:
PSET (X coordinate, Y coordinate) [, colour code]
PSET STEP (X coordinate, Y coordinate) [, colour code]

Descriptions:
1) The PSET statement is used to draw a point at a specified coordinate point on a graphic screen, in the colour specified by the colour code.
2) When the colour code is omitted, the colour specified in the COLOR statement is assumed.
3) When the relative coordinate specification, STEP (X, Y), is used, it refers to the last reference point (LP) as the origin.
When the PSET statement is executed, the LP is relocated to the specified coordinate point.

Example:
```
10 SCREEN 2
20 FOR I=0 TO 255 STEP 2
30 PSET (I, 50), 1
40 NEXT
50 GOTO 50
```

PUT SPRITE

Statement

Function: This statement is used to draw a sprite figure on the screen.

Formats: PUT SPRITE sprite screen number, (X, Y) [, colour code
[, sprite figure number]]

PUT SPRITE sprite screen number, (X, Y), [colour code], Sprite figure number

The relative coordinate specification, STEP (X, Y), may be used in place of the absolute coordinate specification, (X, Y).

Descriptions:

1) The PUT SPRITE statement is used to draw a sprite figure defined in the SPRITE$ statement, at a specified coordinate point on a sprite screen with the specified sprite screen number.
2) The sprite screen number is specified with an integer from 0 to 31.
   Up to 32 sprite screens are available, each assigned one of the numbers from 0 to 31.
   Only one sprite figure can be drawn on each sprite screen, and up to 32 sprite figures (screens) can be shown on the display at a time.
   However, the number of sprite figures that can be drawin in each row is limited to four.

   A sprite screen with a smaller number has a higher priority; a sprite screen with a larger number is hidden behind the one with a smaller number when overlapped.
3) The X coordinate is specified with an integer expression whose value is in the range from −32 to 255; Y coordinate is specified with another integer expression whose value is in the range from −32 to 191.
   When the coordinate point is specified with relative coordinates, STEP (X, Y), it refers to the last reference point (LP) as the origin.
   Once the PUT SPRITE statement is executed, the LP is relocated to the specified coordinate point.
4) To clear a sprite figures from the screen:
   Specify 208 for the Y coordinate. The sprite figure on the sprite screen with the specified screen number will be cleared, as well as all the sprites on the higher screens.
   If you specify 209 for the Y coordinate, and only the sprite figure on the sprite screen with the specified screen number is cleared.
5) When the colour code is omitted, the current display colour is assumed.
6) Sprite figure numbers correspond to those defined in the SPRITES$ statement.
   When the sprite figure number is omitted, the number of the sprite figure currently shown on the specified sprite screen will be assumed. The initial value of the sprite figure number is equal to the specified sprite screen number.

Example:

```
10 SPRITE$ (0)=STRING$ (8, 255)
20 PUT SPRITE 1, (100, 100), 1, 0
30 PUT SPRITE 0, (105, 105), , 8
40 END
```

READ — Statement

Function: This statement is used to read a value from a DATA statement and assign it to a variable.

Format: READ variable [, variable ···]

Descriptions:

1) The READ statement is used to read constants from a DATA statement in sequential order, and assigns them to a variable or variables.
2) The variable type in the READ statement must match the constant type in the DATA statement from which data is to be read.
3) When the number of variables in the READ statement exceeds the number of constants in the DATA statement, the constants in the next DATA statement are assigned to the extra variables in the READ statement.
   If there is no next DATA statement, an error will result.
4) When the number of variables in the READ statement is less than the number of constants in the DATA statement, the extra constants in the DATA statement will be assigned to variables in the next READ statement.
5) The first READ statement in a program reads data from the first DATA statement in the program (with the smallest line number).
   The RESTORE statement allows the user to specify the DATA statement from which the READ statement begins to read data.

Example:

```
10 DATA ABC, 123, 4.56
20 DATA 7.89, DE, "G Y"
30 READ X$, X:PRINT X$; X
40 READ Y, Z, Y$, Z$:PRINT Y; Z; Y$; Z$
50 END
```

REM (Remark) — Statement

Function: This statement is used to explain the reason for a portion of a program.

Formats:

```
REM [string]
  , [string]
```

Descriptions:

1) The REM statement is not executed. Remarks that follow the REM statement are not displayed or acted upon during a program run.
2) This statement is used to explain the reason for a portion of a program.

Example:

```
10 REM *** SAMPLE PROGRAM
20 GOTO 100
100 ' CHECK ROUTINE
110 END
```

RENUM (Renumber) — Command

Function: This command is used to renumber program lines.

Formats:
RENUM [new line number [, old line number [, incrment]]]
RENUM [new line number], , increment

Descriptions:
1) The RENUM command is used to renumber an old line number and all subsequent line numbers into a new numbering sequence beginning with the new line number and having the specified increment.
2) When only RENUM is used by itself:
   It automatically numbers the lines in increments of 10, beginning with the first line of the program.
   When the new line number is omitted:
   Renumbering begins with line number 10.
   When the old line number is omitted:
   Renumbering begins with the first line of the program.
   When the increment is omitted:
   Lines are numbered in increments of 10.
3) This command is unable to make branches in the program.
4) When executed, line numbers specified in GOTO, GOSUB, ON GOTO, ON GOSUB, and ERL statements are also renumbered.

Example:
```
RENUM
RENUM 1000, , 100
```

RESTORE — Statement

Function: This statement is used to specify the line number of a DATA statement from which data is to be first read by a READ statement.

Format: RESTORE [line number]

Descriptions:
1) A READ statement placed after a RESTORE statement reads data from the DATA statement whose line number is specified in the RESTORE statement.
2) When the line number is omitted, the READ statement starts reading data from the DATA statement with the smallest line number.

Example:
```
10  RESTORE 130: READ A, B:PRINT A;B
20  READ C, D, E:PRINT C;D;E
30  RESTORE :READ A$:PRINT A$
40  END
100 DATA LINE 100
120 DATA 1.23
130 DATA 130
140 DATA 140, 10, 20, 30, 40, 50
```

RESUME Statement

Function: This statement is placed in an error service routine to return execution to the main program.

Formats:
RESUME [0]
RESUME NEXT
RESUME line number

Descriptions:

1) The RESUME statement is used in an error service routine specified by an ON ERROR GOTO statement. When it is executed, execution returns to the main program.
2) When RESUME alone or RESUME 0 is specified:
   Execution returns to the line where the error occurred, and the same statement is retried.
   When RESUME NEXT is specified:
   Execution returns to the statement after the statement where the error occurred.
   When "RESUME line number" is specified:
   Execution returns to the line with the specified line number.

RETURN Statement

Function: This statement is used to return from a subroutine.

Format: RETURN [line number]

Descriptions:

1) The RETURN statement is used in a subroutine specified by a GOSUB or ON GOSUB statement, or in an interrupt service routine, to return execution to a program line specified by the line number.
2) When the line number is omitted, execution returns to the statement after the GOSUB statement.
3) More than one RETURN statement may exist in a subroutine.

Example:

```
10   X=1 : GOSUB 100
20   X=2 : GOSUB 100
30   END
100  PRINT X
110  IF X=2 THEN RETURN
120  RETURN 20
```

RIGHT$ Function

Function: This function returns a desired number of characters from a character string, starting with the rightmost position of the string.

