PROGRAMMING MANUAL

SANYO

SANYO

MSX-BASIC PROGRAMMING MANUAL

MSX PERSONAL COMPUTER

PREFACE

This manual has been made up to discuss the programming language called MSX BASIC for the Sanyo Personal Computer. If falls into the following six chapters. Please read carefully, together with the Operating Instructions, for your applications. The Operating Instructions, an independent volume, gives in details how to use the personal computer and peripheral units.

MSX-BASIC

PROGRAMMING MANUAL

TABLE OF CONTENTS

CHAPTER 1

1

HOW TO PROGRAM

The utilization of each MSX-BASIC command and function is explained in this manual. Also, additional explanations and explanations that cover several groups of commands are provided in chapter 4 with actual examples.
Please use this manual to learn MSX-BASIC or for actual MSX-BASIC programming.

1 WHAT IS BASIC ?

Computers run on a machine language which is a combination of 0 and 1. However the use of the machine language is difficult. A more simplified language, called BASIC, or an acronym for **Beginners All-purpose Symbolic Instruction Code** is used with this computer.

2 EXECUTION OF COMMAND

All commands are executed by typing **RETURN** key (↵) after commands are typed in with keys on the keyboard.

Taking **PRINT** command as an example, this subsection will explain the use of the BASIC language to display alphanumeric characters on the screen. To begin with, clear the screen by typing SHIFT + CLS HOME.

2-1 DISPLAY OF ALPHANUMERIC CHARACTERS ON THE SCREEN:

PRINT command is used to display alphanumeric characters on the screen after the word **PRINT**. The alphanumeric characters desired to be displayed must be enclosed in quotation marks (") at the beginning and the end of any string of characters.

```
PRINT "MSX BASIC" ———— to be followed by ↵
MSX BASIC ———————————— appears as the result of the command
Ok ——————————————————— the completion of command execution
□ ———————————————————— the cursor appears
```

2-2 DISPLAY OF NUMERIC CHARACTERS AND CALCULATIONS:

PRINT command can be substituted by ?. Thus, type:

?58*9-21 and ⏎ , and 501, which is the result

In BASIC, the asterisk (*) and the slash (/) are used to denote multiplication and division, respectively. In MSX-BASIC, the following Arithmetic symbols are used.

Arithmetic expression evaluations

Arithmetic operator	Semantics	Example	Priority order
+	Addition (X+Y)	X+Y	6
−	Subtraction (X−Y)	X−Y	
*	Multiplication (X x Y)	X*Y	3
/	Division (X÷Y)	X/Y	
∧	Power (X^2)	X^2	1
−	Changes a sign (−X)	−X	2
\	Integer division	6.7\2.3	4
MOD	Remainder of integer division	X MOD 10	5

Here are some particular symbols for your programming.

. (period) — Used to input a line number for the current BASIC program. A new line can be inserted or an error be corrected with the screen editor in the current program. Practicable for LIST, RENUM and other statements instead of a line number.
Example: LIST.

- (minus) — Used to specify a numeral value range. In LIST statement, for example, a command can specify its related range of lines such as n- thru m-line.
Example: LIST 100-200

: (colon) — Used for delimiting a multi-statement.
Example: A=B+C: PRINT A

, (comma) — Used for delimiting two or more parameters or numeral values in PRINT, INPUT, DATA and other statements.
Example: INPUT A, B, C
DATA 8, 64, 256

;	(semicolon)	Used for delimiting between numerical values or character strings in PRINT statement, for example. Example: PRINT A$; B$
'	(apostrophe)	Used in place of REM statement. Example: 'MUSIC
?	(question mark)	Used in place of PRINT statement. Example: ?5*3.14
"	(quotation mark)	Used to indicate a character string in specifying the string constant by putting a mark before and after. A character string contains up to 255 characters. Example: PRINT "MSX"
	(space)	Any blank spaces may be put in a statement to make the program more readable. No space can be, however, given in reserved words such as commands, statements, functions and system functions. Note also that spaces in the string constant have a meaning as characters. Spaces in statements are ignored when executing a command, but stored together when the line is transferred into the memory program area. So the spaces too are retrieved from the memory. A string which contains 0 characters is called a "null" string. Before a string variable is set to a value in the program, it is initialized to the null string. PRINTing a null string on the terminal will cause no characters to be printed, and the cursor will not be advanced to the next column. Setting a string variable to the null string can be used to free up the string space used by a non-null string variable.

2-3 RE-EXECUTION OF COMMANDS:

If required, commands can be reexecuted by moving the cursor to the beginning of the command and typing ↵ as follows:

```
?100-2^5 ↵
 68
Ok
```

Bring the cursor over ? and type ↵ , then the command is reentered and re-executed.

A partial or total change of input is also possible. Move the cursor to the location where any change is desired. If for instance 5 is desired to be replaced with 6, just type 6 over 5.

3 DIRECT MODE V.S. INDIRECT MODE

3-1 DIRECT MODE:

1. Indirect mode command can be typed in for an immediate execution as follows:

```
?9*25 .............. Type in command and ↵ .
 225 ............... The result of executing command.
Ok
□ .................. The cursor
```

None of these commands in direct mode is stored in memory and with the clearance of the screen by typing **SHIFT** + **CLS HOME**, all commands typed in are completely erased.

2. Programming and its modifications:

Line numbers from 0 to 65529 can be programmed before executing commands using BASIC.

Typing 10 PRINT "MSX" ↵ will only store this command in RAM of computer but will not execute it.

Then type:

```
20 PRINT "BASIC" ↵
30 ? 10*8-5 ↵
```

Then clear the screen.
Next type **LIST** ↵ and the screen will display as follows:

```
list
10 PRINT "MSX"
20 PRINT "BASIC"
30 PRINT 10*8-5
Ok
```

Since the program is stored in RAM memory of the computer, the program once cleared can be redisplayed time and time again, by listing, and modification of any part of the program can be made at will.

3-2 INDIRECT MODE:

By typing RUN, the operation mode is changed from direct to indirect. If either of the following applies, the operation mode is reverted back to direct from indirect:

1. If the execution of the program is interrupted by typing **CTRL**+**STOP**, or

2. If any error is contained in the program, in grammar, in calculation formula (a division by 0, etc.), and so on, or

3. If the program is terminated by **END** command, or if the program has exhausted the line numbers.

During the command execution, this computer will not accept any typing input, unless especially so specified. Type **RUN** and ↵ to display the following on the screen:

RUN ··················	direct mode
MSX ··················	execution of program
BASIC ··············	ditto
75 ····················	ditto
Ok ····················	back to direct mode
□ ······················	the cursor

Note:

The computer operation is controlled by an LSI called CPU (Central Processing Unit). The alphanumeric characters entered are memorized in the keyboard buffer. The CPU controls such input memories for display on the screen, for programming, and for execution of programs responding to the operator's command such as typing [↵] which will execute the program loaded in the computer. If the command is preceded by **line numbers**, it is stored in memory for the indirect mode operation.

The **RUN** command which is entered in direct mode will execute the program and upon completion of its execution, or if any error occurs, the operation mode is reverted back to the direct mode.

CPU takes care of both the program stored in memory and the cursor position. Thus by moving the cursor back and typing [↵], the command once executed can be executed again.

4 MODIFYING THE PROGRAM

It is almost impossible to make the perfect program at the first trial. Usually the so called debugging process is required to correct errors in typing, in calculation formula, etc.

4-1 ADDING A LINE:

The program is executed in sequence of the line numbers.

```
list
10 PRINT "MSX"
20 PRINT "BASIC"
30 PRINT 10*8-5
Ok
```

If additional command is required between line numbers, for instance to insert between the line numbers 20 and 30:

```
PRINT "10*8-5=";
```

Just type **25 ?"10*8–5=";** [↵] and clear the screen ([SHIFT] + [CLS HOME]) and then type **list** again. The line 25 is now inserted between the lines 20 and 30.

4-2 REPLACING A LINE:

If MSX of the line 10 in the program shown in 4-1 above is desired to be replaced with ABC, just type **10 PRINT "ABC"** ↵ . In BASIC, if more than one inputs bearing the same line number are entered by typing, the latest input prevails over all prior inputs:

```
list
10 PRINT "ABC"
20 PRINT "BASIC"
25 PRINT "10*8-5=";
30 PRINT 10*8-5
Ok
```

4-3 DELETING A LINE:

1. Any line can be deleted by just typing only the particular line number not required and ↵ .

2. Any particular range of lines can be deleted by the **DELETE** command as follows:

```
DELETE -20 ↵      ····· Will delete up to line 20.
DELETE 20-25 ↵    ··· Will delete from line 20 to line 25.
```

Programming titbits — TO ENTER SIMILAR COMMANDS QUICKLY:

1. Enter the command: KEY 10, "PRINT"+CHR$(&H22) ↵ and type f10 key and **PRINT"** command is entered at each touch of f10 key.
2. Entering command by changing line numbers is also possible.
   After entering:
   10 PRINT "M",
   move the cursor to the letter "M" and replace it with the letter "S" and then the cursor to 10 and replace it with 20.

```
10 PRINT "M"
20 PRINT "S"
30 PRINT "X"
```

3. Exception to line deletion:
   When auto command is used to generate line numbers automatically, and if the line number already entered is entered again, the asterisk (*) appears after that line number to show that the number previously entered is still valid.

5 RESERVED WORDS OF BASIC

The following are the fundamental reserved words necessary to start practicing the programming based on BASIC.

5-1 RESERVED WORDS

1. Generally speaking, entries into computer under direct mode are called commands and under indirect mode, statements. However, a clear cut distinction between commands and statements is not possible because some of the entries can be made in both **direct** and **indirect modes**.

2. Reserved words can be entered in upper or lower case characters or any combination of upper or lower cases. For instance the PRINT command can be entered as Print, PRint, or pRINT but P RINT (a space between characters) will result in an error entry.

Reserved Words of BASIC

COMMANDS	COMMANDS/STATEMENTS	STATEMENTS	FUNCTIONS
run	print	IF-THEN	SQR( )
list	color	FOR-NEXT	INT( )
			ABS( )

5-2 FUNCTIONS:

1. If given data to work on, functions will calculate or execute operations. Functions are never used alone but in combination with other commands.

   INT (3.14) ↵ — This command is used to obtain integer of the number enclosed in the parentheses.

   Typing the foregoing alone will result in the **Syntax error** message on the screen. However, type **PRINT INT (3.14)** ↵ and the answer is 3 is given.

2. Some of the functions are already defined by BASIC, such as ABS, INT, SIN, COS, PEEK, etc. while others can be defined also by the user.

3. The user defined functions are called **DEF FN**. The name of the DEF FN can be further defined by a suffix which follows the FN, (for example, FNA, FNB, FNC, etc.).

User defined function: (Example)

Calculation formula (Example)
Variables used
Function name

```
DEF FNA(A,B)=A*B-B/10
```

6 CONSTANTS AND VARIABLES:

There are two types of calculations, 1. a calculation using constants and 2. a calculation using variables, as follows:

1. PRINT 100^5
2. A=100:B=2:C=5:PRINT A-B^C

While the result of calculation is the same with 1. and 2., in the latter, values for calculation can be changed.

```
10 INPUT A:B
20 PRINT A;"*";B;
30 PRINT "=";A*B
40 GOTO 10
```

Type run and ↵ will display **?** on the screen. Type 4,5 ↵ and 4*5=20 will then be displayed and **?** will reappear showing that computer is ready for next assignment of command.

6-1 STRING CONSTANTS AND VARIABLES

In the following example, both constant and variable string characters are included.

```
10 PRINT "MSX" ····  10 MSX is a fixed string constant.
20 A$="BASIC" ·····  20 A$ is an string variable the constants
30 PRINT A$               which can be changed.
```

String variables are expressed in two digits of alphanumeric characters, starting with an alphabet followed by the dollar mark ($), for example:

```
A$,B$,C$....
A1$,B2$,CV3$....
A1$(1),BC$(1)...
```

MSX-BASIC reserved words (command names, function names, etc.) or a character string that includes a reserved word cannot be used as a variable name.
Only the first two characters are significant (the 1st character must be an alphabetical character).

6-2 NUMERIC CONSTANTS AND VARIABLES:

Numeric constants and variables are simply called constants and variables in general. In the following program, A is a variable while 5 is a constant.

```
10 INPUT A
20 PRINT A*5
```

Constants and variables take the form of Integer, Single Precision and Double Precision.

1. Integers:

Any value in the range −32768 to 32767 which is suffixed with the percentage symbol (%) is called an integer as follows:

3.14% 3,   3168% 3168

Thus A% will mean an integer variable.
Thus, statements, A=5.14   A%=5.14, will result in A%=5.

2. Single Precision:

Real quantities suffixed with ! mark are Single Precision numbers. Up to 6 digits are valid and the 7th digit and over are truncated at the 7th digit and such numbers are expressed in 6 digits.

100! . . . . 100.000 (displayed on the screen as 100)

12345678! 12345700

123.45678! 123.45700

A! (Single Precision Variables)

3. Double Precision:

Real quantities with 7 digits and over or real quantities suffixed with # mark are called Double Precision numbers. Up to 14 digits are valid with the 15th digit truncated.

3.14159265358979323# 3.1415926535898

314159265358979323# . . . . 3.1415926535898E+17

A# (Double Precision Variables)

Any number which is not suffixed with any of the marks, %, !, or #, is treated as Double Precision Variables.

Type declaration

Declaration by a type declaration character	Declaration by a DEF statement	Type declared
Add % Example: A%	DEFINT Example: DEFINT A	Integer type
Add ! Example: B!	DEFSNG Example: DEFSNG B	Single precision
Add # Example: C#	DEFDBL Example: DEFDBL C	Double precision
Add $ Example: D$	DEFSTR Example: DEFSTR D	String type

When a different type of type declaration character is placed for the variable name after the type declaration statement (DEFINT, etc.) was executed, the type declaration character has priority.

Type conversion of numeric constant:

When necessary MSX-BASIC will convert a numeric constant from one type to another. The following rules and examples should be kept in mind.

Binary expression	Octal expression	Decimal expression	Hexadecimal expression
&B0	&O1	1	&H1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10

The decimal number 13 is expressed in MSX-BASIC for each type as follows:

&B1101 &015 13 &HD

7 FUNDAMENTAL OPERATIONS IN PROGRAMMING:

The following descriptions are important for typing inputs and for understanding the language used in programming.

7-1 AUTO COMMAND TO GENERATE LINE NUMBERS AUTOMATICALLY.

1. Type AUTO [<the opening line number>] [, <incremental unit number>] : If no numbers are filled within the brackets [ ], AUTO command will begin with the line number 10, and will increase by the unit of 10 at each typing of [↵]

2. To disengage AUTO mode, type [CTRL] + [STOP] or [CTRL] + [C].

- [CTRL] + [C] can be used to interrupt input commands, while typing [CTRL] + [STOP] can stop both input and execution modes of operation.

7-2 INPUT DURING THE PROGRAM OPERATION

Any entry during execution mode will not be accepted by the computer, with the exception of special commands, such as [CTRL] + [STOP], or when the program includes special commands and statements as follows:

1. INPUT command:

INPUT command if included in the program will stop execution of the operation to wait for input from the keyboard.

```
10 INPUT B
20 IF B>24 THEN GOTO 10
30 C$=STRING$(B,"@")
40 LOCATE 4,CSRLIN-1:PRINT C$
50 GOTO 10
```

This program will show ? after it is executed by typing RUN to wait for keyboard input. Type numbers (integer) in the range 0–24, and @ mark will be displayed on the screen in the exact number typed. INPUT C$ will mean that the execution of command will wait for an input of string variables.

2. INKEY$ command:

This command in the program will permit accepting key input during the program operation.
10 replaces INKEY$ with string variables A$. 20 will assume that space key is entered if in fact no key is pressed.

```
10 A$=INKEY$
20 A$="" THEN 20
30 PRINT A$
40 GOTO 10
```

7-3 JUMPING OUT OF THE NORMAL PROGRAM SEQUENCE:

1. GOTO command:

This command is used to branch unconditionally out of the normal program sequence to a specified line number.

If line number is an executable statement, that statement and those following are executed. If it is a nonexecutable statement, execution proceeds at the first executable statement encountered after line number.

2. GOSUB command:

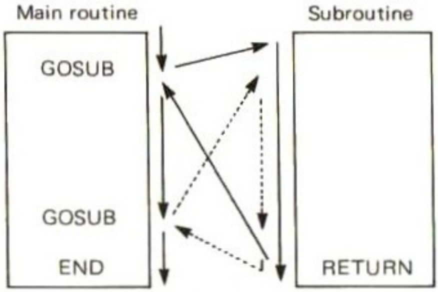
This command is a subroutine which may be called any number of times in a program, and a subroutine may be called from within another subroutine. The RETURN statement in a subroutine causes BASIC to branch back to the statement following the most recent GOSUB statement.

A subroutine may contain more than one RETURN statement, should logic dictate a return at different points in the subroutine.

Subroutines may appear anywhere in the program, but it is recommended that the subroutine be readily distinguishable from the main routine.

To prevent inadvertent entry into the subroutine, it may be preceded by a STOP, END, or GOTO statement that directs program control around the subroutine.

Otherwise, a "RETURN without GOSUB" error message is issued and execution is terminated.



7-4 CONDITIONAL COMMAND:

1. IF <expression> THEN <statement(s)> or <line number>
[ELSE <statement(s)> or <line number>]

2. IF <expression> GOTO <line number>
[ELSE <statement(s)> or <line number>]

- To make a decision regarding program flow based on the result returned by an expression.

- If the result of <expression> is true (except zero), the THEN or GOTO clause is executed. THEN may be followed by either a line number for branching or one or more statements to be executed. GOTO is always followed by a line number. If the result of <expression> is false (zero), the THEN or GOTO clause is ignored and the ELSE clause, if present, is executed. Execution continues with the next executable statement.

■**Example:**

A=1:B=2 . . . A=B is false (zero)
A=2:B=2 . . . A=B is true (except zero)

- IF . . . THEN . . . ELSE statements may be nested. Nesting is limited only by the length of the line. If the statement does not contain the same number of ELSE and THEN clauses, each ELSE is matched with the closest unmatching THEN.

- If an IF . . . THEN statement is followed by a line number in the direct mode, an "Undefined line" error results unless a statement with the specified line number had previously been entered in the indirect mode.

3. ON <expression> GOTO <line number> [, <line number> . . . .]
ON <expression> GOSUB <line number> [, <line number> . . . .]

- To branch to one of several specified line numbers, depending on the value returned when an expression is evaluated. The value of <expression> determines which line number in the list will be used for branching. For example, if the value is three, the third line number in the list will be the destination of the branch. (If the value is a noninteger, the fractional portion is disregarded.)

- In the ON . . . GOSUB statement, each line number in the list must be the first line number of a subroutine.

- If the value of <expression> is zero or greater than the number of items in the list (but less than or equal to 255), BASIC continues with the next executable statement. If the value of <expression> is negative or greater than 255, an "illegal function call" error occurs.

```
10 PRINT " INPUT ABSOLUTE(No.) "
20 INPUT B
30 A=ABS(B/10)
40 IF A>3 THEN 100
50 ON A GOTO 70,80,90
60 PRINT "LESS THAN 10":GOTO 20
70 PRINT "10 TO 19":GOTO 20
80 PRINT "20 TO 29":GOTO 20
90 PRINT "30 TO 39":GOTO 20
100 PRINT "MORE THAN 39"
110 GOTO 20
```

7-5 LOGICAL OR RELATIONAL OPERATOR:

- Conditional command, IF . . . THEN, is called logical or relational operator.
- Relational operators are used to compare two values. The result of the comparison is either "true" (−1) or "false" (0). This result may then be used to make a decision regarding program flow.