Format: RIGHT$ (string expression, number of characters)

Descriptions:

1) The RIGHT$ function returns the specified number of characters from the specified character string, starting with the rightmost position of the string.
2) When the specified number of characters exceeds the number of characters in the string, all the characters in the string will be returned.
3) When the specified number of characters is zero, a null (" ") will be returned.
4) The graphic character header (&H01) in a graphic symbol is counted as a character. This means that each graphic symbol occupies two character positions.

Example:

```
10 A$="ABCDF"
20 B$=RIGHT$ (A$, 3)
30 PRINT B$
40 END
```

RND (Random) Function

Function: This function returns a random number between 0 and 1.

Format: RND (numeric expression)

Descriptions:

1) The RND function returns a random number between (but not including) 0 and 1.
2) When the value of the numeric expression is positive:
   The same sequence of random numbers is generated each time the program is run.
   When the value of the numeric expression is 0:
   The previous generated random number is returned.
   When the value of the numeric expression is negative:
   Different sequences of random numbers are generated depending on the value of the expression.
3) A positive value is normally used for the value of the numeric expression. So that, the same sequence of random values is generated each time the program is run, unless the random number generator is receded.
   To change the sequence of random numbers, execute the X=RND(−TIME) function to select another sequence, then execute the RND (positive value). The variable X is a dummy, and may be any number.

Example:

```
10 X=RND (-TIME)
20 A=RND (1) :A=INT (20*A) +5
30 PRINT A; :LOCATE A:PRINT "*"
40 GOTO 20
```

RUN — Command

Function: This command begins program execution.

Format: RUN (line number)

Descriptions:

1) The RUN command is used to begin program execution from the specified line number.
   When this command is executed, all variables are cleared, and all open files are closed.
2) When the line number is omitted, execution starts with the first line of the program.

Examples:

```
RUN
RUN 1000
```

SAVE — Command

Function: This command is used to save an ASCII form program to a specified file.

Format: SAVE "device name [file name]"

Descriptions:

1) The SAVE command is used to save an ASCII code program to the file specified by the file name on the device specified by the device name.
2) An ASCII code program is a memory image program in which program lines are stored in the form of ASCII character codes. It is characterized by the following:
   A file space larger than that required for the CSAVE command.
   Programs to be merged with the MERGE command must both be in ASCII form.
   Since an ASCII program file can be treated as a data file, each program line can be assigned to a variable by using the LINE INPUT# statement.
   By CR (&H0D) and LF (&H0A) codes at the end of each line of the ASCII program.
   An end code (&H1A) at the end of the file.
3) When saving to a cassette tape file, the data transfer rate may be specified with the SCREEN statement.

Example:

```
SAVE "CAS:SAMPLE"
```

SCREEN Statement

Function: This statement is used to specify screen modes, sprite figure sizes, key click tone, data transfer rates for cassette files, and printer specifications.

Formats:
SCREEN screen mode
SCREEN [screen mode], sprite size
SCREEN [screen mode], [sprite size], key click switch
SCREEN [screen mode], [sprite size], [key click switch], cassette baud rate
SCREEN [screen mode], [sprite size], [key click switch], [cassette baud rate], printer switch

Descriptions:

1) Screen mode
Specifies the display screen modes. For more details on screen modes, see Chapter 1, Section 5, "SCREEN SETTING."
Screen mode is specified with an integer expression whose value is from 0 to 3:
   - 0 Text mode of 40 columns x 24 rows (initial value is 37 columns x 24 rows.)
   - 1 Text mode of 32 columns x 24 rows (initial value is 29 columns x 24 rows.)
   - 2 High-resolution graphic mode
   - 3 Multicolour mode

2) Sprite size
Specifies the sizes of sprite figures defined in the SPRITE$ variable, and the display magnification for the sprite figures displayed by the PUT SPRITE statement. Sprite size is specified with an integer expression whose value is from 0 to 3:
   - 0 Displays sprite cells in a 8 x 8 dot configuration as they are (8 x 8).
   - 1 Magnifies 8 x 8 dot sprite figures into double size (16 x 16 dots) when printing on the screen.
   - 2 Displays sprite cells in a 16 x 16 dot configurarion as they are (16 x 16).
   - 3 Magnifies 16 x 16 dot sprite figures into double size (32 x 32 dots) when printing on the screen.

3) Key click switch
Turns on and off the echo-back key click tone.
The switch is specified with an integer expression whose value is 0 or 1:
   - 0 Key click switch is off.
   - 1 Key click switch is on.

4) Cassette baud rate
Specifies the baud rate at which data is transferred to a cassette tape file by a CSAVE, BSAVE or SAVE command, or a PRINT# statement.
When a file is read from cassette, the baud rate is automatically set, and need not be set in the SCREEN statement.
Cassette baud rate is specified with an integer expression whose value is 1 or 2:
   - 1 1200 baud
   - 2 2400 baud

6) Printer switch

Specifies whether an attached printer is an MSX printer or no.

Printer switch is specified with an integer expression whose value is 0 or 1:

0 MSX type printer attached.

1 Non-MSX type printer attached.

SGN — Function

Function: This function returns the polarity of a value.

Format: SGN (numeric expression)

Description: The SGN function returns the polarity of the value of the numeric expression that follows it, with the integers −1, 0, and 1.

When −1	The value is negative.
When 0	The value is zero.
When 1	The value is positive.

Example:

```
10 A=SGN (-1.34)
20 B=SGN (0)
30 C=SGN (4.56)
40 PRINT A;B;C
50 END
```

SIN — Function

Function: This function returns the trigonometric sine of a number.

Format: SIN (numeric expression)

Descriptions:

1) The SIN function calculates the trigonometric sine of the value of the numeric expression that follows it.
2) The value of the numeric expression must be in radians.
3) The result is always returned as a double-precision real number regardless of the numeric expression type.

SOUND

Statement

Function: This statement is used to load values into registers of the programmable sound generator (PSG).

Format: SOUND register number, integer expression

Descriptions:

1) The SOUND statement is used to load the value of the numeric expression into the PSG register with the specified register number.
   Combinations of more than one SOUND statement provides for various sound effects which are not available with the PLAY statement alone.
2) Register number is specified with an integer expression whose value is from 0 to 13.
   The PSG has 16 registers, from which 14 registers may be used to load values.

Register No.	Function	Bit							
		b7	b6	b5	b4	b3	b2	b1	b0
0	Frequency on CH.A	FT (A)							
1						CT (A)			
2	Frequency on CH.B	FT (B)							
3						CT (B)			
4	Frequency on CH.C	FT (C)							
5						CT (C)			
6	Noise frequency				NP				
7	Output channel select			Noise			Tone		
		1	0	C	B	A	C	B	A
8	Loudness on CH.A				M	L (A)			
9	Loudness on CH.B				M	L (B)			
10	Loudness on CH.C				M	L (C)			
11	Envelope period	FT (E)							
12		CT (E)							
13	Envelope pattern					EP			

SPACE $ — Function

Function: This function returns a specified length of blanks.

Format: SPACE$ (integer expression)

Descriptions:
1) The SPACE$ function returns a null string whose length is specified by the integer expression that follow it.
2) The value of the integer expression must be an integer from 0 to 255.

Example:
```
A$=SPACES (5)
PRINT "AB"; SPACES$ (5); "CD"
```

SPC (Space) — Function

Function: This function outputs a null string of specified length to the screen or printer.

Format: SPC (integer expression)

Descriptions:
1) The SPC function is used in output statements, such as LPRINT or PRINT, to output a null string whose length is specified by the integer expression that follows it.
2) This function is only used in output statements, and cannot be used in assignment statements, such as LET.
3) The value of the integer expression must be an integer from 0 to 255.

Example:
```
PRINT "AB"; SPC (5); "CD"
```

SPRITE ON — Statement

Function This statement is used to enable sprite interrupts.

Format SPRITE ON

Description The SPRITE ON statement is used to enable interrupts caused by sprite figure clashes on the screen.
If a sprite figure clash occurs on the screen after the SPRITE ON statement has been executed, an interrupt occurs, causing execution to branch to the interrupt service routine specified in the ON SPRITE GOSUB statement.

SPRITE OFF — Statement

Function This statement is used to disable sprite interrupts.

Format SPRITE OFF

Description After the SPRITE OFF statement is executed, no interrupt will occur if sprite figure clashes occur on the screen.

SPRITE STOP — Statement

Function This statement is used to hold sprite interrupts.

Format SPRITE STOP

Description

1) The SPRITE STOP statement is used to hold interrupts caused by sprite figure clashes, until the SPRITE ON statement is subsequently executed. If a sprite figure clash occurs on the screen after the SPRITE STOP statement has been executed, an interrupt will occur when the SPRITE ON statement is subsequently executed, causing execution to branch to the interrupt service routine specified in the ON SPRITE GOSUB statement.
2) If no SPRITE ON statement has been executed before the SPRITE STOP statement is executed, interrupts are not held but ignored.

SPRITE$ System Variable

Function This system variable is used to define a sprite figure.

Format SPRITE$ (integer expression) = string expression

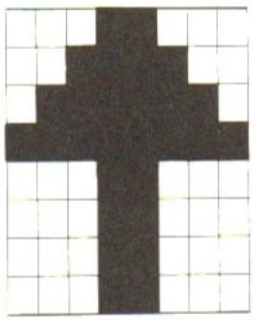
Descriptions

1) The SPRITE$ variable is used to define the sprite figure specified by the string expression for the sprite whose number is specified by the integer expression.
2) The range of the integer expression value differs depending on the sprite figure size specified in the SCREEN statement

Sprite size in SCREEN statement	Figure size	Integer expression range
0 or 1	8 x 8 points	0 ∿ 255
2 or 3	16 x 16 points	0 ∿ 32

3) The string expression uses 8 characters to specify a sprite figure with a size of 8 x 8 points, and 32 characters for one with a size of 16 x 16 points.

- String expression for 8 x 8 point size:
  Each row (comprised of 8 points) of a sprite figure matrix is represented by a bit pattern.
  i.e. a screen point is turned on with a set bit, and is turned off with a reset bit.
  Eight characters are used to represent an entire sprite figure matrix.



CHR$ (&B00011000)
CHR$ (&B00111100)
CHR$ (&B01111110)
CHR$ (&B11111111)
CHR$ (&B00011000)
CHRS (&B00011000)
CHR$ (&B00011000)
CHR$ (&B00011000)

- String expression for 16 x 16 point size:
  A sprite figure is divided into four sections of 8 x 8 points each, and these sections are arranged as shown in the following figures. Since the matrix in each section is represented by 8 characters, 32 characters are required to represent the four matrix sections.

Example

```
10  SCREEN 1, 1
20  P$ = " "
30  FOR I = 0 TO 7
40  READ D$
50  P$ = P$ + CHR$ (VAL (" &B" + D$)
60  NEXT
70  SPRITE$ (0) = P$
80  PUT SPRITE 0, (50, 50), 1
90  END
100 DATA 00011000
110 DATA 00111100
120 DATA 01111110
130 DATA 11111111
140 DATA 00011000
150 DATA 00011000
160 DATA 00011000
170 DATA 00011000
```

SQR (Square Root) — Function

Function This function returns the square root of a number.

Format SQR (numeric expression)

Descriptions
1) The SQR function returns the square root of the value of the numeric expression that follows it.
2) The value of the numeric expression must be larger than zero.
3) The result is always returned as a double-precision real number regardless of the numeric expression type.

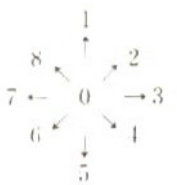
Example
```
A = SQR(4)
```

STICK — Function

Function This function returns the direction of joystick operation.

Format STICK (integer expression)

Descriptions
1) The STICK function returns the direction of operation of the joystick specified by the integer expression, or which cursor control key is being pressed.
2) The value of the integer expression is 0, 1, or 2, specifying the joystick or cursor control key to be tested-
   0 Cursor control key or keys on the keyboard.
   1 Joystick connected to JOYSTICK socket 1
   2 Joystick connected to JOYSTICK socket 2
3) The returned result is an integer from 1 to 8, representing the direction of operation. When the joystick is not operated, the function returns zero.



When two cursor control keys with orthogonal directions are simultaneously operated, the function returns a value indicating a diagonal direction of operation.

For instance, simultaneous operation of the ⇧ and ⇨ keys causes the function to return 2.

Example
```
10 A = STICK (0) : B = STICK (1)
20 PRINT A ; B
30 GOTO 10
```

STOP

Statement

Function

This statement stops program execution and returns the system to the command mode.

Format

STOP

Descriptions

1) The STOP statement is used to stop program execution and returns the system to the command mode, with the following message shown on the screen:

   BREAK in nnnn
   (line number)

2) Unlike the END statement, the STOP statement does not close open files.
3) Program execution stopped by a STOP statement can be resumed with the CONT statement.

Example

```
10 PRINT "A"
20 STOP
```

STOP ON — Statement

Function This statement is used to enable an interrupt caused by the simultaneous operation of the CTRL and STOP keys.

Format STOP ON

Description After the STOP ON statement is executed, an interrupt occurs when the CTRL and STOP keys are simultaneously pressed, causing execution to branch to the interrupt service routine specified in the ON STOP GOSUB statement.

STOP OFF — Statement

Function This statement is used to disable an interrupt caused by the simultaneous operation of the CTRL and STOP keys.

Format STOP OFF

Description After the STOP OFF statement is executed, simultaneous operation of the CTRL and STOP keys will not cause an interrupt.

STOP STOP — Statement

Function This statement is used to hold an interrupt caused by the simultaneous operation of the CTRL and STOP keys.

Format STOP STOP

Descriptions

1) The STOP STOP statement is used to hold an interrupt caused by the simultaneous operation of the CTRL and STOP keys, until the STOP ON statement is subsequently executed.
   If the CTRL and STOP keys are simultaneously operated after the STOP STOP statement has been executed, an interrupt will occur when the STOP ON statement is subsequently executed, causing execution to branch to the interrupt service routine specified in the ON STOP GOSUB statement.
2) If no STOP ON statement has been executed before the STOP STOP statement is executed, interrupts will not be held but ignored.

STRIG (Stick Trigger) — Function

Function This function returns values indicating whether a trigger button on the joysticks is operated or not.