Logical expressions

Logical expressions perform logical operations between numeric type constants, variables, and functions.
Logical operation . . . . . Converts data to an integer considered as 16 bit binary, and performs an operation for each corresponding bit.

Logical operation	Logical operation result for each bit		
NOT (negation)	X		NOT X
	1		0
	0		1
AND (logical product)	X	Y	X AND Y
	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR (logical sum)	X	Y	X OR Y
	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR (exclusive OR)	X	Y	X XOR Y
	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV (exclusive OR negation)	X	Y	X EQV Y
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP (Implication)	X	Y	X IMP Y
	1	1	1
	1	0	0
	0	1	1
	0	0	1

Relational expressions

The value of two data are compared and the result is given as true (−1) or false.

Relational operator	Semantics	Example
=	Equal	X=Y, X$=Y$
<	Smaller	X<Y, X$<Y$
>	Larger	X>Y, X$<Y$
<>, ><	Not equal	X<>Y, X$><Y$
<=, =<	Smaller or equal	X<=Y, X$<=Y$
>=, =>	Larger or equal	X>=Y, X$>=Y$

7-6 CONDITIONAL LOOP COMMAND:

1. FOR . . . NEXT command:

For <variable> = x to y [STEP z]
NEXT [<variable>] [, <variable> . . .]

Note:

<Variable> can be integer, single-precision or double-precision, where x, y, z, are numeric expressions.

- To allow a series of instructions to be performed in a loop a given number of items:

- <Variable> is used as a counter. The first numeric expression (x) is the initial value of the counter. The second numeric expression (y) is the final value of the counter. The program lines following the FOR statement are executed until the NEXT statement is encountered. Then the counter is incremented by the amount specified by STEP. A check is performed to see if the value of the counter is now greater than the final value (y).
  If it is not greater, BASIC branches back to the statement after the FOR statement and the process is repeated. If it is greater, execution continues with the statement following the NEXT statement. This is a FOR . . . NEXT loop. If STEP is not specified, the increment is assumed to be one.

- If step is negative, the final value of the counter is set to be less than the initial value. The counter is decremented each time through the loop, and the loop is executed until the counter is less than the final value.

- The body of the loop is executed one time at least if the initial value of the loop times the sign of the step exceeds the final value times the sign of the step.

- FOR . . . NEXT loop may be nested, that is, a FOR . . . NEXT loop may be placed within the context of another FOR . . . NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop. If nested loops have the same end point, a single NEXT statement may be used for all of them. Such nesting of FOR . . . NEXT loops is limited only by available memory.

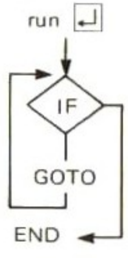
- The variable(s) in the NEXT statement may be omitted, in which case the NEXT statement will match the most recent FOR statement. If a NEXT statement is encountered before its corresponding FOR statement, a "NEXT without FOR" error message is issued and execution is terminated.

```
10 FOR N=&H40 TO &H5F
20 PRINT CHR$(1)+CHR$(N);" ";
30 NEXT N
40 FOR K=&H20 TO &HFF
50 PRINT CHR$(K);" ";
60 NEXT K
```

2. IF <expression> GOTO <statement(s)>; <line number> used as a loop command:

A combination of IF . . . GOTO command with variables can repeat the command within the specified line numbers. To use this command, the following two alternatives are available. Both of these operations however are almost identical.

Preceding IF



Succeeding IF

- Preceding IF will determine whether to execute the program by IF command, and if executed, GOTO statement will skip line numbers as programmed.

- Succeeding IF will skip line numbers by conditional GOTO statement based on judgement made by IF statement, after once running the program.

■ **Example**

```
10 N=1
20 IF N>15 GOTO 70
30 X=RND(1)*30
40 Y=RND(1)*21
50 LOCATE X,Y:PRINT "@"
60 N=N+1:GOTO 20
70 END
```

Note: Programming Tibits – Multiple statements

- In each line number in the BASIC program, (not the line displayed on the screen), program of upto 255 characters can be entered. The total line numbers can be reduced by packing as much information as possible per line number.

- Command or statement within the line number can be segregated by colon (:). The program is a sample of multiple statement:

```
10 COLOR 15,1,1:SCREEN 3:FOR R=1 TO 76:X
1=-1*R*COS(R):X2=125-X1:Y1=R*SIN(R):Y2=1
00-Y1:C=RND(-TIME)*13+2:PSET(X2,Y2),C:PS
ET(X1+125,Y1+100),C:PLAY "N=R;":NEXT R:F
OR X=0 TO 127:Y=191/255*X:LINE(X,Y)-(255
-X,191-Y),RND(-TIME)*13+2,B:NEXT X:COLOR
 15,4,7:END
```

7-7 RENUM COMMAND (RENUMBERING LINE NUMBERS):

●Since it is not infrequent that programs are modified and edited several times before they are completed, sometimes renumbering the line numbers becomes necessary as follows:

RENUM [[<new string line number>] [, [<old string line number>] [,<increment>]]

●<new string line number> is the first line number to be used in the new sequence. The default is 10. <old string line number> is the line in the current program where renumbering is to begin. The default is the first line of the program. <increment> is the increment to be used in the new sequence. The default is 10.

●RENUM also changes all line number references following GOTO, GOSUB, THEN, ELSE, ON .. GOTO, ON .. GOSUB and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the error message 'Undefined line nnnn in mmmm' is printed. The incorrect line number reference (nnnn) is not changed by RENUM, but line number mmmm may be changed.

Note:

RENUM cannot be used to change the order of program lines (for example, RENUM 15, 30 when the program has three lines numbered 10, 20 and 30) or to create line numbers greater than 65529. An "Illegal function call" error will result.

7-8 REM (REMARK) — INSERTION OF EXPLANATORY REMARKS:

- Insertions of explanatory remarks on the program are helpful for long programs or to read programs prepared by others. The explanatory remarks can be entered by typing REM, or apostrophe mark (') after line number. See line number 20 and 50 of the sample given below:

```
10 REM sample explanatory remarks
20 'to enter statement
30 INPUT "A+B...A";A
40 INPUT " .....B";B
50 'statement
60 PRINT A;"+";B;"=";A+B
70 END
```

- REM or (') cannot be used in a DATA statement as it would be considered legal data.

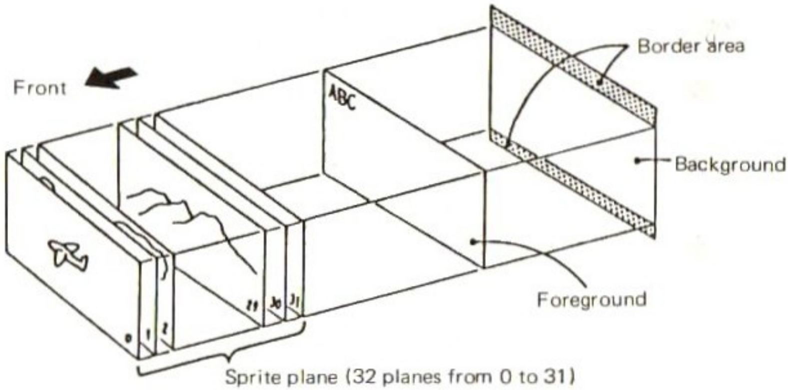
CHAPTER 2

FUNCTIONS OF MSX-BASIC

1 SCREEN CONFIGURATION

1-1 SCREEN CONFIGURATION

The display screen configuration for MSX-BASIC is as shown below.



1. Text mode and graphic mode

The text mode displays characters (alphanumeric characters), and the graphic mode displays graphics (dots, lines, circles, etc.). MSX-BASIC includes two text modes and two graphic modes that are selected by a SCREEN statement:

The modes selected by a SCREEN statement are as follows.

SCREEN statement	Mode		Sprite plane	Characteristics
SCREEN 0	Text	40 characters max. horizontal, 24 lines vertical.	Can't be used	Width per character is 6 dots. Since the width of a part of graphic characters is 8 dots, they cannot be completely displayed.
SCREEN 1		32 characters max. horizontal, 24 lines vertical.	Can be used	Width per character is 8 dots. Since most characters use only 6 dots, the display characters are read more easily compared to SCREEN 0.
SCREEN 2	Graphic	256 x 192 dots high resolution mode	Can be used	Graphics are drawn with 1 dot units.
SCREEN 3		256 x 192 dots multi color mode	Can be used	Graphics are drawn with block units of 4 x 4 dots.

The foreground, background, and border area are used in any mode. With characters or graphics displayed in the foreground, only color can be changed for the background and border area.

Also, the sprite planes can be used in addition to the above in modes other than the SCREEN 0 mode. A sprite plane is a plane on which a dynamic picture can be displayed by using freely defined sprite patterns which will be explained in the "How to use the sprite pattern" section.

2. Color specification

A COLOR statement specifys the colors of the foreground, background, and border area.

COLOR foreground color, background color, border area color

Both characters and graphics are displayed with the color specified for the foreground color, unless specifically specified.

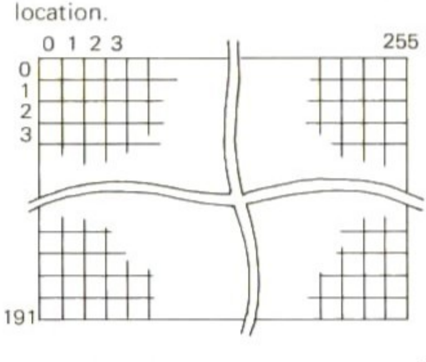
Also, in the SCREEN 0 mode, the color of the border area is always the same as that of the background.

1-2 HIGH RESOLUTION GRAPHICS... SCREEN 2 MODE

Graphics can be drawn with the following commands in a graphic mode.

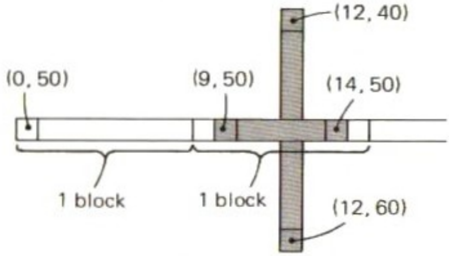
PSET, PRESET	Marks a dot or erases it.
LINE	Draws a straight line or square.
CIRCLE	Draws a circle.
PAINT	Colors
DRAW	Draws arbitrary graphics.

When these commands are used, screen coordinates are set to specify the screen location.



In the high resolution graphic mode, the location and color can be specified for each dot with 256 dots arranged vertically and 192 arranged horizontally as shown in the above figure.
However, if each specified color is restricted to 8 horizontal dots, only 1 color can be specified, and the color specified last is valid.

```
10 SCREEN 2
20 LINE (9,50)-(14,50),15
30 LINE (12,40)-(12,60),1
40 GOTO 40
```



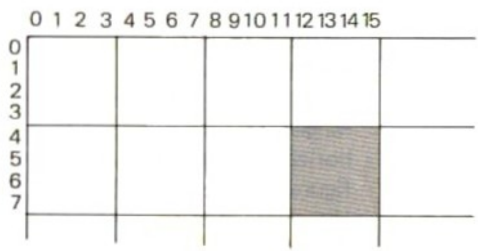
In the above program, with horizontal block coordinates from 8 to 15, although the color was specified as white, the straight line drawn by line 20 is displayed as black because the black line drawn next overlaps this line.
The specification of white becomes valid when the LINE statement in line 20 is changed as follows.

```
LINE (8,50)-(15,50)
```

This allows a maximum horizontal line to be drawn in the block of 8 dots.

1-3 MULTI COLOR GRAPHICS... SCREEN 3 MODE

Graphics can also be drawn in the SCREEN 3 mode by using a graphic command such as a PSET or LINE statement. Also, the location can be specified by utilizing 0–255 horizontal and 0–191 vertical coordinates. The unit for drawing graphics is a 4 x 4 dot block.




```
PSET(12,4),1
PSET(14,5),1
PSET(15,7),1
```

For example, since the above statements specify 1 dot in the same block, the part of the ■ part of the above figure is colored black by using any of them.

```
LINE(17,5)-(130,110)
```

This program draws a rough line to connect blocks that include (17.5) and (130, 110), or in other words to connect Fig. A and Fig. B.

Fig. A

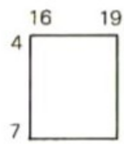
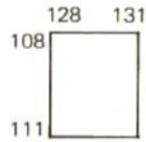


Fig. B

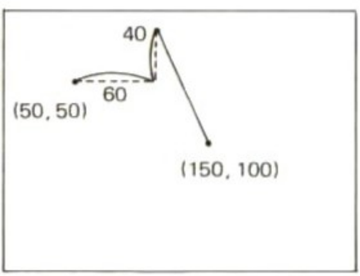


2 STEP SPECIFICATION

To specify coordinates (X, Y), the STEP (X, Y) specification can be performed by CIRCLE, LINE, PAINT, PSET, PRESET, and PUT SPRITE commands.
When these graphic commands are executed, the dot specified last is memorized by MSX-BASIC. After this, when STEP (X,Y) is specified next, the location of (X, Y) is determined on a new coordinate system with a dot specified last as the origin (0, 0). However, if STEP is omitted, the location can always be specified on the ordinary coordinate system using the extreme top left of the screen as the origin.

■ **Example 1**

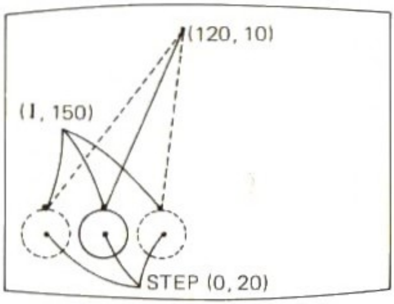
```
10 SCREEN 2
20 PSET (50,50)
30 LINE STEP(60,-40)-(150,100)
40 GOTO 40
```



In this program, the coordinates (50, 50) specified when the PSET statement was executed are memorized in line 20 then the program advances to line 30. Since STEP (60, −40) is used as a specification for the LINE statement starting point, the new starting point is a location that is 60 toward X and −40 toward Y with (50, 50) as a new origin.

■ **Example 2**

```
10 SCREEN 2
20 FOR I=30 TO 240 STEP 20
30 LINE (120,10)-(I,150)
40 CIRCLE STEP(0,20),20
50 CLS
60 NEXT I
```



In this program, although the LINE statement end point co-ordinates in line 30 are changed by the repetition of a FOR–NEXT loop, the center of the circle is specified by STEP (0, 20) in the CIRCLE statement of line 40, and the center of the circle is always determined to be a certain distance from the origin which is the end point of a straight line.

3 HOW TO USE THE SPRITE PATTERN

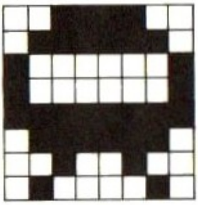
In MSX-BASIC, a pattern (called a sprite pattern) with a freely defined format is displayed as one of 32 sprite planes and can be moved.

3-1 SPRITE PATTERNS

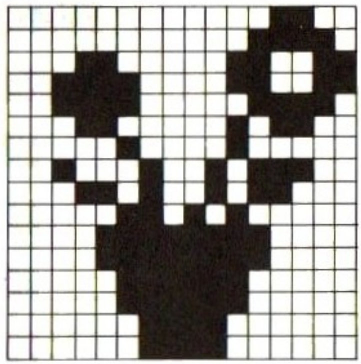
A sprite pattern consists of 8 x 8 or 16 x 16 dots for which two different sizes (magnified or unmagnified) can be selected. The magnified size is twice as big as the unmagnified size both horizontally and vertically.

8 x 8 dots unmagnified

16 x 16 dots unmagnified



8 x 8 dots magnified



16 x 16 dots magnified

The size of a sprite pattern is determined by a SCREEN statement. The 2nd parameter of a SCREEN statement selects the sprite size.

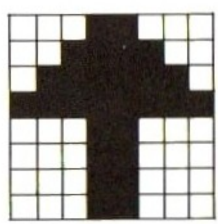
Parameter	Sprite size
0	8 x 8 dots unmagnified
1	8 x 8 dots magnified
2	16 x 16 dots unmagnified
3	16 x 16 dots magnified

```
SCREEN 2,2
```

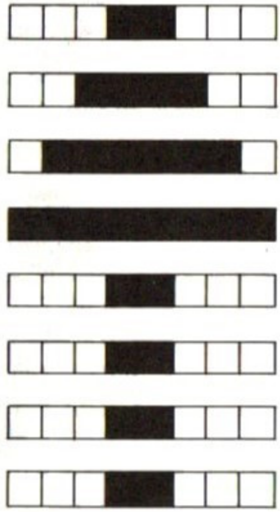
This statement specifies that a 16 x 16 unmagnified sprite is used in the high resolution graphic mode. The sprite size displayed on all sprite planes remains constant once the sprite size is specified by a SCREEN statement.

3-2 SPRITE PATTERN DEFINITION

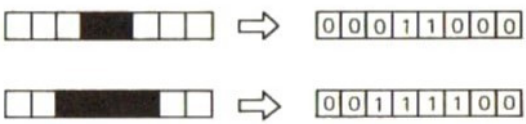
When an 8 x 8 dot pattern is defined, the pattern is first separated by 8 lines horizontally. For example, an arrow pattern is defined as shown in the following figure.



When this pattern is separated into 8 horizontal lines, it is divided into small patterns that consist of 8 dots.



Next the pattern in each line is arranged with 1 used to mark a dot and 0 used to indicate an unmarked dot which results in a binary number. For example, the top line is 00011000, and the next line is 00111100.



The binary numerals realized as mentioned above are converted to hexadecimal (or decimal).

For the top line,
00011000 (binary) = 18 (hexadecimal) or 24 (decimal).

For the second line,
00111100 (binary) = 3C (hexadecimal) or 60 (decimal).

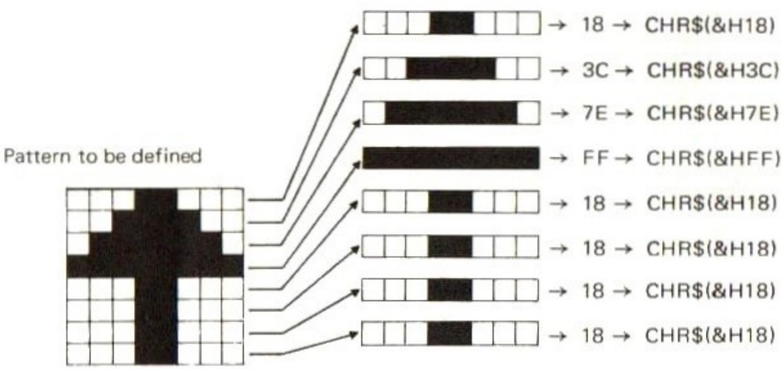
It is easier for the user who is unaccustomed to convert binary to hexadecimal to divide the 8 dot pattern into 4 dots on the left and 4 dots on the right to convert to one hexadecimal digit (0–F) by referring to the following table.

Pattern	Hexadecimal	Pattern	Hexadecimal
□□□□	0	■□□□	8
□□□■	1	■□□■	9
□□■□	2	■□■□	A
□□■■	3	■□■■	B
□■□□	4	■■□□	C
□■□■	5	■■□■	D
□■■□	6	■■■□	E
□■■■	7	■■■■	F

With the □□□■■□□□ pattern, left 4 dots is □□□■ and right 4 dots ■□□□

Therefore, they are converted to hexadecimal 18 based on the above table.