Format STRIG (integer expression)

Descriptions

1) The STRIG function returns values indicating whether the trigger button on the joystick, or the Space bar on the keyboard, specified by the integer expression, is pressed or not.
2) The value of the integer expression can only be from 0 to 4, and specifies the Space bar on the keyboard or one of the trigger buttons on a joystick connected to either of the JOYSTICK sockets:
   - 0 Space bar on the keyboard
   - 1 1st trigger button on the joystick connected to JOYSTICK socket 1
   - 2 1st trigger button on the joystick connected to JOYSTICK socket 2.
   - 3 2nd trigger button on the joystick connected to JOYSTICK socket 1.
   - 4 2nd trigger button on the joystick connected to JOYSTICK socket 2.
3) The returned result is the integer −1 or 0.
   When −1 the button is pressed
   When 0 the button is not pressed.

Example

```
10 A = STRIG (0) : B = STRIG (1)
20 PRINT A ; B
30 GOTO 10
```

STRIG (n) ON

Statement

Function This statement is used to enable interrupts caused by trigger button operations on a joystick.

Format STRIG (integer expression) ON

Descriptions

1) If any of the trigger buttons on the joystick specified by the integer expression, or the Space bar on the keyboard, is pressed after the STRIG(n) ON statement is executed, an interrupt occurs causing execution to branch to the interrupt service routine specified to branch to the interrupt service routine specified by the ON STRIG GOSUB statement.
2) The meanings of the integer expression value are the same as those described for the STRIG function.

STRIG (n) OFF

Statement

Function This statement is used to disable interrupts caused by trigger button operations on a joystick.

Format STRIG (integer expression) OFF

Descriptions

1) The STRIG(n) OFF statement is used to disable interrupts caused by the operation of a trigger button on a joystick or the Space bar on the keyboard. After the STRIG(n) OFF statement is executed, no interrupt will occur if a trigger button on the joystick specified by the integer expression, or the Space bar on the keyboard, is operated.
2) The meanings of the integer expression value are the same as those described for the STRIG function.

STRIG (n) STOP

Statement

Function This statement is used to hold an interrupt caused by a trigger button operation on a joystick.

Format STRIG (integer expression) STOP

Descriptions

1) The STRIG(n) STOP statement is used to hold an interrupt caused by the operation of a trigger button on the joystick specified by the integer expression, or of the Space bar on the keyboard, until a STRIG(n) ON statement is subsequently executed.
   If a trigger button on the specified joystick, or the Space bar on the keyboard, is pressed after the STRIG(n) STOP statement has been executed, an interrupt will occur when the STRIG(n) ON statement is subsequently executed, causing execution to branch to the interrupt service routine.
2) If no STRIG(n) ON statement has been executed before the STRIG(n) STOP statement is executed, interrupts will not be held but ignored.
3) The meanings of the integer expression value are the same as those described for the STRIG function.

STR$ — Function

Function This function converts a number into a string.

Format STR$ (numeric expression)

Descriptions
1) The STR$ function returns a string representing the decimal value of the numeric expression that follows it.
2) The numeric expression may be of any type.

Example
```
A$ = STR$ (123.45)
B$ = STR$ (&HFF)
```

STRING$ — Function

Function This function returns a specified length of a string of the same character which is specified.

Formats
```
STRING$ (integer expression, string expression)
STRING$ (integer expression, character code)
```

Descriptions
1) The STRING$ function returns a string (whose length is specified by the integer expression) of the first character of the string expression or of the character given by the character code.
2) The value of the integer expression must be from 0 to 255.
3) Only the first character of the string expression is meaningful; the second and all remaining characters are ignored.
   A graphic character header (&H01) in a graphic symbol is counted as a character. So if a graphic symbol is placed in the first position of the string expression, a string of the character whose character code is &H01 will be returned.

Examples
```
A$ = STRING$ (10, "ABC")
A$ = STRING$ (10, &H41)
```

SWAP — Statement

Function This statement exchanges the values of two variables.

Format SWAP variable 1, variable 2

Descriptions
1) The SWAP statement exchanges the value of variable 1 with that of variable 2.
2) The type of variable 1 must match that of variable 2.
   It is not possible to exchange values between a numeric variable and a string variable.
   For numeric variables, values can be exchanged between two variables of the same type: integer type, single-precision real type, or double-precision real type.

Examples
```
SWAP A,B
SWAP A%, B%
SWAP A$, B$
SWAP A(5), B
```

TAB — Function

Function This function returns, to the screen or printer, blanks to the specified column position.

Format TAB (integer expression)

Descriptions
1) The TAB function is used in output statements, such as PRINT or LPRINT to output blanks to the screen or printer, beginning with the current cursor position to the specified column position.
2) This function is not usable in assignment statements such as LET.

Example
```
PRINT" ABC " ; TAB (10) ; "CDE"
```

TAN (Tangent) — Function

Function This function returns the trigonometric tangent of a number.

Format TAN (numeric expression)

Descriptions
1) The TAN function returns the trigonometric tangent of the value of the numeric expression that follows it.
2) The numeric expression value must be in radians.
3) The result is always returned as a double-precision real number regardless of the numeric expression type.

TIME — System Variable

Function This variable is used to return, or set up the value of the internal clock (interval timer).

Format TIME

Descriptions
1) The TIME variable is assigned the value of an internal interval timer, which is incremented by one at approximately 1/50 second intervals.
2) The interval timer can be preset by assigning the desired value to the TIME variable.
3) The timer stops while data is being input from or output to a cassette file. Since the timer counts interrupts in approximately 1/50 second intervals from the VDP, it stops when I/O operation is performed to a cassette file, during which VDP interrupts are disabled.

Examples
```
A=TIME
TIME=0
```

TRON (Trace On) — Command

Function This command is used to trace program execution step-by-step.

Format TRON

Descriptions
1) The TRON command causes the program line just executed to be printed on a text mode screen, with the line number of that program line enclosed in square brackets [ ].
2) To clear the Trace mode, execute the TROFF or NEW command.

Example
```
TRON
```

TROFF (Trace OFF) — Command

Function This command is used to clear the Trace mode.

Format TROFF

Description The TROFF command is used to clear the trace mode which was selected with the TRON command.

USR (User) — Function

Function This function causes a machine language subroutine to be executed.

Format USR [number] (argument)

Descriptions
1) The USR function causes a machine language subroutine to be executed from the execution start address defined in the DEFUSR statement.
2) The number is an integer from 0 to 9, which corresponds to the number defined in the DEFUSR statement. When no number is specified, zero is assumed for it.
3) The argument must be specified. So a dummy argument must be used if no argument transfer is needed.
4) Erroneous execution of this function may cause a program runaway, from which the system can be recovered only by turning it off.

Examples
```
A = USR ("ABC")
B = USR2 (0)
```

VAL (Value) — Function

Function This function returns the numeric value of a string.

Format VAL (string expression)

Descriptions
1) The VAL function returns the numeric value of a string specified by the string expression that follows it.
2) The string may be binary (&B), octal (&O), or hexadecimal (&H), as long as it represents a numeric value.
3) For a non-numeric string, this function returns a zero.

Examples
```
A = VAL ("-1.234")
B = VAL ("&B1010")
C = VAL ("&H" + "1F")
```

VARPTR (Variable Pointer) — Function

Function This function returns the first address of a variable area or file control block.

Formats
```
VARPTR (variable)
VARPTR (#file number)
```

Descriptions
1) When a variable is specified, the VARPTR function returns the first address of the variable area in which the value assigned to that variable is stored.
2) When a #file number is specified, the VARPTR function returns the first address of the file control block for the file which is specified by the file number.

Examples
```
A = VARPTR (X)
B = VARPTR (#1)
```

VDP (VDP) — System Variable

Function This variable is used to load values into the registers of the video display processor (VDP), or returns register values.

Format VDP (integer expression)

Descriptions

1) The VDP variable is used to load a value into the VDP register whose number is specified by the integer expression, or to return the value of that register.
2) The value of the integer expression is from 0 to 8, and is used to specify one of the VDP registers.
3) Erroneous use of this variable may cause an abnormal screen. Before using this variable, refer to the MSX system hardware description to fully acquaint yourself with the VDP.

Examples

```
A = VDP (0)
VDP (7) = 1
```

VPEEK (Video Peek) — Function

Function This function returns the contents of the video RAM (VRAM).

Format VPEEK (address)

Descriptions

1) The VPEEK function returns the contents of the specified address in VRAM.
2) The address is specified with an integer expression whose value is from &H0 to &H3FFF.

Example

```
A = VPEEK (&H1000)
```

VPOKE (Video Poke) — Statement

Function This statement is used to write one byte of data into a specific location in video RAM (VRAM).

Format VPOKE address, integer expression

Descriptions

1) The VPOKE statement is used to write one byte of data, specified by the integer expression, into the specified address in VRAM.
2) The address is specified with an integer expression whose value is from &H0 to &H3FFF.
3) The value of the integer expression must be from 0 to 255.
4) Erroneous use of this statement may cause an abnormal screen. Before using this statement, refer to the MSX system hardware description to fully acquaint yourself with the VDP.

Example

```
VPOKE &H1803, &H41
```

WAIT Statement

Function This statement causes the system to wait until an input port has a specified value.

Format WAIT port address, integer expression 1 [, integer expression 2]

Descriptions

1) The WAIT statement causes the system to wait for data input until the data whose bit pattern is specified by integer expressions 1 and 2 is input from the port with the specified port address.
2) The system reads the value of the specified input port, and XOR's the input value with the value of integer expression 2. It then AND's the result of the XOR with the value of integer expression 1. If the result of the AND operation is −1 (true), execution proceeds with the next statement. If the result is 0 (false), the system again reads a value from the same input port.
3) When integer expression 2 is omitted, zero is assumed for it.
4) Erroneous use of this statement may cause the system to ignore data from all input ports other than the specified port, from which state the system can be recovered only by turning it off.
   For details on port addressing, refer to Chapter 3, Section 5, "I/O Map."

WIDTH Statement

Function This statement is used to specify the number of characters per row to be printed on a text mode screen.

Format WIDTH integer expression

Descriptions

1) The WIDTH statement is used to set the width of a row on a text mode screen.
2) The specifiable range of the integer expression differs depending on the screen mode:
   40 x 24 Text mode from 1 to 40
   32 x 24 Text mode from 1 to 32
3) Either end of a frame may overflow your display screen depending on the display type used. In such a case, use the WIDTH statement to reduce screen width.
4) The initial screen widths are as follows:
   40 x 24 Text mode: 37 columns
   32 x 24 Text mode: 29 columns

Example WIDTH 20

CHAPTER 3

REFERENCE

1. CHARACTER CODE TABLE

Character codes are represented by the hex integers from &H00 to &HFF, to which the characters and symbols listed in the following table are assigned.
The character code of a given character is represented by a two-digit hexadecimal number composed of the most significant 4 bits and least significant 4 bits.

Most significant 4 bits →

Least significant 4 bits ↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			Blank (Space)	ø	@	P	`	p	C	É	á	Ã			α	
1			!	1	A	Q	a	q	ü	æ	í	ã			β	±
2		INS	"	2	B	R	b	r	é	Æ	ó	Ĩ			Γ	≥
3			#	3	C	S	c	s	â	ô	ú	ĩ			Π	≤
4			$	4	D	T	d	t	ä	ö	ñ	Õ			Σ	⌠
5			%	5	E	U	e	u	à	ò	Ñ	õ			σ	⌡
6			&	6	F	V	f	v	å	û	a	Ũ			μ	÷
7	BL		´	7	G	W	g	w	ç	ù	º	ũ			τ	≈
8	BS	SELECT	(	8	H	X	h	x	ê	ÿ	¿	⌐		△	Φ	°
9	TAB		)	9	I	Y	i	y	ë	Ö	⌐	ij		╪	Θ	•
A	LF		*	:	J	Z	j	z	è	Ü	¬	¾		ω	Ω	•
B	HOME	ESC	+	;	K	[	k	{	ï	¢	½	~			δ	√
C	CLS	→	,	<	L	\	l	¦	î	£	¼	◇			∞	n
D	CR	←	−	=	M	]	m	}	ì	¥	¡	‰			φ	²
E		↑	.	>	N	^	n	~	Ä	Pt	«	¶T			∈	∎
F		↓	/	?	O	_	o	Blank (DEL)	Å	ƒ	»	§			∩	Blank (FF)

← Hex numbers

↑ Hex numbers

With the graphic character header

	4	5
0	Blank (NULL)	
1	☺	
2	☻	
3	♥	
4	♦	
5	♣	
6	♠	
7	•	
8	◘	
9	○	
A	◙	
B	♂	
C	♀	
D	♪	
E	♫	
F	☼	

Graphic character header (hex &H01)

Characters preceded by a graphic character header are used to represent graphic symbols.

(Example): Execution of PRINT CHR$(1); CHR$(&H41) prints a ☺ on the screen.
Execution of PRINT CHR$(&H41) prints "A" on the screen.

When a graphic symbol is entered from the keyboard, it is automatically preceded by a graphic character header, and the resulting character code consists of two bytes.

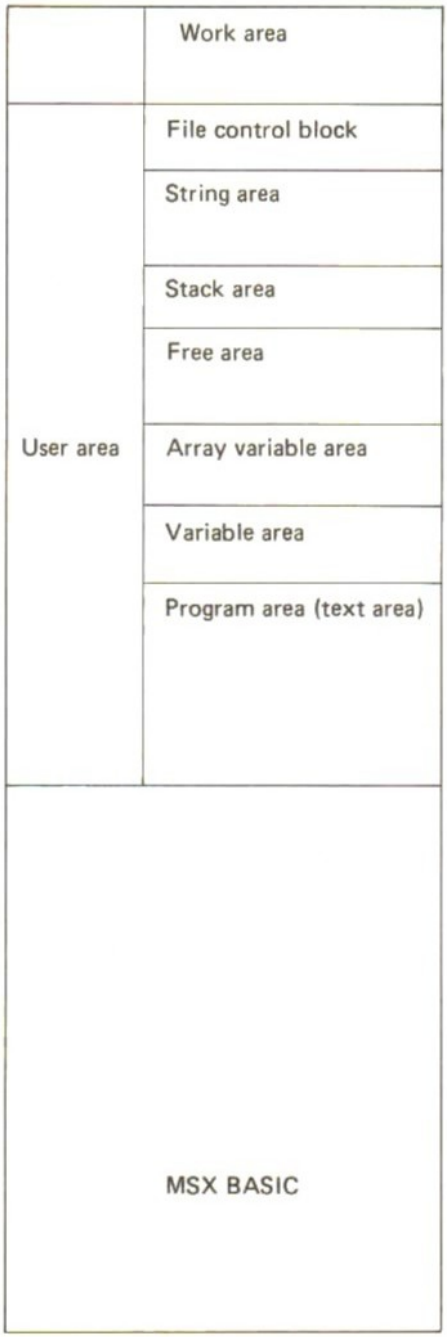
Control Codes

Character codes with their high order 4 bits representing a value of 0 or 1 are used for cursor or screen control, and their character equivalents are not shown on the screen.