The character, for which hexadecimal (or decimal) is the character code, is obtained by using the CHR$ function. The definition of the sprite pattern explained above is arranged as follows.

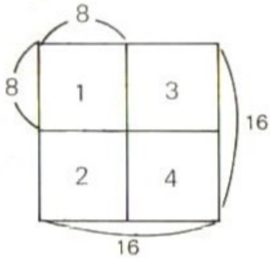


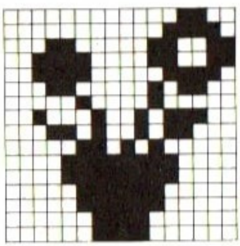
In regard to the 8 x 8 dot sprite pattern, the character data obtained as shown above is added sequentially from the top and is assigned to the SPRITE$ variable as a character string which defines the sprite pattern. For the arrow pattern in the above example, it is defined as follows.

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&H
7E)+CHR$(&HFF)+CHR$(&H18)+CHR$(&H18)+CHR
$(&H18)+CHR$(&H18)
```

The number of the defined sprite pattern is 1 and is indicated by the numeral 1 inside the parentheses of SPRITE$ (1).

A 16 x 16 dot sprite pattern can be defined with the same procedure. However, a 16 x 16 dot sprite pattern is considered to be a collection of four 8 x 8 dot sprite patterns, and these four patterns are defined after putting them together in the sequence shown below.





```
A$=CHR$(&H0)+CHR$(&H0)+CHR$(&H18)+CHR$(&H3C)
+CHR$(&H3C)+CHR$(&H18)+CHR$(&H04)+CHR$(&H22)
B$=CHR$(&H1A)+CHR$(&H06)+CHR$(&H0F)+
CHR$(&H0F)+CHR$(&H07)+CHR$(&H07)+CHR$(&H03)
+CHR$(&H03)
C$=CHR$(&H0C)+CHR$(&H1E)+CHR$(&H33)+
CHR$(&H33)+CHR$(&H1E)+CHR$(&H2C)+CHR$(&H20)
+CHR$(&H5C)
D$=CHR$(&H58)+CHR$(&HA0)+CHR$(&HF0)+
CHR$(&HF0)+CHR$(&HE0)+CHR$(&HE0)+CHR$(&HC0)
+CHR$(&HC0)
SPRITE$(2)=A$+B$+C$+D$
```

3-3 NUMBER OF SPRITE PATTERNS THAT CAN BE DEFINED

The numbers of 8 x 8 dot sprite patterns are from 0 to 255, and those of 16x16 dot sprite patterns are from 0 to 63. In other words, up to 256 8x8 dot sprite patterns can be defined, and up to 64 16x16 dot sprite patterns can be defined. (However, this is sometimes restricted depending on the memory capacity.)

3-4 SPRITE PATTERN DISPLAY

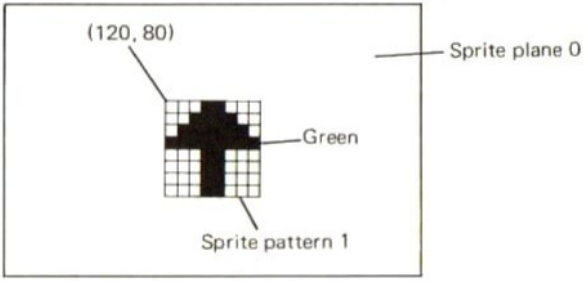
A PUT SPRITE statement is used to display a defined sprite pattern on a sprite plane.

PUT SPRITE sprite plane number, (X-coordinate, Y-coordinate), color code, sprite pattern number

To display a sprite pattern defined by the above at location (120, 80) of sprite plane 0 with green (color code 2), the program is as follows.

```
PUT SPRITE 0,(120,80),2,1
```

The specified display location is a dot on the left top of the sprite pattern frame. The X, Y-coordinates are specified using a coordinate system with (0, −1) on the graphic screen as the origin (0, 0).



PUT SPRITE 0, (120, 80), 2, 1

SPRITE PATTERN DISPLAY RULES

- Only one sprite pattern can be displayed on one sprite plane.
- When sprite patterns overlap on different sprite planes, the sprite pattern on the sprite plane at the back (larger number) is hidden by the sprite pattern in front.
- When five or more sprite patterns are arranged horizontally , up to four sprite patterns with a higher priority (on sprite planes with smaller numbers) are displayed.
- When the display location specification is omitted, it is considered that the location has been specified by a previous graphic instruction.
- When the color code is omitted, it is considered that the foreground color has been specified.
- When a sprite pattern number is omitted, it is considered that the same number as the sprite plane number has been specified.

3-5 TO MOVE A SPRITE PATTERN

To move a sprite pattern, replace the X and Y-coordinates of the display location specified by a PUT SPRITE statement with a variable, then execute the PUT SPRITE statement repeatedly by changing the value of the variable. Since the previous sprite pattern on a sprite plane disappears when a PUT SPRITE statement has been executed once, it is unnecessary to erase it in a program.

Also, since a pattern can be moved in 1 dot units, the movement is smooth.

In the following program, a UFO-shaped sprite pattern files about on the screen by changing its direction.

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 X=100:Y=100
40 S=INT(RND(1)*80)——— Determines the movement distance.
50 D=INT(RND(1)*4)
60 IF D=0 THEN VX=0:VY=-1
70 IF D=1 THEN VX=1:VY=0     Determines the direction.
80 IF D=2 THEN VX=0:VY=1
90 IF D=3 THEN VX=-1:VY=0
100 FOR I=0 TO S
110 PUT SPRITE 0,(X,Y),1,0
120 X=X+VX:Y=Y+VY
130 IF X>240 OR X<0 THEN VX=-VX     Moves the sprite.
140 IF Y>175 OR Y<0 THEN VY=-VY
150 NEXT I
160 GOTO 40
```

4 MUSIC PERFORMANCE

MSX-BASIC is provided with two music performance commands which are PLAY and SOUND. PLAY is a command that performs as specified by a subcommand using the LSI that controls pitch, rhythm, and timbre. Sound is output by writing several different data items into the LSI register. Specified data can be written directly to the LSI register by a SOUND statement. Therefore, a program that directly controls the sound with a SOUND statement can be prepared by knowing the function of the LSI sound register and the data to be written in.

4-1 CONTROL OF VOLUME VARIATIONS WITH A PLAY STATEMENT

Although the utilization of the PLAY statement is covered in the Chapter 3 PLAY section, the S subcommand and M subcommand can be explained as follows.

```
PLAY "CDEFG" ----------------①
PLAY "S13M255CDEFG"---------②(Each _ is a subcommand.)
```

Execute ① first and ② next in BASIC, and compare these two statements which have the same timbre.

When you execute

```
PLAY "S8M900CDEFG"
```

it sounds as if a piano is being continuously played at high speed.

Sn — Subcommand that selects the volume variation pattern.
Mn — Subcommand that determines the cycle of the pattern selected by Sn.

The initial values of Sn and Mn are S13 and M255 respectively. A different timbre can be generated by changing the value of n for Sn and Mn.

Pattern and cycle combinations

There are 8 patterns that can be selected by the S subcommand as shown in the table on page 117. The cycle becomes shorter as the value of n is minimized by the M subcommand. (In other words, the pattern repetition number in a certain period of time becomes larger.)
This can be proved by executing the following statement.

```
PLAY "S8M300CDEFG"
```

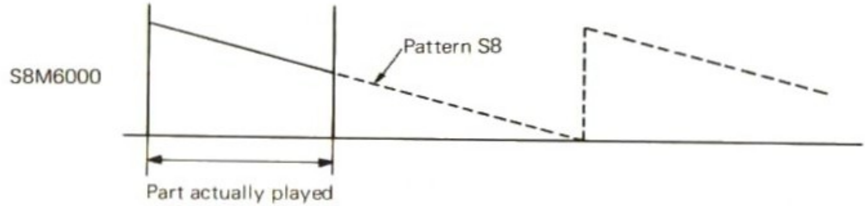
Let's listen to the following two statements and compare them.

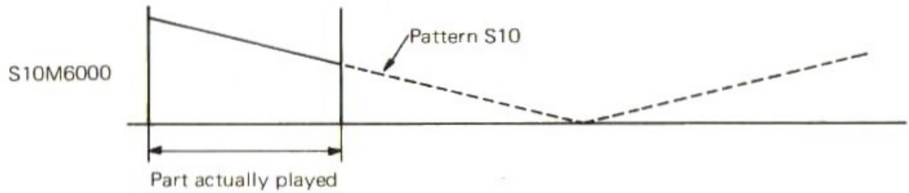
```
PLAY "S8M900CDEFG"
PLAY "S10M900CDEFG"
```

Now the difference in the patterns specified by the S subcommand is clear. However, if the value of n becomes too large in the M subcommand, the cycle becomes too long. Therefore, sometimes the difference is not clear.

```
PLAY "S8M6000CDEFG"
PLAY "S10M6000CDEFG"
```

When these two statements are executed, they both sound the same because the pattern was stretched horizontally (period) too long, and when the scale is played, the matching parts of different patterns are only used.





Since the length of the part actually played in the above figures is changed by the L subcommand specification, many enjoyable music performance programs can be prepared by skillfully selecting the right combinations.

4-2 SOUND AND NOISE WITH A SOUND STATEMENT

SOUND is a command that generates arbitrary sound or noise by writing data to a sound LSI register called a PSG (Programmable Sound Generator). The PSG is provided with 3 channels that generate sound (with a certain frequency). Noise can also be applied to all these channels. So the generation of triple chords and noise is possible. The PSG is provided with 16 registers which have different functions.

Register No.	Function
0, 1	Determines the frequency of channel A.
2, 3	Determines the frequency of channel B.
4, 5	Determines the frequency of channel C.
6	Determines the noise frequency.
7	Selects a channel.
8	Determines the volume of channel A.
9	Determines the volume of channel B.
10	Determines the volume of channel C.
11, 12	Determines the cycle of the volume variation pattern.
13	Selects the volume variation pattern.

(Registers 14 and 15 have no relationship with the musical performance.)

Sound frequency determination

The frequencies generated by the 3 different channels are determined by using 6 registers from 0 to 5. Data written in a register can be obtained with the following expression.

$$\frac{1789772.5\ (\mathrm{Hz})}{16 \times (\text{output frequency}\ (\mathrm{Hz}))} = 256 \times (\text{register } 1, 3, 5 \text{ data}) + (\text{register } 0, 2, 4 \text{ data})$$

For example, when 300 Hz sound is to be generated from channel A, the following expression is realized.

$$\frac{1789772.5}{16 \times 300} \doteqdot 373 = 256 \times 1 + 117$$

Therefore, write 117 to register 0, and 1 to register 1.
The actual statements are as follows.

```
SOUND 0,117
SOUND 1,1
```

In the case of channel B, since register 2 and 3 are used instead of register 0 and 1, the statements are as follows.

```
SOUND 2,117
SOUND 3,1
```

Noise frequency determination

Date from 0 to 31 can be written in register 6 which determines the noise (zoo sound) frequency. The following relational expression is realized between the data and frequency.

$$\text{Data value} = \frac{1789772.5\ (\mathrm{Hz})}{16 \times \text{noise frequency}\ (\mathrm{Hz})}$$

For example, when data 15 is written to register 6,

$$15 \doteqdot \frac{1789772.5}{16 \times 7457}$$

Therefore, the noise frequency is about 7457 Hz.

Channel specification

The channel used is determined by the data written in register 7.

Noise			Sound		
Channel C	Channel B	Channel A	Channel C	Channel B	Channel A
32	16	8	4	2	1

Add the numeric values that correspond to the channel used based on the above table and subtract the result from 255 to obtain the data to be written.

For example, when sound is only to be generated from channels A and B, and sound and noise from channel C, the following expression is realized in which 216 is the data to be written.

$$63 - (32 + 4 + 2 + 1) = 24$$

Sound generation after volume determination

Write data that determines the volume of channels A, B, and C to register 8, 9, and 10 respectively. Data from 0 to 15 can be written with 15 as the maximum volume.

The conditions required to generate sound are as mentioned above.
The following program generates three different sound pitches from channel A, B, and C.

`10 SOUND 0,47` `20 SOUND 1,1`	Channel A frequency as 200 Hz.
`30 SOUND 2,140` `40 SOUND 3,0`	Channel B frequency as 800 Hz.
`50 SOUND 4,56` `60 SOUND 5,0`	Channel C frequency as 2000 Hz.
`70 SOUND 7,56` `80 SOUND 8,9`	Specifies the sound output from channels A, B, and C.
`90 SOUND 9,10` `100 SOUND 10,11`	Determines the volume of each channel and generates the sound.

When the volume of each channel is changed in lines 80, 90, and 100 in this program, the sound output from each channel can be distinguished.
Also, when the program is executed once, the sound keeps generating.
Press the CTRL key and STOP key simultaneously to stop this.

Add:

```
65 SOUND 6,31    (Determines the noise frequency.)
```

to this program and modify line 70 as follows.

```
70 SOUND 7,48    (Outputs sound and noise from channel A
                  and sound from channel B and C.)
```

Now sound mixed with noise is generated.

Sound effect generation by volume variation patterns

Functions that are the same as the S subcommand and M subcommand of a PLAY statement can be performed with a SOUND statement. Volume variation patterns are determined by data written to register 13, which is the same as the n specification of a PLAY statement S subcommand (Sn).

See the table on page 117 for then values of corresponding patterns.

The cycle of a volume variation pattern is determined by data written to register 11 and 12 for which the following expression is realized.

$$\frac{1789772.5\ (\text{Hz})}{256 \times \text{cycle}\ (\text{Hz})} = 256 \times (\text{data in register 12}) + (\text{data in register 11}).$$

For example, when the cycle is set as 10 Hz, write 187 to register 11 and 2 to register 12 based on the following expression.

$$\frac{1789772.5}{256 \times 10} \doteqdot 699 = 256 \times 2 + 187$$

Set 16 as the volume of the channel in which the pattern specified above is to be used. For example, when the volume variation is to be applied to channel C, the statement is as follows.

```
SOUND 10,16
```

Many different sound effects can be generated by applying the volume variation pattern mentioned above to the noise, and by mixing the sound (tone) with a very high frequency and sound with a low frequency to generate a metallic sound or humming.

The following program generates the sound of a steam locomotive by periodically changing the noise volume.

```
10 FOR I=6 TO 13
20 READ J
30 SOUND I,J
40 NEXT I
50 DATA 31 -------------- Noise frequency
60 DATA 7---------------- Generates noise with channel A, B, and C.
70 DATA 16,16,16----- Changes the volume of channel A, B, and C.
80 DATA 71,2 ----------- Volume variation cycle 12 Hz.
90 DATA 14 --------------- Volume variation pattern 14.
```

5 FILE PROCESSING

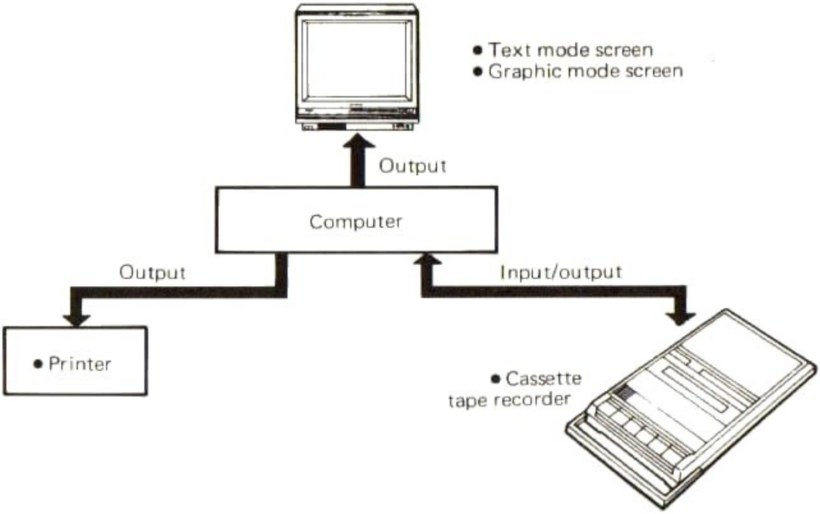
5-1 FILES AND FILE DEVICES

Sometimes program data provided in a program as a package is exchanged between a computer and equipment connected to a computer.

For example, lets consider that you keep a diary. There are several bookshelves in your room and a notebook entitled "diary" is on one of the book shelves. When you read your diary or write in it, first you got to the bookshelf of the subject and remove the notebook entitled "diary".

When this is applied to a computer, you are the computer and the contents of the diary is a program or data. The notebook where the program or data is recorded is called a file as far as computer terminology is concerned. The "diary" title on the notebook is the name given to a file and is a file name. The bookshelves are equivalent to connected equipment. If the wrong equipment is specified, the subject file cannot be found.

MSX-BASIC commands have been prepared to allow a file to be exchanged between a computer and four different kinds of connected equipment. The four different kinds of equipment are called basic file divices. The relationship between a basic file device and a computer is as shown in the following figure. There are two different file devices with one that only provides output to a file and another that provides both input and output based on the computer.



File Input/Output with a file can only be performed with a cassette tape recorder among the basic file devices of MSX-BASIC as shown in the above figure. Also, the screen of the monitor TV includes a text mode screen and a graphic mode screen.

Device names

When file exchantes are made with each file device in MSX-BASIC, a command is provided that specifies the file device used. At that time, the device name determined by MSX-BASIC is used.

File device	Device name
Cassette tape recorder	CAS:
Text mode screen	CRT:
Graphic mode screen	GRP:
Printer	LPT:

File name

A file must have a name with a character string that has up to 6 characters starting with an alphabetical character. If 7 or more characters are specified, the 7th character and after are ignored.

Although a file name can be omitted, it is recommended that a file name be used to distinguish one file from another when cassette tape Input/Output is performed.

5-2 PROGRAM FILES

The following commands save a BASIC program to file, load it from a file, or combine them.

CSAVE, CLOAD .... Cassette tape recorder dedicated.

SAVE, LOAD, BSAVE, BLOAD, MERGE .. Device can be specified.

When a program in memory is saved on cassette tape, execute:

$\text{CSAVE } \underbrace{\text{"PROG1"}}_{\text{File name}}$

or $\text{SAVE " }\underbrace{\text{CAS:}}_{\text{Device name}}\ \underbrace{\text{PROG1"}}_{\text{File name}}$

However, a program is saved with an intermediate language format when CSAVE is used, and with an ASCII format when SAVE is used.

A program saved by CSAVE can be loaded by using a CLOAD statement by specifying the same file name. Also, a program saved by a SAVE statement is loaded by a LOAD statement. Besides this, a program can be combined with another program that exists in memory by using a MERGE statement. However, this cannot be performed for a program saved by a CSAVE statement.

Since LOAD and MERGE statements are used to input a program from a file, only CAS: can be specified for a basic device. Also, if a SAVE statement is executed for the CRT: the result is the same as LIST execution. If a SAVE statement is executed for LPT:, the result is the same as LLIST execution.

5-3 DATA FILES

When data to be processed in a BASIC program is exchanged with a device, the concept of a file is utilized.

The following commands are used for data file Input/Output.

OPEN	Opens a file.
PRINT# PRINT# USING	Outputs data to a file.
INPUT# LINE INPUT#.	Inputs data from a file.
CLOSE	Closes a file.

5-4 CASSETTE TAPE FILE OPERATION

Output to a file (Write-in)

File data output procedures are roughly as follows.

1. Open a file with an OPEN statement.
2. Write data to the file with a PRINT# statement.
3. Close the file with a CLOSE statement.

The format of an OPEN statement is as follows when data is output.

OPEN "device name [file name]" FOR OUTPUT AS [#] file number

When this is executed, the set up of data output to a specified device with a specified file name is completed for a file. When file Input/Output is performed, the computer inputs or outputs data after storing it. The area prepared in memory for storing data is called a buffer. Up to 16 buffers can be prepared in MSX-BASIC. The file number specified by an OPEN statement is a buffer that is used from among 16 buffers, in which only 1 is specified initially.

After a file is opened by an OPEN statement, data is actually output by a PRINT# statement.

PRINT# file number, expression [separator expression] --------

The same file number as that specified by the OPEN statement is specified.

When data is output to a file with a PRINT# statement, a return code (&HOD) and a line feed code (&HOA) are automatically written next to data. When the data is read, these two codes indicate the punctuation of data.

When the data is string type, insert "," between each data if several data are output with one PRINT# statement.

For example, make a statement as follows:

PRINT# 1, A$; ","; B$

The comma also indicates the punctuation and the data A$ and B$ are handled as two separate data when they are input form the file.
When the data is numeric type, each data is automatically punctuated.

After data is output, the file is closed by a CLOSE statement.

CLOSE [#] file number

After this, since the relationship between the file number and the file is released, another file can be opened with the same file number.

■ **Program example**

```
10 DIM A$(1,3)
20 OPEN "CAS:DATA" FOR OUTPUT AS #1
30 FOR I=0 TO 1
40 FOR J=0 TO 3
50 READ A$(I,J)
60 PRINT #1,A$(I,J);",";
70 NEXT J
80 NEXT I
90 CLOSE #1
100 DATA JAPAN,ENGLAND,FRANCE,U.S.A
110 DATA TOKYO,LONDON,PARIS,NEW YORK
```

When this program is executed, the string type data "JAPAN", Comma (,), "ENGLAND" and so forth are sequentially written to cassette tape. The data is actually written as follows.

```
JAPAN,ENGLAND,FRANCE,U.S.A,TOKYO,LONDON,
PARIS,NEW YORK,
```

In line 60, a comma is inserted between data which indicates the punctuation of data so that the data can be distinguished from other data when data is input by an INPUT# statement.

File input (Read-out)

The procedure for data input from a file is as follows.

1. Open a file with an OPEN statement.
2. Read out data from the file with an INPUT# statement or LINE INPUT# statement (Assigns input data to a variable).
3. Close the file with a CLOSE statement.

The format of an OPEN statement when data is input from a file is as follows.

OPEN "device name [file name]" FOR INPUT AS [#] file number

The set up for data input from a file is prepared by this. Only file No. 1 can be specified initially.

After a file is opened, data is read-out by an INPUT# statement.
Data that is read-out when an INPUT# statement is used is as shown in the following table.

	For numeric type data	For string type data
Space, return code, line feed code before data.	Ignored	Ignored
Punctuation for data, or when data is punctuated.	Space, comma, return code, line feed code	Comma, return code, line feed code. For 255 character input.
When data is inside " "	——	Items inside " " are input as one data.

Also, a LINE INPUT# statement is only used for character data read-out in which input is performed with a return code as only punctuation for data.

After data input has been terminated, the file is closed by a CLOSE statement to separate the relationship between the file number and file.

■ **Program example**

```
10 DIM A$(1,3)
20 OPEN "CAS:DATA" FOR INPUT  AS #1
30 FOR I=0 TO 1
40 FOR J=0 TO 3
50 INPUT #1,A$(I,J)
60 NEXT J
70 NEXT I
80 CLOSE #1
90 FOR J=0 TO3
100 PRINT A$(0,J),A$(1,J)
110 NEXT J
```

This program is used to read-out a file on cassette tape, named "DATA", prepared in the previous program (lines 20 – 80) and to display the content on the screen (lines 90 – 110). In line 50, data is continuously assigned to the A$ (I, J) array variable.

```
10 OPEN "CAS:DATA" FOR INPUT AS #1
20 INPUT #1,A$
30 PRINT A$
40 GOTO 20
```

What happens if the file called "DATA" is input by using the program above? JAPAN, ENGLAND . . . are continuously assigned to the A$ character variable and are displayed on the screen. However, after the last data, NEW YORK, has been input, the program tries to input continuously data. When this occurs although the file has ended, an "Input past end" error occurs. To prevent this, the EOF function is used.

```
10 OPEN "CAS:DATA" FOR  INPUT AS #1
15 IF EOF(1)=-1 THEN GOTO 50
20 INPUT #1,A$
30 PRINT A$
40 GOTO 15
50 CLOSE #1
```

The EOF (file number) function gives – 1 when the last file data has been read out. In this program, if data remains or not is checked every time data is input when this function is used.

5-5 DISPLAYING CHARACTERS ON THE GRAPHIC SCREEN

When SCREEN 2 or SCREEN 3 is specified by a SCREEN statement, the screen enters the graphic mode which does not allow characters to be displayed by a PRINT statement.

To display characters on the graphic mode screen, a method is used in which the graphic mode screen is considered to be a file device and characters to be displayed are output as a file data.

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PRINT #1,"How do you do?"
40 GOTO 40
```

When this program is executed, the screen is converted to the graphic mode and "How do you do?" is displayed.

Execute one of the graphic instructions just before to specify the display location. After this, the location specified by the instruction last (256 horizontal, 192 vertical dots) is the top left corner of an 8 x 6 dot frame that holds the first character of the output character string.

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PRESET (100,50)
40 PRINT #1,"How do you do?"
50 GOTO 50
```

In this program, the location (100, 50) used by the PRESET instruction in line 30 is the top left corner of the character string output in line 40.

5-6 NUMBER OF FILES OPENED ONCE

Only one file can be specified when MSX-BASIC is initialized. In other words, only one file can be opened in one program at one time. When two or more files are to be opened at the same time, the number of lines are previously specified by:

```
MAXFILES=5
```

Based on this, 5 files with file numbers from 1 to 5 can be simultaneously opened. The maximum value that can be specified is 15.

Also, since file 0 is dedicated to CSAVE, CLOAD, CLOAD?, SAVE and LOAD, when:

```
MAXFILES=0
```

is executed, only CSAVE, CLOAD, CLOAD?, SAVE and LOAD commands can be used after this.

6 INTERRUPTS

An interrupt, used to suspend program flow that began during program execution, is caused by the occurrence of a specific external condition, and is used to perform other processing. The processing program executed when an interrupt occurs is called an interrupt processing program or an interrupt processing routine.

Another concept similar to an interrupt is a subroutine. However, a subroutine is only executed when a GOSUB statement is executed in MSX-BASIC. In other words, the execution of a subroutine is previously determined internally in a program.
On the other hand, an interrupt processing routine is executed by an external condition (for example, when the F1 key is pressed).

After execution of an interrupt processing routine has been terminated, the execution of the main program is normally resumed the same as for a subroutine.

6-1 MSX-BASIC INTERRUPTS

MSX-BASIC is provided with several commands to transfer control to an interrupt processing routine when an interrupt occurs. An interrupt can be used in the following cases. When an interrupt is used, its utilization is first declared by a command, and the starting line number of the interrupt processing routine is specified.

An interrupt can be used when:	Interrupt declaration command
A function key is pressed.	ON KEY GOSUB line number
A space bar, or joystick trigger button is pressed.	ON STRIG GOSUB line number
CTRL + STOP is pressed.	ON STOP GOSUB line number
Sprites overlap	ON SPRITE GOSUB line number
A certain period of time has passed.	ON INTERVAL = interval GOSUB line number

For example,

```
ON KEY GOSUB 1000
```

is a statement that declares when a function key is pressed, it is transferred to the routine from line 1000.

6-2 INTERRUPT UTILIZATION

An interrupt cannot actually be applied by only declaring an ON – GOSUB statement. A command that validates the interrupt used must be executed next. For example, to the interrupt that occurs when the [F1] key is pressed, execute:

```
KEY(1) ON
```

There are five commands that validate interrupts as follows.

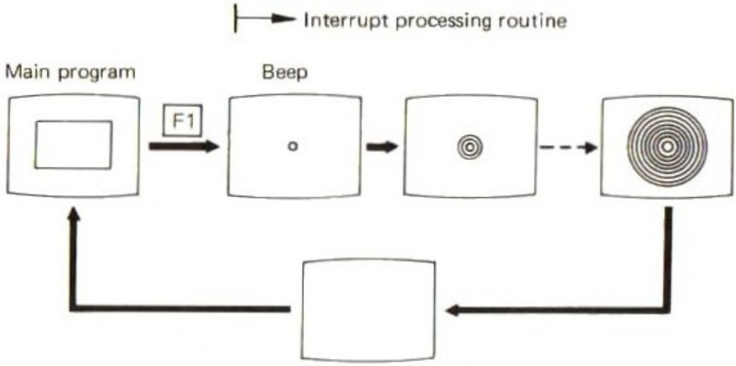
Command	Valid interrupt
KEY (function key number) ON	Interrupt by a function key.
STRIG (joystick number) ON	Interrupt by a space bar, joystick.
STOP ON	Interrupt by [CTRL] + [STOP] keys.
SPRITE ON	Interrupt by a sprite overlap.
INTERVAL ON	Interrupt with a certain spacing.

- **Program example**

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2                              } Main program
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'SUBROUTINE
110 BEEP:CLS
120 FOR I=10 TO 90 STEP 10
130 CIRCLE (120,100),I                   } Interrupt processing
140 NEXT I                                 routine
150 CLS
160 RETURN 40
```

In this program, when the [F1] key is pressed, it is set so that a transfer is made to a subroutine from line 110 in line 10 and 20.

When this program is executed, a rectangle is continuously displayed by line 40 and 50 of the main program. However, when the [F1] key is pressed, an interrupt occurs to provide a specified transfer to line 100. As a result, the rectangle disappears with a beep sound (BEEP: CLS), and 9 circles are continuously drawn. After the last circle has been drawn, the screen is cleared and a return is made to line 40 again.



6-3 INVALIDATING AN INTERRUPT

Lets add the following line to the program above.

```
105 KEY(1) OFF
```

Execute the program. When F1 is pressed the first time, an interrupt occurs. However, it does not occur after this even when the F1 key is pressed.

The reason for this is that when the interrupt processing routine was first executed,

```
KEY(1) OFF
```

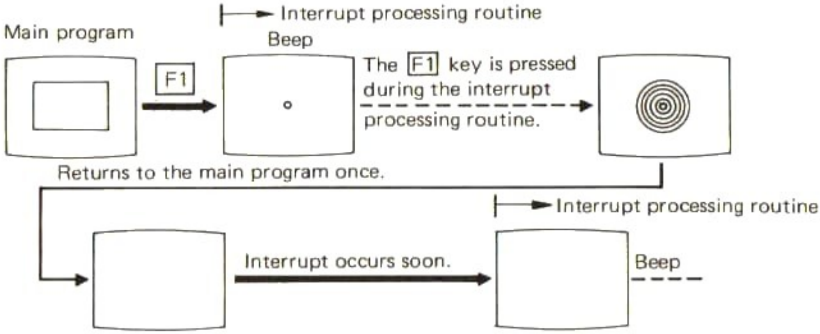
on line 105 was executed which invalidates the F1 key interrupt.

6-4 INTERRUPT HOLD

When execution is transferred to an interrupt processing routine by an interrupt, an interrupt hold state occurs. In this state, when an interrupt is applied again, an interrupt does not occur and a return is made to the main program by a RETURN statement for which an —ON statement automatically occurs, the main program is not executed and a transfer is made to an interrupt processing routine soon.

In other words, during the interrupt hold state, a return is not made to the start line of interrupt processing routine when an interrupt is applied but the interrupt application is memorized and an interrupt occurs after coming out of the processing routine once.

In regard to the program on page 52, when the [F1] key is pressed, 9 circles are drawn by an interrupt. However, an interrupt does not occur if the [F1] key is pressed before the last circle is drawn. Then, after the last circle has been drawn, a return is made to the main program. However, an interrupt occurs due to the second pression of the [F1] key and a rectangle is not drawn, but circles are drawn again.



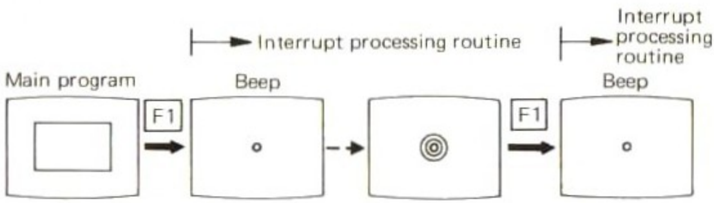
6-5 VALIDATING AN INTERRUPT DURING AN INTERRUPT PROCESSING ROUTINE

To further validate an interrupt during the interrupt processing routine, insert a command such as KEY(1) ON. As a result, the interrupt processing routine can be executed from the beginning by applying an interrupt during the interrupt processing routine.

■ **Program example**

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'SUBROUTINE
105 KEY(1) ON
110 BEEP:CLS
120 FOR I=10 TO 90 STEP 10
130 CIRCLE (120,100),I
140 NEXT I
150 CLS
160 RETURN 40
```

This is the same as the previous program except that the command, KEY(1) ON, is inserted in line 105.
As a result, when the F1 key is pressed again while the circles are being continuously drawn by an interrupt, an interrupt occurs immediately in which the interrupt processing routine from line 100 is executed from the beginning.



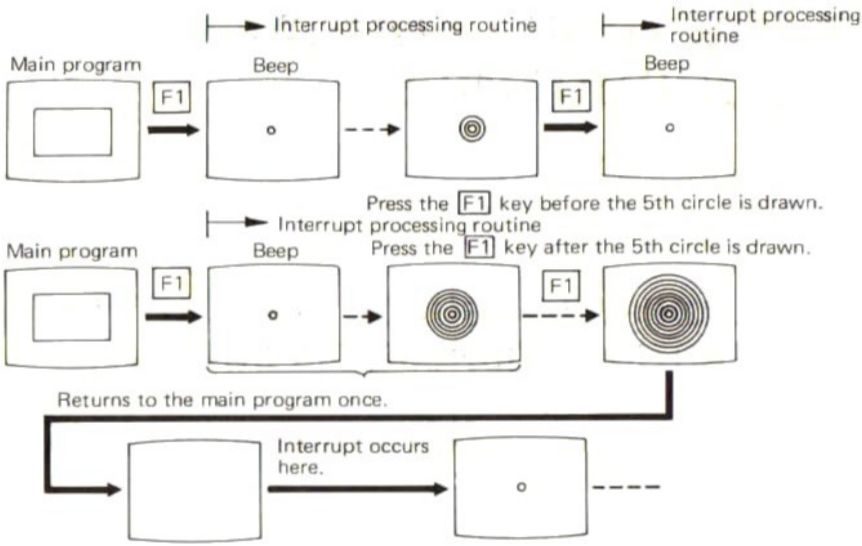
6-6 HOLDING INTERRUPT IN A PROGRAM

To enter the hold state again after validating the interrupt with an – ON statement during the interrupt processing routine, insert a – STOP statement.

■ **Program example**

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 SCREEN 2
40 LINE (50,50)-(200,150),,B
50 GOTO 40
100 'SUBROUTINE
105 KEY(1) ON
110 BEEP:CLS
120 FOR I=10 TO 90 STEP 10
130 CIRCLE (120,100),I
135 IF I=50 THEN KEY(1) STOP
140 NEXT I
150 CLS
160 RETURN 40
```

This program is the same as the previous one. However, in this program, when the value of I becomes 50, KEY(1) STOP is executed in line 135. As a result, an F1 key interrupt occurs immediately during interrupt processing execution if it occurs before the 5th circle is drawn. However, an interrupt hold occurs after the 5th circle is drawn and an interrupt does not occur immediately when the F1 key is pressed.



6-7 SPRITE OVERLAP INTERRUPT EXAMPLE

When two or more sprite patterns overlap by 1 dot, an interrupt can be generated by an ON SPRITE GOSUB statement and SPRITE ON.

In the following program, UFOs fly from left and right and a beep sound occurs when the UFOs overlap.

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(&H3C)+CHR$(&H7E)+CHR$
(&H81)+CHR$(&H81)+CHR$(&HFF)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
30 ON SPRITE GOSUB 100
40 SPRITE ON
50 FOR X=0 TO 255
60 PUT SPRITE 0,(X,100),15,0
70 PUT SPRITE 1,(255-X,100),10,0
80 NEXT X
90 END
100 SPRITE OFF
110 BEEP
120 SPRITE ON
130 RETURN
```

7 MACHINE LANGUAGE SUBROUTINES

With MSX-BASIC, a program can be written by using the machine language of Z-80A (the MSX personal computer CPU) to which control is transferred from BASIC, and the execution result of the machine language program can be given to a variable defined by BASIC.

7-1 MACHINE LANGUAGE SUBROUTINE STARTING ADDRESS DEFINITION

First secure an area where the machine language subroutine is written by using a CLEAR statement. Then define the starting address of the subroutine by using a DEFUSR statement.

DEFUSR N = Starting address

N is an integer from 0 to 9. The starting address of 10 subroutines can be defined as a USR function.

```
CLEAR 200,&HDFFF
DEFUSR1=&HE000
```

With these statements, a machine language subroutine from address &HE000 is defined as a USR 1 function.

7-2 MACHINE LANGUAGE SUBROUTINE EXECUTION

Variable = USR N(1)

The defined machine language subroutine is executed by executing the above statement. When the machine language subroutine has been executed, the value of the execution result is given to a variable, and the BASIC program is also continuously executed.

When execution is transferred to a machine language subroutine, the value of "I" specified as a USR function parameter is given to a subroutine.

```
X=USR1(I)
```

The value of variable I is stored at the following memory location by the above statement, and at the same time, data that indicates the type is entered to register A depending on the type of I. The starting address of the area where the value of I is stored is entered to the HL register.

Type of I	Data input to A register*	HL register address indication	Address where the value of I is stored.
Integer type	2	&HF7F6	&HF7F8–&HF7F9
Single-precision type	4		&HF7F6–&HF7F9
Double-precision type	8		&HF7F6–&HF7FD

*The same data is input to the &HF663 memory address.

When I is a string type variable, the above mentioned is as follows.

Data input to A register	Data input to DE register	String descripter	
3	String descripter starting address	1st byte:	Length of character string
		2nd and 3rd bytes:	Starting address of the area where the character string is stored.

When execution of the machine language subroutine has been terminated, the value of the result is given to variable X by setting the register and memory during termination.

Result value type	&HF663 memory address	DE register	HL register	Result storage address
Integer type	2		&HF7F6	&HF7F8–&HF7F9
Single-precision type	4		&HF7F6	&HF7F6–&HF7F9
Double-precision type	8		&HF7F6	&HF7F6–&HF7FD
String type	3	String descripter starting address		Area start address indicated by the 2nd and 3rd string descripter byte.

7-3 MACHINE LANGUAGE PREPARATION

A machine language subroutine is written to memory by using a POKE statement.

A return from a machine language subroutine to the BASIC program is accomplished with a RET instruction.