(Example): Execution of PRINT CHR$(&H0C) clears the screen. Control codes are also generated when a data key plus the CTRL key, or a control key, are pressed.

Control code	Abbrev.	Key	Description
01	–	CTRL and A	• Generates a graphic character header.
02	–	CTRL and B	• Repositions the cursor to the first location of the preceding data item.
03	–	CTRL and C , or CTRL and STOP	• Stops program execution.
05	–	CTRL and E	• Deletes characters from the current cursor position and all the remaining positions to the right.
06	–	CTRL and F	• Repositions the cursor to the first location of the next data item.
07	BL	CTRL and G	• Sounds the buzzer.
08	BS	CTRL and H or BS	• Deletes the character from the location just preceding the current cursor position.
09	TAB	CTRL and I or TAB	• Causes the cursor to shift at eight position intervals.
0A	LF	CTRL and J	• Performs a line feed operation (repositions the cursor to the first location of the next row).
0B	HOME	CTRL and K or HOME	• Repositions the cursor to the first location of the first row.
0C	CLS	CTRL and L or SHIFT and HOME	• Clears the entire screen.
0D	CR	CTRL and M or RETURN	• Performs a carriage return (end of entry operation).
0E	–	CTRL and N	• Repositions the cursor to the bottom row.
12	INS	CTRL and R or INS	• Inserts a character.
15	–	CTRL and U	• Deletes a line
1C	→	CTRL and \ or ⇨	• Shifts the cursor to the right.
1D	←	CTRL and] or ⇦	• Shifts the cursor to the left.
1E	↑	CTRL and ^ or ⇧	• Shifts the cursor upward.
1F	↓	CTRL and SHIFT and _ or ⇩	• Shifts the cursor downward.

2. MEMORY MAP



- Work area
  BASIC'S work area.

- User area
  This area is available to the user for storage of user programs. User area addresses can be set at and below &HF380 by the CLEAR statement.

- File control block
  This area is used for file I/O. The size of this block corresponds to the number of files specified in the MAXFILES statement.

- String area
  This area is used for storing strings assigned to string variables. The size of the area is specified with the CLEAR statement. The initial size is 200 bytes.

- Stack area
  This area stores the return addresses for the FOR-NEXT, GOSUB, or other branch statements.

- Free area
  Unused area, whose size can be returned by the FRE function.

- Array variable area
  This area stores data assigned to array variables. For string array variables, the pointers that point to string data stored in the string area are stored in this area. This area is set aside in memory when the DIM statement is executed or an array subscripted with 10 or less is first used.

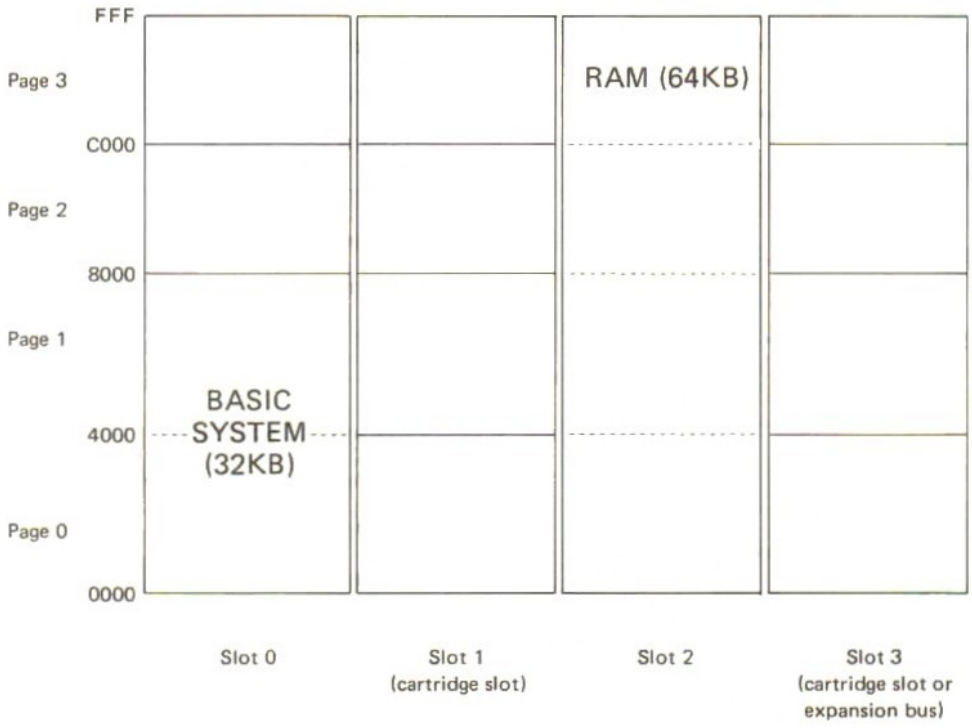
- Variable area
  This area stores data assigned to variables.
  For string variables, the pointers that point to string data stored in the string area are stored in this area.

- Program area
  This area stores BASIC programs.

3. SLOTS

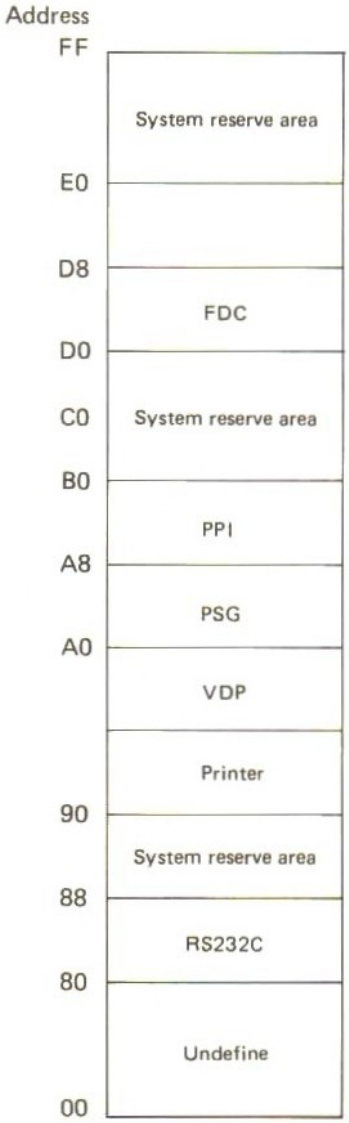
On MSX systems, more than 64K bytes of memory space is available by adding additional slots. There are four standard slots in an MSX system, and up to four expansion slots can be attached to each of the standard slots.
Memory space is divided into pages of 16K bytes each, and a slot is assigned to each page.
Slots are automatically selected by BASIC at the time of power on. When you wish to select a specific slot, you will have to use a machine language program. For more details, read the books on MSX software specifications.