CHAPTER 3

COMMANDS, FUNCTIONS AND STATEMENTS

1. SYSTEM CONTROL AND SYSTEM VARIABLES

2. PROGRAM CONTROL

3. FUNCTION AND STRING CONTROL

4. SOUND CONTROL

A B C D E F G H I K L M N O P R S T U V W

5. GRAPHIC AND SPRITE

6. FILE AND I/O CONTROL

In this chapter, MSX-BASIC commands and functions are explained in an alphabetical sequence.

INTRODUCTORY REMARKS

In regard to a function, Function is written in front of the function name.

Command, function name

PSET (point set)

Puts a dot on the graphic screen. ← Function

□ FORMAT

PSET [STEP] (X-coordinate, Y-coordinate) [, color] ← Format

X-coordinate, Y-coordinate   Cond.   Numerical constants, variables, array variables and their expressions from −32768 to 32767.

Input condition

Color   Cond.   Integers from 0 to 15.

Omit   Current foreground color ← When input is omitted

□ FUNCTION AND UTILIZATION

Supplementary command and function explanations, and execution examples in which commands and functions are utilized.

Input item omission

An input item inside [ ] in the FORMAT section can be omitted.

Example

For SCREEN [Mode], [Sprite size], [Key click switch], [Baud rate], [Printer type], when only the mode and sprite size are specified, it is as follows.

```
SCREEN 2,3
```

Items after this, including commas, can be omitted.

When only the printer type is specified, it is as follows.

```
SCREEN ,,,,1
```

Commas cannot be omitted.

Input item omission

Example

DATA Constant [, Constant]

As many constants as desired can be repeated after DATA within the input range per line.

A

Function ABS (absolute)

Gives the absolute value for numeric data.

☐ FORMAT

ABS(X)

X [Cond.] Numeric constants, variables, array variables, and their expressions.

Given value: Numeric type

☐ FUNCTION AND UTILIZATION

Gives X when $X \geqq 0$ and $-X$ when $X < 0$.

EXECUTION EXAMPLE

```
PRINT ABS(2)
 2

PRINT ABS(3-10)
 7
```

Function ASC (ascii)

Gives the character code for the first character of string data.

☐ FORMAT

ASC(X$)

X$ [Cond.] String constants, variables, array variables, and their expressions.

Given value: Single-precision integers, decimal expressions.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT ASC("d")
 100 ←—— Character code of "d".

PRINT ASC("data")
 100 ←—— Character code of "d".
```

Function ATN (arc tangent)

Gives the arc tangent value for numeric data.

☐ FORMAT

ATN(X)

X [Cond.] Numeric constants, variables, array variables, and their expressions.

Given value: Numeric type

☐ FUNCTION AND UTILIZATION

The ATN function gives a floating-point type numeric value which indicates an angle in which the value of the trigonometric function, tan, is X. Its unit is a radian. To obtain the result in degree units, multiply 180/π.

EXECUTION EXAMPLE

```
PRINT ATN(1)
 .7853981633974 5 ←—Unit is radians.
PRINT ATN(1)*180/3.14159
 35.000038009905 ←——— Unit is degrees.
```

AUTO

Line numbers are automatically generated from a specified line number with a specified increment.

☐ FORMAT

AUTO [starting line number] [, increment]

Starting line number	[Cond.]	An integer from 0 to 65529.
	[Omit]	0. However if ", increment" is omitted, it is 10.
Increment	[Cond.]	Integers from 1 to 65529.
	[Omit]	10

☐ FUNCTION AND UTILIZATION

Used to eliminate the keying in of line numbers while entering a program.

- When a program statement exists for a generated line number, "*" appears on the right of the line number. To modify this program statement, move the cursor to "*", then input a new statement after deleting "*" with a space. When no modification is required, press [RETURN].
- To stop automatic line number generation, press [STOP] while pressing [CTRL] or press [C] while pressing [CTRL].

EXECUTION EXAMPLE

```
AUTO 100,50
100 PRINT"12345"
150*
   ↑———Indicates that line number 150 exists.
```

BASE (base)

Used to read and write a VDP table base address.

□ FORMAT

BASE(N)

BASE (N)= expression

N [Cond.] Integers from 0 to 19.

Expression [Cond.] Integers from 0 to 65535.

□ FUNCTION AND UTILIZATION

Used to read or rewrite a VDP table base address in memory.
BASE(N) corresponds with the base addresses shown in the table below depending on the value of N.

Value of N	Table
0	40 characters x 24 lines text mode pattern name table.
2	40 characters x 24 lines text mode pattern generator table.
5	32 characters x 24 lines text mode pattern name table.
6	32 characters x 24 lines text mode color table.
7	32 characters x 24 characters text mode pattern generator table.
8	32 characters x 24 characters text mode sprite attribute table.
9	32 characters x 24 characters text mode sprite pattern table.
10	High resolution graphic mode pattern name table.
11	High resolution graphic mode color table.
12	High resolution graphic mode pattern generator table.
13	High resolution graphic mode sprite attribute table.
14	High resolution graphic mode sprite pattern table.
15	Multi color mode pattern name table.
17	Multi color mode pattern generator table.
18	Multi color mode sprite attribute table.
19	Multi color mode sprite pattern table.

N=1, 3, 4, 16 are not used.

Precautions

The register contents and the table base address of the TMS9929A, which is the screen display LSI, can be directly modified by using a BASE variable and a VDP variable. However, adequate knowledge of the TMS9929A is required to perform this. If the base address is carelessly rewritten, a normal screen display can not be performed. Therefore, precautions shall be taken.

BEEP (beep)

A beep is sounded.

□ FORMAT

BEEP

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
FOR I=0 TO 9
BEEP
NEXT I
```

This program generates a beep sound 10 times continuously.

B

Function BIN$ (binary dollar)

Gives a binary expression of numeric data as string type data.

□ FORMAT

BIN$(X)

X	Cond.	Numeric constants, variables, array variables, and their expressions from −32768 to 65535. For a negative number, it has the same value as if its value was added to 65536.
Given value:		String type

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT BIN$(100)
1100100

PRINT BIN$(-32768)
1000000000000000
```

BLOAD (binary load)

Loads a machine language program, or loads and executes it.

□ FORMAT

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

Device name	Cond.	CAS: . . . Cassette tape
File name	Cond.	String within 6 characters. If 7 or more characters are specified, the 7th character and after and ignored.
	Omit	Loads the file which was found first.
R option	Omit	Load only.
Offset	Cond.	Integers.
	Omit	0

□ FUNCTION AND UTILIZATION

Loads a machine language program saved by a BSAVE statement at an address between the starting address and an end address specified by a BSAVE statement. If offset is specified, the value is added to the starting address and end address.

- If, R is specified, the program is executed after load termination. At that time, the execution start address is an address specified by a BSAVE statement.

B
C

BSAVE (binary save)

Saves the content within a specified memory range with binary.

□ FORMAT

BSAVE "device name [file name]", starting address, end address, [execution start address]

Device name	Cond.	CAS: . . . Cassette tape
File name	Cond.	String within 6 characters. If 7 or more characters are specified, the seventh character and after are ignored.
	Omit	Null string.
Starting address, end address		
	Cond.	Integers
Execution start address		
	Cond.	Integers from −32768 to 65535.
	Omit	Considered as a starting address.

□ FUNCTION AND UTILIZATION

Saves the content within a memory range from a starting address to an end address with binary code which is used for saving machine language.

- If an execution start address is specified, execution starts from the address specified when the machine language program was loaded by a BLOAD statement with an R option. If omitted, the starting address is considered as an execution start address.

EXECUTION EXAMPLE

screen ,,,2 (run)
[illegible]

```
BSAVE"CAS:PROG4",&HE000,&HE800,&HE100
```

CALL (call)

Executes an extended command.

□ FORMAT AND FUNCTION

CALL extended command [(argument, argument. . .)]

Argument	Cond.	Integer constants, variables, array variables, and their expressions. Character constants, variables, array variables, and their expressions.

When an extended command is provided by a ROM cartridge etc., it can be executed by a CALL statement.

- _(underline) can be utilized instead of a character CALL.

Function CDBL (convert to double precision)

Converts numeric data to double precision data.

☐ FORMAT

CDBL(X)

X	Cond.	Numeric constants, variables, array variables, or their expressions.
Given value:		Double precision numeric type

☐ FUNCTION AND UTILIZATION

Given numerical data is internally treated as double precision data by the CDBL function.

C

Function CHR$ (character dollar)

Gives the character of a specified character code.

☐ FORMAT

CHR$(X)

X	Cond.	Numeric constants, variables, array variables, and their expressions from 0 to 255.
Given value:		String type

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT CHR$(100)
d
```

See the character code table (page 165).

Function CINT (convert to integer)

Converts numeric data to integer type data.

☐ FORMAT

CINT(X)

X	Cond.	Numerical constants, variables, array variables, and their expressions from −32768 and less than 32768.
Given value:		Integer type

☐ FUNCTION AND UTILIZATION

When numeric data X is an integer value, it is maintained as it is. When it is a floating point type value, it is converted to an integer value by omitting values below the decimal point. It differs from the INT function in that the INT function gives the whole number out of X while CINT converts X to an integer in which the internal processing is different.

EXECUTION EXAMPLE

```
PRINT CINT(9/2)
 4
PRINT CINT(12*200*55)
Overflow
```

C

CIRCLE (circle)

Draws a circle, oval, a part of a circular arc or a fan shape on the foreground in the graphic mode.

□ FORMAT

CIRCLE [STEP] (central coordinate), radius, [color], [start angle], [end angle], [aspect ratio]

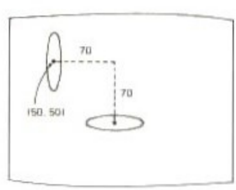
Central X-coordinate, central Y-coordinate	Cond.	Numerical constans, variables, array variables, their expressions from −32768 to 32767.
Radius	Cond.	Numerical constans, variables, array variables, their expressions from −32768 to 32767.
Color	Cond.	Integers from 0 to 15.
	Omit	Current foreground color
Start angle	Cond.	From -2π to 2π (unit is radians).
	Omit	0
End angle	Cond.	From -2π to 2π (unit is radians).
	Omit	2π
Aspect ratio	Cond.	Positive numerical constants, variables, array variables, their expressions. If the aspect ratio is omitted, an oval is drawn.
	Omit	1

□ FUNCTION AND UTILIZATION

Draws a circle with a specified radius and with specified coordinates as its center. When a start angle and end angle are specified, only a part of a circular arc is drawn. A fan shape can be drawn by placing − (minus) for the start angle and end angle. An oval can be drawn with an aspect ratio by specifying the power of the vertical radius for the horizontal radius.
* See page 29 for STEP specifications.

EXECUTION EXAMPLE

```
10 CLS
20 SCREEN 2
30 CIRCLE (50,50),30,,,,4
40 CIRCLE STEP(70,70),30,,,,.25
50 GOTO 50
```



CLEAR (clear)

Initializes all variables and sets the size of the character area and the highest memory address used in BASIC. Also, closes all open files, if any.

☐ FORMAT

CLEAR [size of character area] [, highest address]

Size of character area	Cond.	Numeric constants, variables, array variables, their expressions.
	Omit	Current set value (initial state is 200). However, the character area size cannot be independently omitted.
Highest address	Cond.	Numerical constants, variables, array variables, their expressions.
	Omit	Current set value.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
CLEAR 400,55296
```

All variables are initialized by this statement. Also, the size of the character string area is set to 400 bytes and the highest address of the BASIC program area is set to 55296.

CLOAD (cassette load)

Loads an MSX-BASIC program from cassette tape.

☐ FORMAT

CLOAD ["file name"]

File name	Cond.	String within 6 characters. If 7 or more characters are specified, the seventh character and after are ignored.
	Omit	Loads the first program file found.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
CLOAD"PROG1"
```

Loads the program with the PROG1 file name from cassette tape to memory.

- When an error occurs during load, rewind the tape to reload it.

C

CLOAD? (cassette load verify)

Compares a program saved on cassette tape with one in memory.

☐ FORMAT

CLOAD? ["file name"]

File name	Cond.	String within 6 characters. If 7 or more characters are specified, the seventh character and after are ignored.
	Omit	Compares the first program file found with one in memory.

☐ FUNCTION AND UTILIZATION

A command that checks if a program is correctly saved or not. When it is executed, the program in memory is compared with a program saved on cassette tape with a specified file name.

- After comparison shows that the programs match, OK is displayed and input wait occurs. When they do not match, "Device I/O error" is displayed and input wait occurs.
- If the file name is omitted or CLOAD? "␣" is input, the first program file found on a tape is compared with the program in memory. (␣ means a space.)

EXECUTION EXAMPLE

```
CLOAD?"PROG1"
```

CLOSE (close)

Closes a file which was opened by an OPEN statement.

☐ FORMAT

CLOSE[#] [file number] [, file number]. . . .

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES = statement
	Omit	Closes all the files.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 MAXFILES=3
20 SCREEN 2
30 OPEN "GRP:"FOR OUTPUT AS #1 ←——Opens file 1
40 OPEN "GRP:"FOR OUTPUT AS #2 ←——Opens file 2
50 OPEN "GRP:"FOR OUTPUT AS #3 ←——Opens file 3
60 PRINT #1,"ABC"
70 PRINT #2,"DEF"
80 PRINT #3,"GHI"
90 CLOSE ←——Closed all the files.
100 GOTO 100
```

CLS (clear screen)

Erases all displays on the screen.

☐ FORMAT

CLS

- In the graphic mode, the background color is changed by executing CLS after specifying it with a COLOR statement.

C

COLOR (color)

Specifies the color of the foreground, background, and border area.

☐ FORMAT

COLOR [foreground color], [background color], [border color]

Foreground color, background color, border color

Cond.	Integers from 0 to 15. (See the color table below.)
Omit	Current color

- Color code table

Code	Color	Code	Color	Code	Color	Code	Color
0	Transparent	4	Dark blue	8	Medium red	12	Dark green
1	Black	5	Light blue	9	Light red	13	Magenta
2	Medium green	6	Dark red	10	Dark yellow	14	Gray
3	Light green	7	Sky blue	11	Light yellow	15	White

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

`COLOR 6` ←Only the foreground color (character color in text mode, and graphics color in graphic mode) is changed.

`COLOR ,2` ←Only the background color is changed.

`COLOR ,,11` ←Only the border area color is changed.

`COLOR 15,4,4` ←Initialized.

- See page 26 for the screen configuration.
- In the graphic mode, the background color is not changed by only specifying the background color with a COLOR statement but is changed only after executing CLS.

CONT (continue)

Restarts a program.

☐ **FORMAT**

CONT

☐ **FUNCTION AND UTILIZATION**

Restarts a program that was interrupted by [CTRL] + [STOP] or by a STOP statement in a program. When a CONT statement is executed, execution starts from the statement next to the interrupted statement. However, if an interrupt occurred during the execution of an INPUT statement, execution starts from the beginning of the statement.

Function COS (cosine)

Gives the value of the cosine for numeric data.

☐ **FORMAT**

COS(X)

X	[Cond.]	Numeric type constants, variables, array variables, their expressions. (Unit is radians.)
Given value:		Floating-point type constants from −1 to 1.

☐ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT COS(3.14/3)
 .50045968900814

PRINT COS(60*3.14/180)
 .50045968900814
```

To give X in degree units, use the formula COS (X * π/180).

CSAVE (cassette save)

Saves an MSX-BASIC program file on cassette tape.

☐ **FORMAT**

CSAVE "file name" [, baud rate]

File name	[Cond.]	String within 6 characters. If 7 or more characters are specified, the seventh character and after are ignored.
Baud rate	[Cond.]	1 (1200 baud) or 2 (2400 baud).
	[Omit]	1 (1200 baud)

☐ FUNCTION AND UTILIZATION

Although up to 6 characters can be used for a file name, a numeral cannot be used at the beginning. As for the baud rate, when 1 is specified, the baud rate is 1200 baud, and when 2 is specified, it is 2400 baud.

EXECUTION EXAMPLE

```
CSAVE "PROG1"
```

Saves a BASIC program in memory to cassette tape with a file name "PROG1".

Function CSNG (convert to single precision)

C

Converts numeric data to single precision data.

☐ FORMAT

CSNG(X)

X [Cond.] Numeric type constants, variables, array variables, their expressions.

Given value: Single-precision type.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 PRINT SQR(3)
20 PRINT CSNG(SQR(3)
RUN
 1.7320508075688
 1.73205
```

Function CSRLIN (cursor line)

Gives the Y-coordinate of the cursor location.

☐ FORMAT

CSRLIN

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 CLS
20 INPUT A$
30 PRINT A$;
40 CL=CSRLIN
50 LOCATE 0,CL+3:PRINT "END"
```

The character data displayed by line 30 occupies only one line or plural lines depending its length. However, the Y-coordinate (vertical location) of the cursor after display is input to variable CL and "END" is displayed with a value which is greater than CL by 3 as the Y-coordinate. Therefore "END" is displayed 3-lines below notwithstanding the A$ data length.

DATA (data)

Gives data read by a READ statement.

☐ FORMAT

DATA constant [, constant] . . . .

Constant	Cond.	Numeric or string type.

☐ FUNCTION AND UTILIZATION

- When data items are arranged in one DATA statement, they are punctuated by a comma (,).
- If data in a DATA statement sequentially matches variables in a READ statement, it can be located anywhere for a READ statement and as many DATA statements as desired can be utilized.
- When string type data includes a comma (,) or colon (:), or when a space is inserted in front and at the back, it is placed inside quotation marks (").

EXECUTION EXAMPLE

```
10 CLS
20 SCREEN 2
30 READ A,B,C,D
40 LINE (A,B)-(C,D)
50 DATA 0,0,255,191
60 GOTO 60
```

DEF FN (define function)

Defines a user function.

☐ FORMAT

DEF FN function name [(parameter [, parameter] . . . .)] = expression.

Function name	Cond.	Numeric type, string type variables (Type is in accord with the expression.)
Parameter	Cond.	Up to 9 variables.
Expression	Cond.	Numeric type, string type constants, variables, array variables, their expressions.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 DEF FNA(X,Y)=(X*2+Y*3)/(X-Y)
20 B=FNA(4;2)
30 PRINT B
RUN
 7
```

In line 10, the function FNA(X,Y) is defined as the following expression. In line 20, 4 and 2 are given as values for the X and Y parameter, then the function is called. The result, 7, is assigned to variable B.

DEFINT (define integer)
DEFSNG (define single precision)
DEFDBL (define double precision)
DEFSTR (define string)

Defines the correspondence of the first character of the variable name and the variable type.
(INT: Integer type, SNG: Single precision, DBL: Double precision, STR: String type.)

D

☐ FORMAT

DEFINT character [– character]
DEFSNG character [– character]
DEFDBL character [– character]
DEFSTR character [– character]

Character | Cond. | One alphabetical character.

☐ FUNCTION AND UTILIZATION

```
DEFINT A-C
```
As a result, all the variables, starting with characters A–C, are integer type.

Priority of type declaration characters (%, !, #, $)
After declaring DEFINT A, A becomes a double-precision variable by declaring A# later.

EXECUTION EXAMPLE

```
10 DEFINT A-C              ← Variables from A to C are integer type.
20 A=1.23456789            } ← Variables A, ABC become integer type
30 ABC=1.23456789          }   by line 10.
40 B#=1.23456789           ← Double-precision type by placing #.
60 C!=1.23456789           ← Single-precision type by placing !.
70 PRINT A;ABC;B#;C!
RUN
 1  1  1.23456789  1.23457
```

D

DEFUSR (define user)

Specifies a starting address when a machine language subroutine to be called by a USR function.

□ FORMAT

DEFUSR [X] = starting address.

X	Cond.	Integers from 0 to 9.
	Omit	0
Starting address	Cond.	Numeric type constants, variables, their expressions from 0 to 65535.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
DEFUSR1=&HE000
```

As a result, a machine language subroutine which starts from address &HE000 is defined as USR1.

- The starting address can be redefined as many times as required in one program without changing the value of user number (X).
  (See page 56 for Machine Language Subroutines.)

DELETE (delete)

Erases a specified line in a program.

□ FORMAT

DELETE [line number] [– line number]

Line number	Cond.	Integers from 0 to 65529.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

`DELETE 40`	Erases line 40
`DELETE 20-40`	Erases lines from 20 to 40.
`DELETE -50`	Erases lines from the starting line to line 50.
`DELETE.`	Erases a line displayed last by a LIST statement or a line that was interrupted due to an error.

- When only one line is to be erased, input the line number only and press RETURN.

DIM (dimension)

Declares the name of an array variable, data type, size and dimension.

☐ FORMAT

DIM variable name (maximum value of a subscript [, maximum value of a subscript] . . .) [, variable name ( ), ...]

Variable	Cond.	Numeric or string type.
Maximum value of a subscript	Cond.	Integer type constants, variables, array variables, and their expressions over 0.
Maximum dimension		255 dimension.

D

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

`DIM A(15)` — Sets up an area of 16 numeric type array variables from A(0) to A(15) in memory. The initial value of variables is 0.

`DIM B$(2,3)` — Sets up an area of 12 variables as shown below (string type). The initial value of variables is a null-string.

B$(0,0)	B$(1,0)	B$(2,0)
B$(0,1)	B$(1,1)	B$(2,1)
B$(0,2)	B$(1,2)	B$(2,2)
B$(0,3)	B$(1,3)	B$(2,3)

To define a plural number of array variables by one DIM statement

`DIM A(2),B$(4,2),C(3,3)` — Each variable is punctuated with a comma.

Multi-dimensional array variables

Multi-dimensional array variables are generated by specifying 2 Maximum values or more for subscript.

`DIM X(3,4,5)` — 3 dimension

DIM statement omission

When an array variable is utilized without declaring a DIM statement, the maximum value of the subscript is considered to be 10.

DRAW (draw)

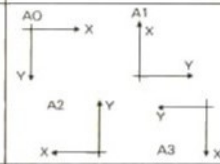
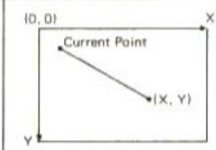
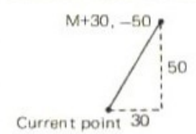
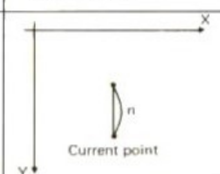
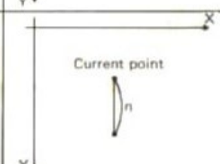
Draws graphics on the graphic screen as specified in graphic subcommands.

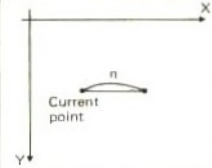
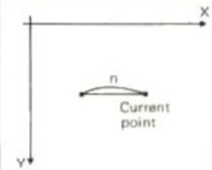
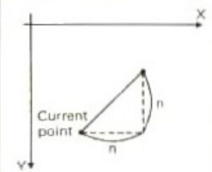
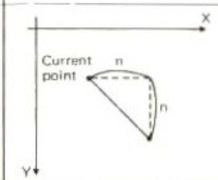
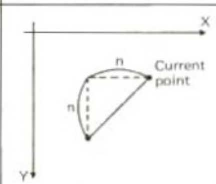
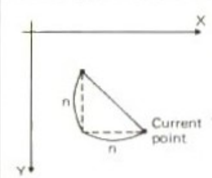
☐ FORMAT

DRAW subcommand

Subcommand	Cond.	Character string (constants) inside " " or string type variables in which a character string is assigned. Capitals or small characters.

Subcommands

Command	Condition	Semantics	
Sn (scale)	$0 \leqq n \leqq 255$	Specifies the number of dots for 1 unit when a line is drawn. 1/4 dot with n=1. Initialization is S4.	
An (angle)	$0 \leqq n \leqq 3$	Rotates coordinate system by step of 90° for a standard coordinate axis (0°). Initialization is A0.	
Cn (color)	$0 \leqq n \leqq 15$	Specifies a color for a line drawn by a color code. Initialization is C15.	
M x, y (move)	$0 \leqq x \leqq 255$ $0 \leqq y \leqq 191$	Draws a line from a current point to an absolute location (x, y).	
M±x, ±y (move)	$0 \leqq x \leqq 255$ $0 \leqq y \leqq 191$	Shifts horizontally ±x from a current point and ±y vertically. The unit for x, y is the number of dots specified by the S subcommand.	
Un (up)		Draws a line toward a negative direction on the Y axis from a current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted.)	
Dn (down)		Draws a line toward a positive direction on the Y-axis from a current point to another point by an n distance. The unit for n is the number of dots specified by the S subcommand. (1 if omitted.)	

Rn (right)		Draws a line in a positive direction on the X axis from the current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted.)	
Ln (left)		Draws a line in a negative direction on the X-axis from the current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted.)	
En		Draws a line in a positive direction on the X-axis and in a negative direction on the Y-axis from the current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted.)	
Fn		Draws a line in a positive direction on the X-axis and in a positive direction on the Y-axis from a current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted.)	
Gn		Draws a line in a negative direction on the X-axis and in a positive direction on the Y-axis from a current point to another point by an n distance. The unit of n is the number of dots specified by the S subcommand. (1 if omitted).	
Hn		Draws a line in a negative direction on the X-axis and in a negative direction on the Y-axis from a current point to another point by an n distance. The unit of n is the number of dots specified by the S subcomma nd. (1 if omitted.)	

D

☐ FUNCTION AND UTILIZATION

The current location is always stored with a command to draw a line except Sn, An, Cn. For example,

```
DRAW "M100,120"
```

by the message above, when a line is drawn from a certain point to another point (100, 120), then this point becomes the current point. Then, when a command to draw a line is made again, a line is drawn from this current point to a specified point.

One of the following two commands can be placed in front of a command to draw a line.

B .... Although the current point is shifted, a line is not drawn. (Example: BM0, 0)

N .... Although a line is drawn, the current point is not shifted. (Example: NU30, 30, NR30, 30)

To express a subcommand with a variable

```
A$="BM100,150U50E50F50D50L100"
DRAW A$
```

D

In this example, a subcommand is assigned once to a string type variable A$, then A$ is specified as a subcommand in a DRAW statement.

To express a part of a subcommand with a variable (X variable;)

```
A$="U20R20D20L20"
DRAW "BM50,50XA$;"
DRAW "BM150,100XA$;"
```

When a subcommand assigned to a string type variable is used inside " " of a DRAW statement, add X before and ";" after that. In this example, a subcommand assigned to A$ is used in two DRAW statements.

To express n in a subcommand with a variable (=variable;)

n which expresses the shift distance, angle and color code with each subcommand can be a constant or a variable in a DRAW statement. When it is expressed with a variable, add = before and ";" after that.

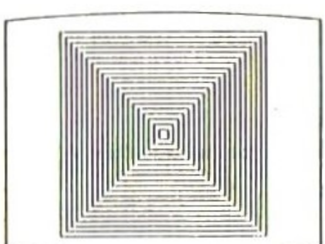
```
X=40
DRAW "U=X;"
```

is the same as

```
DRAW "U40"
```

EXECUTION EXAMPLE

```
10 SCREEN 2 ←—— Graphic mode when a DRAW statement is used.
20 DRAW "BM125,100"←— to (125, 100) without drawing anything.
30 FOR I=4 TO 240 STEP 12      }
40 DRAW "S=I;BURD2L2U2RBD"     }←— continuously draws squares
50 NEXT I                      }    with different sizes.
60 GOTO 60
```



END (end)

Terminates program execution to enter a command wait state and closes all the opened files.

☐ FORMAT

END

☐ FUNCTION AND UTILIZATION

The END statement is used in the last line of the main program when a subroutine is written after a main program to prevent a subroutine from being executed again after the main program is terminated. It can be used as many times as desired in one program such as when a program execution result is branched into some result, it can be used at the end of each branch.

- A RUN or GOTO statement is used to execute it again. It cannot be resumed by a CONT statement.

E

```
      .
      .
100 GOSUB 1000
      .
      .
190
200 END
1000 'SUBROUTINE
      .
      .
1100 RETURN
```

In this program, if an END statement does not exist in line 200, the subroutine from line 1000 is entered without a GOSUB statement after returning from a subroutine and executing line 190, and an occurs.

Function EOF (end of file)

When the last data of a file has been read, −1 is given, otherwise 0 is given.

☐ FORMAT

EOF (file number)

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement
Given value:		Integer type (−1 or 0)

☐ FUNCTION AND UTILIZATION

```
IF EOF(1) THEN CLOSE #1
```

When the last data is read while data is being read from the file whose file number is 1, a file is closed by the above statement.

ERASE (erase)

Erases an array variable.

☐ **FORMAT**

ERASE array variable name [, array variable name] .....

☐ **FUNCTION AND UTILIZATION**

```
10 DIM A(100),B$(4,3)
        .
        .
100 ERASE A,B$
```

In this example, array variables A and B$, declared in line 10, are erased in line 100. After this, the memory area can be used for another purpose. Also, an array variable with the same name can be redefined by a DIM statement.

E

Function ERL (error line)

Gives the line number of a line where an error occurred.

☐ **FORMAT**

ERL

Given value: Numeric type.

☐ **FUNCTION AND UTILIZATION**

When no error has occurred, 0 is given. When an error results from a direct command, 65535 is given. Is used by combining it with an ON ERROR statement or an ERROR statement.

Function ERR (error)

Gives the error number of an error that occurred.

☐ **FORMAT**

ERR

Given value: Integer type

☐ **FUNCTION AND UTILIZATION**

Can be used for error processing in a program by combining it with an ERROR statement or ERL function.

- When no error occurs, 0 is given.

EXECUTION EXAMPLE

```
PRINT 10/0
Division by zero
PRINT ERR
 11
```

ERROR (error)

Simulates an error of a specified error number or defines an error number.

☐ FORMAT

ERROR error number

Error number	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

`ERROR 1` ——— Generates a NEXT without FOR error. (Stops program execution.)

User definition of error number

If A < 0 THEN ERROR 250

When a negative numeral is assigned to variable A based on the above, error 250 occurs. (Since error numbers up to 59 are defined in MSX-BASIC, numbers larger than those shall be used.)

EXECUTION EXAMPLE

When a negative numeral is input in the following program, a message is displayed that indicates a positive numeral is required, and program execution continues.

```
10 ON ERROR GOTO 90
20 FOR I=1 TO 10
30 INPUT A
40 IF A<0 THEN ERROR 250
50 SUM=SUM+A
60 NEXT I
70 PRINT SUM
80 END
90 IF ERR=250 THEN PRINT "Input
a positive number.":RESUME 30
100 PRINT "Error!"
```

Function

EXP (exponential)

Gives e^x which is the natural exponential function of X.

☐ FORMAT

EXP(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions below 145.06286085862.
Given value:		Floating-point type.

- e (2.7182818284588) is the base of a natural logarithm.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT EXP(100)
 2.6881171418087E+43
```

Function FIX (fix)

Gives the integer of numeric data.

☐ FORMAT

FIX(X)

X	Cond.	Numeric type constant, variables, array variables, their expressions.
Given value:		Numeric type.

☐ FUNCTION AND UTILIZATION

Gives the value of numeric data X in which the figure below the decimal point is truncated.

EXECUTION EXAMPLE

```
PRINT FIX(3);FIX(-3);FIX(3.58);FI
X(-3.58)
 3 -3  3 -3
```

FOR-NEXT (for-next)

Repeats program execution between a FOR statement and a corresponding NEXT statement.

☐ FORMAT

FOR variable = initial value TO end value [STEP increment]

⋮

NEXT [variable]

Variable	Cond.	Numeric type. FOR statement variables shall be the same as those in the NEXT statement.
Initial value, end value	Cond.	Numeric type constants, variables, and their expressions.
Increment	Cond.	Numeric type constants, variables, their expressions.
	Omit	1

☐ FUNCTION AND UTILIZATION

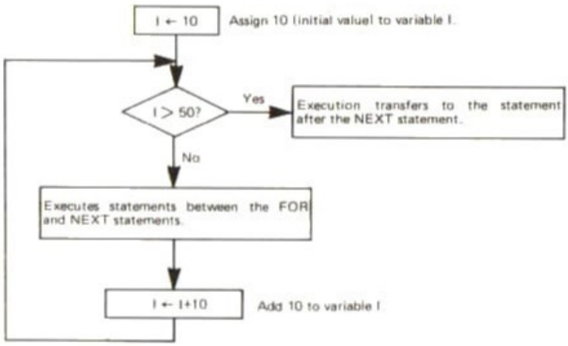
A program between a FOR statement and a NEXT statement is repeatedly executed while the value of the variable specified in the FOR statement is increased from an initial value to an end value. The value of the variable is increased by a specified amount each time program execution is terminated.

- Although the variable in the NEXT statement can be omitted, the correspondence between FOR and NEXT can be easily understood in a program list if it is written.

EXECUTION EXAMPLE

```
10 FOR I=10 TO 50 STEP 10
20 PRINT "I=";I
30 NEXT I
```

This program is executed as follows.



Multi-loop
A FOR – NEXT loop can be placed inside a FOR – NEXT loop. In this case, the inner loop must be completely included inside the outer loop. A different variable is used for each loop.

```
10 FOR I=1 TO 5
20 FOR J=1 TO I
30 PRINT "*";        — Inner loop    — Outer loop
40 NEXT J
50 PRINT
60 NEXT I
RUN
*
**
***
****
*****
```

Several FOR statements can be terminated by one NEXT statement. In this case, the variable name cannot be omitted in the NEXT statement. Variables are arranged sequentially with the inner loop first by punctuating them with commas.

```
FOR I=0 TO 10
FOR J=0 TO 5
      .
      .
NEXT J,I
```

Function FRE (free)

Gives the number of bytes in an unused area of memory which can be used in MSX-BASIC.

☐ FORMAT

FRE(X)

FRE(" ")

X	Cond.	Arbitrary numeric value.
Given value:		Integer type.

☐ FUNCTION AND UTILIZATION

```
PRINT FRE("0")
```
— Displays the number of bytes in an unused area of memory.

```
PRINT FRE("")
```
— Displays the number of bytes in an unused part of a character string area in memory.

F
G

GOSUB-RETURN (go to subroutine-return)

Transfers execution to a specified subroutine.
The RETURN statement indicates the end of the subroutine in which execution is returned to a location next to GOSUB or to a specified line number.

☐ FORMAT

GOSUB line number

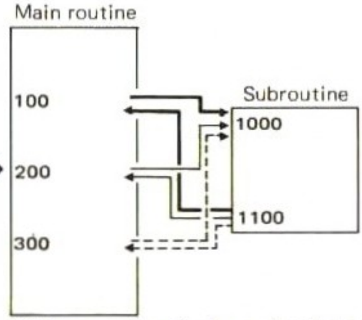
⋮

RETURN [line number]

Line number	Cond.	Integers from 0 to 65529.
	Omit	When omitted in a RETURN statement, it is the line number next to the GOSUB statement.

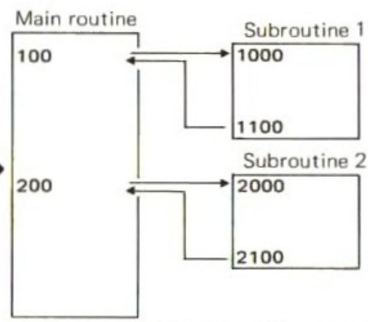
□ FUNCTION AND UTILIZATION

```
        .
        .
100 GOSUB 1000
        .
        .
200 GOSUB 1000
        .
        .
300 GOSUB 1000
        .
        .
1000 ' SUBROUTINE
        .
        .
1100 RETURN
```



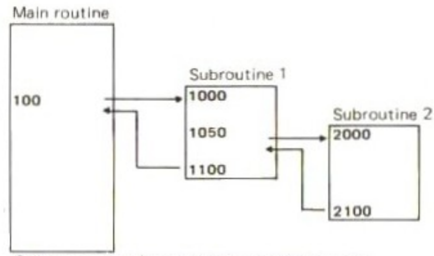
Calls one subroutine by a plural number of GOSUB statements.


```
        .
        .
        .
100 GOSUB 1000
        .
        .
200 GOSUB 2000
        .
        .
1000 ' SUBROUTINE 1
        .
        .
1100 RETURN
        .
        .
2000 ' SUBROUTINE 2
        .
        .
2100 RETURN
```



A plural number of subroutines are placed in one program.