Standard Slots



In the Toshiba Home Computer HX-10, slots 1 and 3 are available to the user, and slots 0 and 2 are used for the system and RAM.
Pages 0 and 1 of slot 0 are used as the BASIC system area, and pages 2 and 3 of slot 2 are used for BASIC RAM: a total of 32 KB.

4. I/O MAP



I/O Address	R/W	Contents	Remarks
AB	W	Mode set	PPI
AA	W	Data write to port C	
	R	Data read from port C	
A9	W	Data write to port B	
	R	Data read from port B	
A8	W	Data write to port A	
	R	Data read from port A	
A2	R	Data read	PSG
A1	W	Data write	
A0	W	Register select	
99	W	Command and address set	VDP
	R	Status read	
98	W	Data write to VRAM	
	R	Data read from VRAM	
91	W	Print data	Latch output
90	W	Strobe output (bit 0)	Latch output
	R	Status input (bit 1)	

W: Write R: Read

For more details, read the books on MSX hardware and software.

5. ERROR CODE LIST

Error message	Error No.	Description
Bad file name	56	File name is wrong. • Wrong or no mode specification in the OPEN statement.
Bad file number	52	File name is wrong. • A file number larger than that specified in the MAX-FILES statement was specified. • A file number not specified in the OPEN statement was specified.
Can't continue	17	Program execution cannot be resumed. • Resume after execution break due to an error. • Resume after program modification.
Device I/O error	19	Data transfer error occurred during communication with an I/O device. • Cassette file read error. • Communication with an I/O device was forcibly stopped by operating the CTRL and STOP keys.
Direct statement in file	57	A direct mode statement exists in an ASCII file being loaded.
Division by zero	11	Division by zero was attempted. • Divider is zero. • Divider is an undefined variable.
File already open	54	File is already open.
File not found	53	The file was not found.
File not open	59	The file is not open.
Illegal direct	12	Execution of a non-direct statement was attempted.
Illegal function call	5	A statement or function is called in an illegal way. • The argument in a statemet or function exceeds the specified range. • An array is subscripted by negative numbers or by unreasonably large numbers.
Internal error	51	An error occurred within BASIC. • Normally this type of error will not occur. Should it occur, temporarily turn off the system.
Input past end	55	File data read was attempted with the INPUT# statement after all data of that file has been read. • The number of variables in the INPUT# statement exceeds the number of data items. • Read operation was attempted to a file where no data exists. (This error can be recovered by using the EOF function.)
Missing operand	24	Necessary operand is missing. • The number of operands is wrong. • Periods (.) are used as separators for operands, instead of commas (,).

Error message	Error No.	Description
NEXT without FOR	1	The number of NEXT statements does not correspond to that of FOR statements. • A FOR-NEXT loop contains part of another FOR-NEXT loop.
No RESUME	21	No RESUME statement exists in an error service routine. • Execution returned from an error service routine by using a GOTO statement.
Out of DATA	4	There is no data to be read by the READ statement. • The number of data items is insufficient. • Wrong line number in the RESTORE statement. • Illegal use of delimiters in the DATA statement.
Out of memory	7	Insufficient memory capacity. • Program is too large. • Too many variables. • Arrays too long. • Unnecessary array variables are not erased by the ERASE statement.
Out of string space	14	String space is insufficient. • String space set by the CLEAR statement is too small.
Overflow	6	Numeric value exceeds the allowable range. • The result of an arithmetic operation is too large or too small.
Re-dimensioned array	10	An array was doublely defined. • An already dimensioned array was re-dimensioned by the DIM statement. • An array subscripted by 10 or less was used without dimensioning, then was subsequently dimensioned by the DIM statement.
RESUME without error	22	A RESUME statement was executed in a portion of a program other than an error service routine.
RETURN without error	3	A RETURN statement was executed before a GOSUB statement was executed. • Execution branched to a subroutine by a GOTO statement. • No END statement was defined at the end of the main program, and the following subroutine was executed.
String formula too complex	16	String is too complex. • Operations for the string written on a line is too complex (too many parentheses).
String too long	15	String is too long. • An attempt was made to assign more than 256 characters to a string variable.
Subscript out of range	9	The subscript for an array exceeds the allowable range. • Subscript is too large. • An undimensioned array is subscripted by a number exceeding 10.

Error message	Error No.	Description
Syntax error	2	Syntax does not match the MSX BASIC syntax. • Wrong entry due to typing error. • Illegal delimiter (comma, period, colon, semicolon, etc.) • Mismatched parentheses. • Variable name beginning with a character other than an alphabetic character.
Type mismatch	13	Variable type mismatch. • An attempt was made to assign a string to a numeric variable. • An argument type in a function does not match.
Undefined line number	8	Wrong line number designation. • Line number specified in a GOTO, GOSUB, RESTORE, or RESUME statement does not exist.
Undefined user function	18	User function is not defined. • Wrong function name in the DEF FN statement. • No DEF FN statement has been executed.
Verify error	20	Verify error. • Disagreement between the program in memory and that read from cassette was found during the execution of the CLOAD? command. • Program saved from a system with different RAM capacity was verified with the program in the current system. (An error will occur if the program contents are normal.)

FUNCTIONAL DESCRIPTION

Function	Description	Command	
Programming	Deletes the program	NEW	88
	Generates line numbers automatically	AUTO	45
	Renumbers all lines	RENUM	111
	Deletes a line	DELETE	60
	Displays the program on the screen	LIST	81
	Prints the program listing on the printer	LLIST	81
	Executes the program	RUN	114
	Restores execution of the program	CONT	54
	Starts tracing	TRON	130
	Stops tracing	TROFE	130
Cassette Recorder	Saves the program	CSAVE	55
		SAVE	114
	Loads the program	CLOAD	51
		LOAD	82
	Verifies the program	CLOAD?	51
	Merges programs	MERGE	85
	Saves the machine language program	BSAVE	47
	Loads the machine language program	BLOAD	47
	Controls the motor	MOTOR ON	87
		MOTOR OFF	87
	Specifies the transfer rate	SCREEN	115
	Specifies the number of files	MAXFILES	85
	Opens a file	OPEN	96
	Outputs data	PRINT #	106
		PRINT # USING	107
	Inputs data	INPUT #	73
		LINE INPUT #	80
		INPUT$	73
	Determines whether EOF has been reached or not	EOF	63
	Closes the file	CLOSE	52
	Determines the file control block address	VARPTR	131
Keyboard	Inputs data	INPUT	72
		LINE INPUT	80
		INKEY$	71
		INPUT$	73
	Controls the key clicking sound	SCREEN	115