```
    .
100 GOSUB 1000
    .
    .
1000 '  SUBROUTINE 1
    .
    .
1050 GOSUB 2000
    .
    .
1100 RETURN
    .
    .
2000 '  SUBROUTINE 2
    .
    .
2100 RETURN
```



Calles another subroutine from one subroutine.
(GOSUB statement multiplexing.)

GOSUB statement multiplexing performance depends on the existing memory.

GOTO (go to)

Transfers program execution to a specified line number.

☐ FORMAT

GOTO line number

Line number [Cond.] Integers from 0 to 65529.

☐ FUNCTION AND UTILIZATION

Program execution is transferred to a line specified by a GOTO statement.

- When executed in the direct command mode, execution starts from a specified line.

Function HEX$ (hexadecimal dollar)

Gives hexadecimal expression of numeric data as string type data.

☐ FORMAT

HEX$(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 65535. In the case of negative numerals, their value is the same as if it is added to 65536.
Given value:		String type.

☐ FUNCTION AND UTILIZATION

```
PRINT HEX$(100)
64

PRINT HEX$(-32768)
8000


PRINT HEX$(255)
FF
```

H
I

IF-THEN-ELSE (if-then-else)

Branches execution according to the values of an expression.

☐ FORMAT

IF expression THEN statement [ELSE] statement]

Expression	Cond.	A relational expression for which the result becomes a numeric expression, logical expression, or arithmetic expression.
ELSE statement	Omit	To the statement after THEN if the expression value is true, and to the next line if it is false.

☐ FUNCTION AND UTILIZATION

If the value of an expression is true (except 0), the statement after THEN is executed and if the value of an expression is false (0), the statement after ELSE is executed. Then execution is transferred to the next line.

- When the ELSE statement is omitted, the statement after THEN is executed if the expression value is true. If it is false, the statement after THEN is ignored and execution is transferred to the next line.
- In the IF – THEN GOTO format, THEN or GOTO can be omitted.

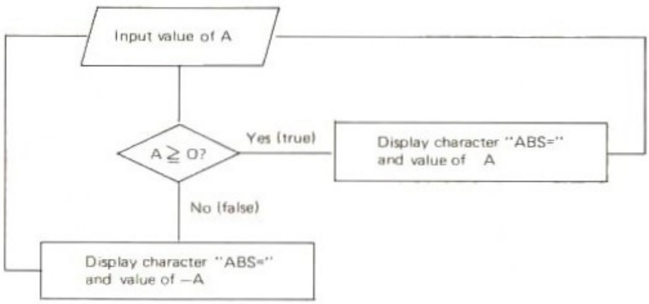
IF A=0 THEN 30
IF A=0 GOTO 30 — Same meaning.

The statement or line number comes after THEN.
The line number comes after GOTO.

- When the GOTO statement comes after ELSE, GOTO can be omitted.
- When a plural number of statements are written after THEN or ELSE, they are executed sequentially with the left statement first. Statements shall be punctuated with a colon (:).

EXECUTION EXAMPLE

```
10 INPUT A
20 IF A>=0 THEN PRINT "ABS=";A E
LSE PRINT "ABS=";-A
30 GOTO 10
```



IF – THEN statement multiplexing

IF – THEN can be continued after THEN or ELSE. Multiplexing can be performed within the range of one line.

Function INKEY$ (inkey dollar)

Gives the character of a depressed key, and a null string if no key is pressed.

☐ FORMAT

INKEY$

Given value: String type.

☐ FUNCTION AND UTILIZATION

When keys other than CTRL + STOP, SHIFT, and CTRL are pressed, their character is given as data. If no key is pressed, a null string is given.

EXECUTION EXAMPLE

```
10 CLS
20 PRINT "Press any key."
30 K$=INKEY$
40 IF K$="" THEN GOTO 30
50 PRINT K$;
60 GOTO 30
```

Repeats until a key is pressed. (lines 30–40)

When any key is pressed, the character is assigned to variable K$ and displayed on the screen in line 50.

Function INP (input)

Reads data of a specified I/O port.

☐ FORMAT

INP(port number)

Port number	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

Inputs and gives data from a specified I/O port.
See page 164 for I/O port allocations.

INPUT (input)

Inputs the value of a variable from the keyboard.

☐ FORMAT

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

Variable	Cond.	Numeric type, string type, their array variables.
"Prompt statement"	Cond.	Comment statement for data input.
	Omit	Displays only "?" without a prompt statement.

☐ FUNCTION AND UTILIZATION

Input's data from a keyboard and assigns it to a variable. At that time, the space before the data is ignored.

- For an INPUT statement of a numeric type variable, the space in the middle of data is also ignored.
- When a comma is input, it is considered to be punctuation for data, and the items before the comma are considered to be one data assigned to a variable while the comma is not assigned.
- When a prompt statement is written, it is displayed on the screen when data input is requested. If a prompt statement is omitted, only "?" is displayed.
- The number of variables must be in accord with the data.

EXECUTION EXAMPLE

```
10 INPUT A
RUN
?
```

When the prompt statement is omitted.

```
10 INPUT "A=";A
RUN
A=?
```

When the prompt statement is used.

```
10 INPUT "A AND B ";A,B
RUN
A AND B ? 7
??
```

Since the input data is less than the number of variables, the missing data is requested by ??.

```
10 INPUT "A'AND B ";A,B
RUN
A AND B ? 1,2,3,4
?Extra ignored
```

Display when more data is input than the number of variables. (Residual data is ignored.)

Function INPUT$ (input dollar)

1. Inputs a specified number of characters from the keyboard.
2. Inputs a specified number of characters from a file.

□ FORMAT

1. INPUT$(X)
2. INPUT$(X, [#] file number)

X	Cond.	Numeric type constants from variables, array variables, their expressions from 1 to 255.
File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement.
Given value:		String type.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 X$=INPUT$(5)
20 PRINT X$
```

When line 10 is executed, keyboard input wait occurs. After 5 characters are input, they are assigned to variable X$. Characters are not displayed on the screen during keyboard input.

```
10 OPEN "CAS:TEST" FOR INPUT AS #1
20 X$=INPUT$(50,#1)
30 CLOSE
```

In this program, 50 characters are input from a file saved on cassette tape and are assigned to string variable X$. Then the file is closed.

Range of "X"

During initial status, if X is outside a range from 1 to 200, an error occurs. When the size of the character area is set to more than 255 by a CLEAR statement, a value from 1 to 255 can be selected.

INPUT# (input number)

Reads data from a file opened by an OPEN statement, and assigns it to a variable.

☐ FORMAT

INPUT# file number, variable [, variable] . . . .

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement.
Variable	Cond.	Numeric type or string type, their array variables.

☐ FUNCTION AND UTILIZATION

Reads data from a file. If the data is numeric, the space, the return code, and the line feed code before the data are ignored.
If the data is string type, the data from the first character to the character before the space, comma, return code, and line feed code is read as one data. If the characters are inside " ", only these characters are read as data.

EXECUTION EXAMPLE

```
10 OPEN "CAS:TEST" FOR INPUT AS #1 —— Opens a file for read out.
20 IF EOF(1) THEN GOTO 50
30 INPUT #1,A$:PRINT A$ —— Reads data, assigns it to variable A$ and displays it on the screen.
40 GOTO 20
50 CLOSE #1
```

(See page 42 for File processing.)

I

Function INSTR (in string)

Retrieves a specified character string from among character strings and gives its location.

☐ FORMAT

INSTR([N,] X$, Y$)

N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
	Omit	1.
X$, Y$	Cond.	String type constants, variables, array variables, their expressions.
Given value:		Integer type.

☐ FUNCTION AND UTILIZATION

Gives the number of a character from the left where Y$ starts in an X$ character string as numeric data. When N is specified, retrieval starts from Nth character of the X$.

EXECUTION EXAMPLE

```
PRINT INSTR(3,"WHAT IS THIS?","IS")
 6
```

- When the N value is larger than the length of X$ or X$ is a null string, or if Y$ cannot be found, 0 is given.

Function INT (integer)

Gives the maximum integer value smaller than given numeric data.

☐ **FORMAT**

INT (X)

X	Cond.	Numeric type constants, variables, array variables, their expressions.
Given value:		Numeric type.

☐ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT INT(3);INT(-3);INT(3.58);INT(-3.58)
 3 -3  3 -4
```

INTERVAL ON (interval on)
INTERVAL OFF (interval off)
INTERVAL STOP (interval stop)

I
K

Validates, invalidates, or holds an interrupt with a built-in timer.

☐ **FORMAT**

INTERVAL ON – Interrupt valid.
INTERVAL OFF – Interrupt invalid.
INTERVAL STOP – Interrupt hold.

☐ **FUNCTION AND UTILIZATION**

A command that actually validates (INTERVAL ON), invalidates (INTERVAL OFF), or holds (INTERVAL STOP) an interrupt after declaring an interrupt with a built-in timer by using ON INTERVAL GOTO.

(See page 50 for Interrupts.)

KEY (key)

Defines a character string for a function key.

☐ **FORMAT**

KEY function key number, character string

Function key number	Cond.	Integers from 1 to 10.
Character string	Cond.	String within 15 characters.

□ **FUNCTION AND UTILIZATION**

When characters are defined for a function key, a defined character string is entered by just pressing a function key.

- Function keys from 1 to 5 correspond to [F1] – [F5], while numbers from 6 to 10 correspond to the pressing of each function key while pressing the [SHIFT] key.
- When the reset button is pressed or the power is turned off, the function key definitions are erased and initialized.
- A code other than that for a character (such as return code) can be defined by using the CHR$ function.

EXECUTION EXAMPLE

```
KEY 1,"JAPAN" ――――――Defines "JAPAN" for [F1]

KEY 2,"CLS"+CHR$(13) ――――――Defines CLS [RETURN] for [F2]
```

KEY LIST (key list)

Displays the content of the function keys.

□ **FORMAT**

KEY LIST

□ **FUNCTION AND UTILIZATION**

When this command is executed, the character string content defined for each function key is displayed.

EXECUTION EXAMPLE

```
KEY LIST
color
auto
goto
list
run
color 15,4,4
cload"
cont
list.
 run
```

An example of the initial state. It is found that "color 15, 4, 4" is defined for the function key 6 (or the [F1] key pressed together with the [SHIFT] key).

K

KEY ON, KEY OFF (key on, key off)

Displays or erases the content of a function key.

□ **FORMAT**

KEY ON or KEY OFF

☐ FUNCTION AND UTILIZATION

Initially the character strings defined for each function key are displayed with 5 characters on the last line of the screen. Execute KEY OFF to erase this display.

- Characters can be output on this line with a PRINT statement after using KEY OFF to erase the display.
- Execute KEY ON to output this display.

KEY (n) ON (key (n) on)
KEY (n) OFF (key (n) off)
KEY (n) STOP (key (n) stop)

Validates, invalidates or holds a function key interrrupt.

☐ FORMAT AND FUNCTION

KEY (function key number) ON – Interrupt valid.
KEY (function key number) OFF – Interrupt invalid.
KEY (function key number) STOP – Interrupt hold.

Function key number	Cond.	Constants, variables, array variables, their expressions from 1 to 5.

☐ FUNCTION AND UTILIZATION

Specifies a function key used for an interrupt with a function key number.

```
KEY(1) ON ———— Validates an [F1] key interrupt.
KEY(2) OFF ———— Invalidates an [F2] key interrupt.
KEY(3) STOP ———— Holds an [F3] key interrupt.
```

(See page 50 Interrupts.)

Function LEFT$ (left dollar)

Gives an arbitray number of characters taken from the left of string data as string data.

☐ FORMAT

LEFT$(X$, N)

X$	Cond.	String type constants, variables, array variables, their expressions.
N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
Given value:		String type.

□ **FUNCTION AND UTILIZATION**

```
PRINT LEFT$("MSX-BASIC",3)
MSX
Ok
```

```
PRINT LEFT$("MSX-BASIC",3.8)
MSX
Ok
```

If N is not an integer, numbers below the decimal point are omitted.

```
PRINT LEFT$("MSX-BASIC",0)

Ok
```

If N is 0, a null string is given.

Function LEN (length)

Gives the number of characters (length) of character data as numeric data.

□ **FORMAT**

LEN(X$)

X$	Cond.	String type constants, variables, array variables, their expressions.
Given value:		Integer type.

L

□ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT LEN("CHRISTMAS")
 9
```

```
PRINT LEN("THE END")
 7
```

When a character string includes a space, the space is counted as 1 character.

- Also, when a character string includes the CHR$ function, it is counted as one character.

LET (let)

Assigns data for a variable.

FORMAT

[LET] variable = X

Variable	Cond.	Numeric type, character type variables, array variables.
X	Cond.	Numeric type, character type constants, variables, array variables, their expressions.

FUNCTION AND UTILIZATION

Assigns a value on the right to the left.

- For string type constants, they are enclosed inside quotation (") marks.
- LET can be omitted.
- When a certain type of numeric data is assigned to another type of numeric variable, the numeric data is converted to that type of variable.

EXECUTION EXAMPLE

```
LET N=N+1
```
Increases the value of N by 1.

```
A%=45.6:PRINT A%
 45
```

```
A$=3+4
Type mismatch
```
Since numeric type data was assigned to a string type variable, an error occurs.

L

LINE (line)

Draws a straight line or square on the foreground in the graphic mode.

☐ FORMAT

LINE [[STEP] (starting point coordinates)] - [STEP]

(end point coordinates), [color] $\left\{\begin{matrix}\text{[, B]} \\ \text{[, BF]}\end{matrix}\right\}$

Starting point coordinates	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 32767.
	Omit	Last location specified by the last graphic instruction.
End point coordinates	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 32767.
Color	Cond.	Integers from 0 to 15.
	Omit	Current foreground color.
B, BF	Omit	Draws a straight line.

☐ FUNCTION AND UTILIZATION

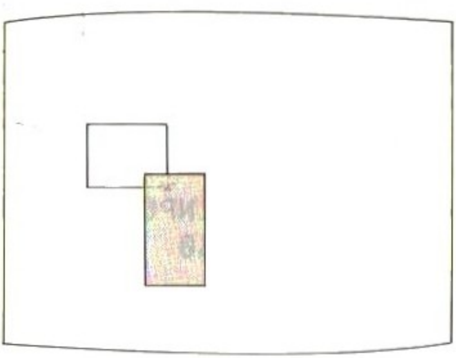
Draws a straight line that connects starting point and end point coordinates (when B, BF is omitted).

- When "B" is specified, draws a square with a straight line that connects two specified points as a diagonal.
- When "BF" is specified, draws a square with a straight line that connects two specified points as a diagonal, and colors the surrounding area.
- See page 29 for STEP specifications.

EXECUTION EXAMPLE

L

```
10 CLS
20 SCREEN 2
30 LINE (60,60)-(100,100),1,B
40 LINE STEP(-10,-10)-(120,160),8 ,BF
50 GOTO 50
```



LINE INPUT (line input)

Gives a string with up to 254 characters by keyboard input as a string type variable.

☐ FORMAT

LINE INPUT ["prompt statement";] variable

"prompt statement"	Cond.	Comment statement for data input.
	Omit	Displays only "?" without a prompt statement.
variable	Cond.	String type variables, array variables.

☐ FUNCTION AND UTILIZATION

A return code is only considered as data punctuation, and assigns a keyboard input character string to a variable. When a comma is included in a character string, it is assigned as part of the character string.

EXECUTION EXAMPLE

```
10 CLS
20 LINE INPUT "NAME,PHONE? ";N$
30 PRINT N$
RUN
NAME,PHONE? JACK,00-11-22
JACK,00-11-22
```

L

LINE INPUT# (line input number)

Reads a string with up to 254 characters from a file, and assigns it to a character type variable.

☐ FORMAT

LINE INPUT #file number, variable

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement.
Variable	Cond.	String type variables, array variables.

☐ FUNCTION AND UTILIZATION

Reads string type data from a file. However, a space, comma, and line feed code are not considered as punctuation for data, which differs from the INPUT# statement, and the character string that includes these items is assigned to a variable as character string data. Only the return code is considered to be punctuation for data.

EXECUTION EXAMPLE

```
10 OPEN "CAS:DATA" FOR INPUT AS # 1
20 IF EOF(1) THEN GOTO 60
30 LINE INPUT #1,A$
40 PRINT A$
50 GOTO 20
60 CLOSE #1:END
```

When a file has been prepared by the following procedure with a file name called DATA.

```
PRINT #1,"ABC";",";"DEF"
PRINT #1,"GHI JKL";
PRINT #1,"MNO"
PRINT #1,"PQR"
```

and when this data is read by the above program and displayed on the screen, it is found that it was read as 3 string type data as follows.

```
ABC,DEF
GHI JKLMNO
PQR
```

LIST (list out)

Displays a currently stored program list.

☐ FORMAT

LIST [starting line number] [–] [end line number]

Starting line number	Cond.	Integers from 0 to 65529.
	Omit	Smallest line number.
End line number	Cond.	Integers from 0 to 65529.
	Omit	Largest line number.

L

☐ FUNCTION AND UTILIZATION

Press STOP to temporarily stop the screen display. Press STOP again to resume it again. Press CTRL and STOP to suspend it.

EXECUTION EXAMPLE

LIST	Displays all lines.
LIST 40	Displays line 40.
LIST 20-40	Displays lines from line 20 to line 40.
LIST -50	Displays lines from the starting line to line 50.
LIST 30-	Displays lines from line 30 to the end line.
LIST.	The last line displayed by a LIST statement or a line with execution interrupted by an error is displayed.

LLIST (line printer list out)

Prints a currently stored program list with a printer.

☐ FORMAT

LLIST [Starting line number] [–] [end line number]

Starting line number	Cond.	Integers from 0 to 65529.
	Omit	Smallest line number.
End line number	Cond.	Integers from 0 to 65529.
	Omit	Largest line number.

☐ FUNCTION AND UTILIZATION

Specification is the same as that for a LIST statement. A list is not displayed on the screen during the execution of an LLIST statement.

- If an LLIST statement is executed when a printer is not connected or when a printer is not operational, the computer stops without accepting keyboard input. If this occurs, input is accepted by pressing the CTRL and STOP key at the same time.

LOAD (load)

Loads a BASIC program file into memory from a specified device.

L

☐ FORMAT

LOAD "device name [file name]"

Device name	Cond.	CAS: . . . Cassette tape.
File name	Cond.	String within 6 characters. If 7 or more characters are specified, the 7th character and after are ignored.
	Omit	Loads the file found first.

☐ FUNCTION AND UTILIZATION

When CAS: is specified as a device name, a program saved by an ASCII format on a cassette tape by SAVE "CAS: file name" is loaded.

EXECUTION EXAMPLE

```
LOAD "CAS:PROG2"
```

LOCATE (locate)

Moves the cursor to a specified location

□ **FORMAT**

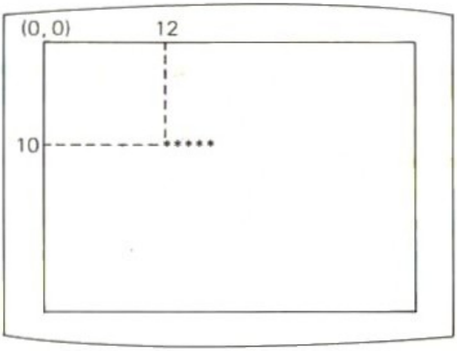
LOCATE [X-coordinate], [Y-coordinate], [cursor switch]

X-coordinate	Cond.	Numeric constants, variables, array variables, their expressions from 0 to 39.
	Omit	0
Y-coordinate	Cond.	Numeric constants, variables, array variables, their expressions from 0 to 24.
	Omit	0
Cursor switch	Cond.	0 . . . Cursor is not displayed. 1 . . . Cursor is displayed.
	Omit	1

□ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
10 CLS
20 LOCATE 12,10
30 PRINT "*****"
```



Function LOG (natural logarithm)

Gives the value of a natural logarithm (Log e).

□ **FORMAT**

LOG(X)

X	Cond.	Numeric constants, variables, array variables, their expressions larger than 0.
Given value:		Numeric type.

☐ FUNCTION AND UTILIZATION

The LOG function gives the value of a natural logarithm in which the base is e (2.7182818284588).

- The value of a logarithm $\mathrm{Log}_a b$ ($b > 0$), in which a is the base that is a positive numeral ($a \neq 1$), can be obtained by LOG(b)/LOG(a).

EXECUTION EXAMPLE

```
PRINT LOG(10)
 2,302585092994
```

Function LPOS (line printer position)

Gives the print head location in the printer buffer.

☐ FORMAT

LPOS(X)

X	Cond.	An arbitrary numeral (dummy argument).
Given value:		Integer type.

☐ FUNCTION AND UTILIZATION

Gives the location of a character currently being printed out to the printer in the line printer buffer memory. (Start=0).

L

LPRINT (line print)

Outputs the value of an expression to the printer.

☐ FORMAT

LPRINT [expression] [separater] [expression] [separater] [expression]· . . . . .

Expression	Cond.	Numeric and string constants, variables, array variables, their expressions.
	Omit	Line feeds
Separater	Cond.	, or ;

☐ FUNCTION AND UTILIZATION

An LPRINT statement outputs data to a printer while a PRINT statement outputs data to the screen. See PRINT for details.

LPRINT USING (line print using)

Outputs data to a printer in a specified format.

☐ FORMAT

LPRINT USING format symbol; expression [expression] . . . . .

Expression	Cond.	String and numeric constants, variables, array variables, their expressions.

☐ FUNCTION AND UTILIZATION

LPRINT USING outputs data to a printer in a specified format while PRINT USING outputs data to the screen in a specified format. See PRINT USING for details such as those for format symbols.

MAXFILES (maxfiles)

Declares the number of files that can be simultaneously opened in one program.

☐ FORMAT

MAXFILES = expression

Expression	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 15.

☐ FUNCTION AND UTILIZATION

Declares the number of files that can be simultaneously opened in one program. Opening files simultaneously means to open a file and open another file before closing the former.

EXECUTION EXAMPLE

```
10 MAXFILES=3
20 OPEN "GRP:"FOR OUTPUT AS #1
30 OPEN "CRT:"FOR OUTPUT AS #2
40 OPEN "LPT:"FOR OUTPUT AS #3
          .
          .
          .
1000 CLOSE
```

Since 3 was selected as the number of files that can be opened in line 10, 3 files can be opened in line 20 and after.
When the number of files is not specified by a MAXFILES = statement, only one file can be opened at one time.
- If a large value is unnecessarily declared, the user area becomes smaller.

MERGE (merge)

Loads a program saved by an ASCII format, and merges it with a program in memory.

☐ FORMAT

MERGE "device name [file name]"

Device name	Cond.	CAS: . . . Cassette tape.
File name	Cond.	String within 6 characters. If 7 or more characters are specified, the 7th character and after are ignored.
	Omit	Merges the first file found.

M

☐ FUNCTION AND UTILIZATION

Only CAS: can be specified as a device name. Loads a program saved on cassette tape in an ASCII format by a SAVE statement. The existing program in memory, maintained as it is, is merged with the program loaded by a MERGE statement.

- If the line numbers of the program loaded by a MERGE statement are the same as that of an existing program in memory, the line numbers of the program newly loaded by a MERGE statement are maintained.

EXECUTION EXAMPLE

```
MERGE "CAS:PROG3"
```

Function MID$ (middle dollar)

Fetches and gives a part of character data.

☐ FORMAT

MID$(X$, M [, N]

X$	Cond.	String type constants, variables, array variables, their expressions.
M	Cond.	Numeric type constants variables, array variables, their expressions from 1 to 255.
N	Cond.	Numeric constants, variables, array variables, their expressions from 1 to 255.
	Omit	Gives all characters after the Mth character.
Given value:		String type.

M

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT MID$("JAPANUKFRANCE",6,2)
UK
```

```
PRINT MID$("JAPANUKFRANCE",6,2.6)
UK
```

If N is not an integer value, figures below the decimal point are omitted.

```
PRINT MID$("JAPANUK",6,4)
UK
```

If N characters do not exist after the Mth character, all characters after the Mth character are given.

```
PRINT MID$("JAPANUK",12,5)

PRINT MID$("JAPANUK",6,0)
```

When the value of M is larger than the length of X$ or when N is 0, a null string is given.