Function	Description	Command	
Keyboard	Defines a function key	KEY	76
	Lists the function key contents	KEY LIST	76
	Displays the function key contents	KEY ON	76
	Deletes the function key contents display	KEY OFF	76
	Controls the function key interrupt	KEY (n) ON	77
		KEY (n) OFF	77
		KEY (n) STOP	77
		ON KEY GOSUB	92
	Controls the [CTRL] and [STOP] key interrupt	STOP ON	125
		STOP OFF	125
		STOP STOP	125
		ON STOP GOSUB	94
	Determines which cursor key is pressed	STICK	123
	Determines if the space key is pressed or not	STRIG	126
	Controls the space key interrupt	STRIG (n) ON	127
		STRIG (n) OFF	127
		STRIG (n) STOP	127
		ON STRIG GOSUB	95
Screen Control	Sets the screen mode	SCREEN	115
	Specifies the colour	COLOR	53
	Clears the screen	CLS	52
	Writes data into the VDP register	VDP	132
	Determines the contents of the VDP register	VDP	132
	Detrmines the start address of the video RAM table	BASE	46
	Determines contents of the video RAM	VPEEK	132
	Writes data into the video RAM	VPOKE	132
Text Screen Mode	Displays data	PRINT	102
	Displays formatted data	PRINT USING	103
	Displays the program	LIST	81
	Specifies the line width (number of characters)	WIDTH	133
Text Screen Mode	Outputs blanks	TAB	129
		SPC	119
	Moves the cursor	LOCATE	82
	Determines the vertical position (line) of the cursor	CSRLIN	55
	Determines the horizontal position (column) of the cursor	POS	101
Graphic Screen Mode	Draws circles and ellipses	CIRCLE	49
	Draws straight lines and rectangles	LINE	79
	Draws graphics	DRAW	62
	Fills in colour	PAINT	98
	Draws dots	PSET	108
	Changes the colour of the dots	PRESET	101
	Determines the colour code of the specified point	POINT	100

Function	Description	Command	
Graphic Screen Mode	Displays characters and numbers	MAXFILES	85
		OPEN	96
		PRINT #	106
		PRINT # USING	107
		CLOSE	52
Sprite Screen	Defines the sprite pattern	SPRITE$	121
	Displays the sprite pattern	PUT SPRITE	109
	Controls the sprite pattern interrupt	SPRITE ON	120
		SPRITE OFF	120
		SPRITES STOP	120
		ON SPRITE GOSUB	93
Printer	Prints data	LPRINT	83
	Formatted data	LPRINT USING	84
	Prints the program listing	LLIST	81
	Outputs blanks	TAB	129
		SPC	119
	Specifies whether the MSX printer is used or not	SCREEN	115
	Determines the position of the printer head	LPOS	83
Sound	Plays music	PLAY	99
	Determines whether music is being played or not	PLAY	100
	Writes data into the PSG register	SOUND	118
	Sounds the buzzer	BEEP	46
Joystick, etc.	Determines the direction of the joystick	STICK	123
	Determines whether the trigger button is pressed or not.	STRIG	126
	Controls the joystick trigger interrupt	STRIG ON	127
		STRIG OFF	127
		STRIG STOP	127
		ON STRIG GOSUB	93
	Determines the condition of the touch pad	PAD	97
	Determines the condition of the paddle	PDL	98
I/O Port	Outputs data	OUT	97
	Determines the value of the input port	INP	71
	Waits until the specified value is input	WAIT	133
File	Specifies the number of files	MAXFILES	85
	Opens the file	OPEN	96
	Outputs data	PRINT #	106
		PRINT # USING	107

Function	Description	Command	
	Inputs data	INPUT #	73
		LINE INPUT #	80
		INPUT$	73
	Determines whether EOF has been reached or not	EOF	63
	Closes the file	CLOSE	52
	Determines the file control block address	VARPTR	131
Branch	The program branches to the specified line	GOTO	69
	Deternines the condition	IF	69
	The program branches to several lines depending on the conditions	ON GOTO	90
Subroutine	Executes a subroutine	GOSUB	68
	Executes a particular subroutine depending on the conditions	ON GOSUB	90
	The program returns to the main routine	RETURN	112
Repeat	Repeats execution for the specified number of times	FOR	66
		NEXT	88
Error	Generates an error intentionally	ERROR	64
	Defines the start line for an error recovery routine	ON ERROR GOTO	89
	The program returns from the error recovery routine to the main routine	RESUME	112
	Determines the line number where the error was generated.	ERL	64
	Determines the error number	ERR	64
Stop	Stops the program	STOP	124
End	Terminates the program	END	63
Remarks	Inserts remarks in the program	REM	110
Character String manipulation	Replaces part of a character string	MID$	87
	Determines part of a character string	LEFT$	78
		MID$	86
		RIGHT$	113
	Determines the specified length of the space character string	SPACE$	119
	Determines the specified character string	STRING$	128
	Determines the position of a character string within a character string	INSTR	74
	Determines the length of a character string	LEN	78

Function	Description	Command	
Type Conversion	Converts a numerical value into a double precision real number.	CDBL	48
	Converts a numerical value into an integer	CINT	48
	Converts a numerical value into a single precision real number.	CSNG	55
	Determines the character code	ASC	44
	Converts a character string into its numerical value	VAL	131
	Converts a character code into the corresponding character	CHR$	48
	Converts a decimal into a binary string	BIN$	46
	Converts a decimal into an octal string	OCT$	89
	Converts a decimal into a hexadecimal string	HEX$	69
	Converts a numerical value into a character string	STR$	128
Numerical Operation	Determines the arc tangent	ATN	44
	Determines the cosine	COS	54
	Determines the sine	SIN	117
	Determines the tangent	TAN	129
	Determines the exponential	EXP	65
	Determines the log	LOG	83
	Determines the square root	SQR	123
	Determines the absolute value	ABS	44
	Determines the integer part	FIX	65
	Determines the maximum integer below the specified number	INT	74
	Determines the sign	SGN	117
Variables	Assigns a value to a variable	LET	78
	Stores constants to be read by a READ statement	DATA	56
	Reads the constants set in the DATA statement	READ	110
	Specifies the DATA statement to be read by the READ statement	RESTORE	111
	Defines dimensions an array	DIM	61
	Deletes an array	ERASE	62
	Initializes all variables	CLEAR	50
	Exchanges values between two variables	SWAP	128
	Determines the memory address where a variable is stored.	VARPTR	131
	Defines an integer variable	DEFINT	58
	Defines a single precision variable	DEFSNG	59
	Defines a double precision variable	DEFDBL	58
	Defines a string variable	DEFSTR	59
Random Number	Determines the value of a random number	RND	113
User Functions	Defines a user function	DEF FN	57

Function	Description	Command	
Memory	Determines the amount of unused memory	FRE	67
	Determines the contents of the specified address	PEEK	99
	Write data into the specified address	POKE	101
	Defines the size of the memory area	CLEAR	50
Machine Language	Defines the start address of a machine language subroutine.	DEFUSR	57
	Executes a machine language subroutine	USR	130
Interval Timer (Internal Clock)	Sets the value of the internal clock	TIME	129
	Determines the value of the internal clock	TIME	129
	Controls the interval timer interrupt	INTERVAL ON	75
		INTERVAL OFF	75
		INTERVAL STOP	75
		ON INTERVAL GOSUB	91
Extended Statement	Calls the extended statement	CALL	8

MEMO

MEMO

TOSHIBA CORPORATION
TOKYO JAPAN
PRINTED IN JAPAN
E0075