MID$ = Y$ (middle dollar)

Replaces a part of a character string with another character string.

☐ FORMAT

MID$(X$, M[, N]) = Y$

X$, Y$	Cond.	String type constants, variables, array variables, their expressions.
M	Cond.	Numeric type constants, variables, array variables, their expressions from 1 to 255.
N	Cond.	Numeric type constants, variables, array variables, their expressions from 1 to 255.
	Omit	Mth character and after in X$ are replaced by Y$.

☐ FUNCTION AND UTILIZATION

Replaces the Mth character and after from the left in the X$ character string with the characters from the beginning to the Nth character in Y$. However, the length of X$ is not changed after execution.

EXECUTION EXAMPLE

```
10 X$="ABCDEFG"
20 Y$="QRSTUVWXYZ"
30 MID$(X$,4,2)=Y$
40 PRINT X$
RUN
ABCQRFG
```

MOTOR (motor)

Turns the motor of the cassette tape recorder on and off.

☐ FORMAT

MOTOR $\left[\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}\right]$

☐ FUNCTION AND UTILIZATION

Connect the computer TAPE terminal to the remote control terminal of a cassette tape recorder and place the recorder in a playback or record mode. Tape operation starts with MOTOR ON and stops with MOTOR OFF.

When only MOTOR is executed, if it is ON, it is switched to OFF, and if it is OFF, it is switched to ON.

NEW (new)

Erases a BASIC program in memory and clears variables.

☐ **FORMAT**

NEW

☐ **FUNCTION AND UTILIZATION**

NEW is executed before entering a new program to erase all previous programs and enter a command wait state.

- When a machine language program exists in memory, it is maintained even if NEW is executed.

Function OCT$ (octonary dollar)

Gives an octal expression of numeric data as string type data.

☐ **FORMAT**

OCT$(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 65535. If it is a negative numeral, it is the same as a value in which the value is added to 65536.
Given value:		String type.

☐ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT OCT$(100)
144

PRINT OCT$(65536-32768)
100000
```

N
O

ON ERROR GOTO (on error go to)

When an error occurs, execution is transferred to a specified line number.

☐ **FORMAT**

ON ERROR GOTO line number

Line number	Cond.	Integers from 0 to 65529.

☐ **FUNCTION AND UTILIZATION**

Used to prevent an execution interruption caused by an error that occured during program execution. When an error occurs after ON ERROR GOTO is declared, execution is transferred to a specified line number. (Also, when an error results from a direct command, execution is transferred to a specified line number.)

EXECUTION EXAMPLE

```
10 ON ERROR GOTO 100
20 INPUT A
30 B=SQR(A)
40 PRINT "SQR(A)=";B
50 END
100 IF ERR=5 AND ERL=30 THEN PRIN
T "Input a positive number."
110 RESUME 20
```

END statement that distinguishes a main routine from the error processing routine.

Error processing routine.

To invalidate an ON ERROR GOTO statement
Execute ON ERROR GOTO 0.

ON-GOSUB (on-go to subroutine)

Branches program execution to subroutines that start with specified line numbers depending on the value of the expression.

□ FORMAT

ON expression GOSUB line number [, line number] . . .

Expression	Cond.	Numeric type variables, array variables, their expressions from 0 to 255.
Line number	Cond.	Integers from 0 to 65529.

□ FUNCTION AND UTILIZATION

```
100 ON X GOSUB 500,600,700
```

In this program, if the value of X is 1, execution branches to a subroutine from line number 500, and if the value of X is 2, execution branches to a subroutine from line 600, and if it is 3, execution branches to a subroutine from line 700.
A return to the main program is accomplished by a RETURN statement.

Expression value and execution result
When the expression value is not an integer . . . Figures below the decimal point are omitted.
When the expression value is 0 or larger than the number of the line number specified by GOSUB . . . Transferred to a statement next to the ON – GOSUB statement.
When the expression value is negative or larger than 255 . . . An error occurs.

O

ON-GOTO (on-go to)

Branches program execution to line numbers that depend on the value of an expression.

☐ FORMAT

ON expression GOTO line number [, line number] . . .

Expression	Cond.	Numeric type variables, array variables, their expressions.
Line number	Cond.	Integers from 0 to 65529.

☐ FUNCTION AND UTILIZATION

```
100 ON X GOTO 120,130,180
```

In this program, if the value of X is 1, it branches to line 120, if it is 2, it branches to line 130, and if it is 3, it branches to line 180.

Expression value and execution result

When the expression value is not an integer . . . Figures below the decimal point are omitted.
When expression value is 0 or larger than the number of line numbers specified by GOTO . . . Transferred to a statement next to the ON – GOTO statement.
When the expression value is negative or larger than 255 . . . An error occurs.

O

ON INTERVAL GOSUB (on interval go to subroutine)

Declares a subroutine to which program branches when an interrupt is caused by a built-in timer.

☐ FORMAT

ON INTERVAL = Interval time GOSUB line number

Interval time	Cond.	Numeric type constants, variables, array variables, their expressions from –32768 to 65535 and other than 0.
Line number	Cond.	Integers from 0 to 65529.

☐ FUNCTION AND UTILIZATION

A statement that declares a subroutine starting line number to which program branches when an interrupt is caused by a built-in timer with a certain interval. The interrupt spacing is about (interval time x 1/50) second. In other words, when the interval time is specified as 50, an interrupt occurs approximately every (See page 50 for Interrupts).

EXECUTION EXAMPLE

```
10 ON INTERVAL=50 GOSUB 100
20 INTERVAL ON
30 SCREEN 2,1
40 SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$
(&H66)+CHR$(&HDB)+CHR$(&HE7)+CHR$(&H7E)+
CHR$(&H24)+CHR$(&H42)
50 GOTO 50
100 X=INT(RND(1)*256):Y=INT(RND(1)*192)
110 C=INT(RND(1)*14)+2
120 PUT SPRITE 1,(X,Y),C,1
130 RETURN 50
```

In this program, an interrupt occurs with about 1 second spacing provided by lines 10 and 20, and each time interrupt occurs, the execution is transferred to a subroutine from line 100. After a UFO shaped sprite pattern is displayed by this subroutine, a return to line 50 occúrs caused by RETURN 50.

- When the interval time is set to a negative numeral, it is equal to a numeral in which the specified interval time is added to 65536.

ON KEY GOSUB
(on key go to subroutine)

Declares a subroutine to which program branches when an interrupt is applied by a function key.

O

☐ FORMAT

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

Line number [Cond.] Integers from 0 to 65529.

☐ FUNCTION AND UTILIZATION

A statement that declares the starting line number of a subroutine to which program branches when an interrupt is applied by a function key. Up to 5 line numbers can be specified after GOSUB by punctuating them to sequentially correspond to [F1], [F2], etc.

EXECUTION EXAMPLE

```
10 ON KEY GOSUB 1000,2000
20 KEY(1) ON:KEY(3) ON
```

When [F1] is pressed, execution is transferred to a subroutine from line 1000, and when [F3] is pressed, it is transferred to the subroutine from line 2000 based on the above two lines of the program.

A return from the subroutine is made by a RETURN statement (See page 50 for Interrupts).

ON SPRITE GOSUB (on sprite go to subroutine)

Declares a subroutine to which program branches when an interrupt occurs due to a sprite overlap.

□ FORMAT

ON SPRITE GOSUB line number

Line number [Cond.] Integers from 0 to 65529.

□ FUNCTION AND UTILIZATION

A statement that declares the starting line number of a subroutine to which program branches when an interrupt occurs due to an overlap of sprite patterns.

EXECUTION EXAMPLE

```
10 ON SPRITE GOSUB 1000
20 SPRITE ON
```

When a sprite overlap occurs, execution is transferred to a subroutine from line 1000 based on the above two lines. A return is made from a subroutine by a RETURN statement.

ON STOP GOSUB (on stop go to subroutine)

Declares a subroutine to which program branches when a [CTRL] + [STOP] key interrupt occurs.

□ FORMAT

ON STOP GOSUB line number

Line number [Cond.] Integers from 0 to 65529.

O

□ FUNCTION AND UTILIZATION

A statement that declares the starting line number of a subroutine to which program branches when a [CTRL] + [STOP] key interrupt occurs.

EXECUTION EXAMPLE

```
10 ON STOP GOSUB 1000
20 STOP ON
```

Execution is transferred to a subroutine from line 1000 by simultaneously pressing [CTRL] and [STOP] based on the above two lines. A return from the subroutine is made by a RETURN statement. (See page 50 Interrupts.)

Precautions

It is necessary for a program to be terminated somehow when a subroutine is executed. The only way to terminate the following program is to press the [RESET] button.

```
10 ON STOP GOSUB 100
20 STOP ON
30 PRINT "MAIN ROUTINE"
40 GOTO 40
100 PRINT "CTRL+STOP EXECUTED"
110 RETURN 30
```

ON STRIG GOSUB (on stick trigger go to subroutine)

Declares a subroutine to which program branches when an interrupt is caused by the space bar or the trigger button of a joy stick.

□ FORMAT

ON STRIG GOSUB line number [, line number] . . .

Line number [Cond.] Integers from 0 to 65529.

□ FUNCTION AND UTILIZATION

A statement that declares the starting line number of a subroutine to which program branches when an interrupt occurs by the pressing of the space bar or joy stick trigger button. Up to five line numbers can be specified after GOSUB by punctuating them with commas.

On STRIG GOSUB line No. 1 , line No. 2 , line No. 3 , line No. 4 , line No. 5 .

Line No. 1 . . . . Branches when the space bar is pressed.
Line No. 2 . . . . Joy stick 1, Trigger button 1.
Line No. 3 . . . . Joy stick 2, Trigger button 1.
Line No. 4 . . . . Joy stick 1, Trigger button 2.
Line No. 5 . . . . Joy stick 2, Trigger button 2.

EXECUTION EXAMPLE

```
10 ON STRIG GOSUB 1000,2000,3000
20 STRIG(0) ON:STRIG(1) ON:STRIG(2) ON
```

When the space bar is pressed, execution is transferred to a subroutine from line 1000, and when trigger button 1 of joystick 1 is pressed, execution is transferred to a subroutine from line 2000. Also, when trigger button 1 of joystick 2 is pressed, execution is transferred to a subroutine from line 3000.
Return from a subroutine is accomplished with a RETURN statement. (See page 50 for Interrupts.)

OPEN (open)

Opens a file and specifies a mode

O

□ FORMAT

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

Device name	[Cond.]	CAS: . . . . Cassette tape CRT: . . . . Text mode screen GRP: . . . . Graphic mode screen LPT: . . . . Printer
File name	[Cond.]	String within 6 characters. If 7 or more characters are specified, the 7th character and after are ignored.
	[Omit]	Null-string
Mode	[Cond.]	OUTPUT . . . Write. INPUT . . . Read.
File number	[Cond.]	1 ≦ file number ≦ numeral specified by MAXFILES = statement

□ FUNCTION AND UTILIZATION

An OPEN statement opens a file with a specified file number to perform file I/O for a specified device. Since CRT:, GRP:, and LPT: of the devices that can be specified are dedicated to write-in, only OUTPUT can be specified as a mode. On the other hand, since write-in and read-out can be performed with CAS:, OUTPUT and INPUT can be specified.

- When write-in is performed with a file name, read-out can be performed by specifying the same file name.
- The file number should be equal to or less than the numeral that indicates the maximum number of files which can be opened, as specified by MAXFILES = statement.

EXECUTION EXAMPLE

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PSET (120,90)
40 PRINT #1,"ABC"
50 GOTO 50
```

This is a program that outputs characters on the screen in the graphic mode (SCREEN 2).
(See page 42 for File Processing)

OUT (out)

Outputs 1 byte data to a specified I/O port.

☐ FORMAT

OUT port number, expression

Port number, expression	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

This is a command that outputs data directly to an I/O port. See page 164 for I/O port assignments.

Function PAD (pad)

Provides the status of the touch pad.

☐ FORMAT

PAD(N)

N	Cond.	Integers from 0 to 7.
Given value:		Numeric type.

☐ FUNCTION AND UTILIZATION

Provides various data from a touch pad by an N value. When N is 0, 1, 2, or 3, the status of the touch pad connected to controller terminal A is provided. When is 4, 5, 6, or 7, the status of the touch pad connected to controller terminal B is given.

Value of N	Semantics for a given value
0 or 4	0: Not touched –1: Is touched
1 or 5	X coordinate of a touched location.
2 or 6	Y coordinate of a touched location.
3 or 7	0: Switch is not pressed. –1: Switch is pressed.

PAINT (paint)

Colors an area surrounded by a border line.

☐ FORMAT

PAINT[STEP] (X-coordinate, Y-coordinate), [display color], [border line color]

X-coordinate	Cond.	Numeric type constant, variables, array variables, their expressions from 0 to 255.
Y-coordinate	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 191.
Display color, border line color	Cond.	Integers from 0 to 15.
	Omit	Current foreground color.

☐ FUNCTION AND UTILIZATION

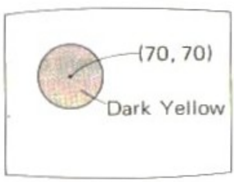
Colors an area with a display color inside a border line with a specified color including the location specified by X, Y coordinates.

- If the border line is not completely closed, the entire screen is colored.
- In the SCREEN 2 (high resolution) mode, if the display color is not the same as the border line color, the entire screen is colored.
- See page 29 for STEP specifications.

EXECUTION EXAMPLE

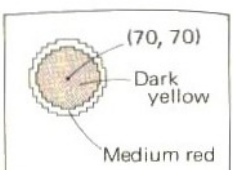
```
10 CLS
20 SCREEN 2
30 CIRCLE (70,70),40,10
40 PAINT (70,70),10,10
50 GOTO 50
```

In SCREEN 2, the same color must be specified for the display color and border line color.




```
10 CLS
20 SCREEN 3
30 CIRCLE (70,70),40,10
40 PAINT (70,70),8,10
50 GOTO 50
```

In SCREEN 3, different colors can be specified for the display color and border line color.



P

Function PDL (paddle)

Gives the value from a paddle.

☐ FORMAT

PDL(N)

PDL(N)

N	Cond.	Integers from 1 to 12.
Given value:		Numeric type from 0 to 255.

☐ FUNCTION AND UTILIZATION

Gives the value obtained from a paddle as numeric type data. When N is an odd number, data is provided from the paddle connected to controller terminal A, and when N is an even number, data is provided from the paddle connected to controller terminal B.

Function PEEK (peek)

Gives the content of a specified memory address.

☐ FORMAT

PEEK(address)

Address	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 65535. In the case of negative numerals, their value is the same as if it is added to 65536.
Given value:		Numeric type decimal format.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
M=PEEK(50000)
```
— Assigns the content of memory address 50000 to variable M.

P

PLAY (play)

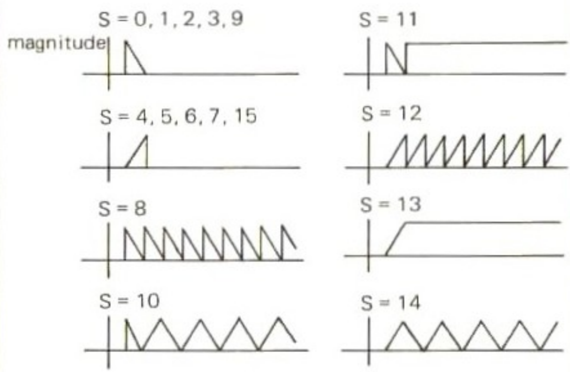
Generates a sound according to a subcommand specification.

☐ FORMAT

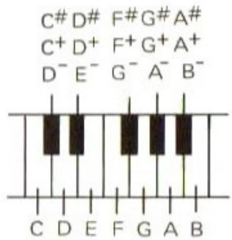
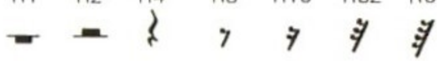
PLAY subcommand

Subcommand	Cond.	Character string (constant) inside " ", or a string type variable which is assigned a character string. Capitals or small characters.

Subcommands

Command	Condition	Semantics
Tn (tempo)	Integers of $32 \leqq n \leqq 255$	Specifies the speed of music. The value of n indicates the counting of a quarter note for one minute. The initial setting is T120.
On (octave)	Integers of $1 \leqq n \leqq 8$	Specifies one of 8 octaves. When O4 is specified, music within the range shown below is performed.  The octave becomes lower as the value of n becomes smaller and becomes higher as the value of n becomes larger. The initial value is O4.
Sn (shape)	$0 \leqq n \leqq 15$	Specifies the volume variation pattern from among the following patterns.  The initial setting is S1. The generation of many different sounds is determined by a combination of the S subcommand and the M sub-command.
Mn (modulation)	$1 \leqq n \leqq 65535$	Determines the cycle of the pattern specified by the S subcommand. The cycle becomes long as the value of n is increased. The initial setting is M255.

P

Ln (length)	$1 \leqq n \leqq 64$	Indicates the length of sound. L1 L2 L4 L8 L16 L32 L64  Initial setting is L4.
Nn (note)	$0 \leqq n \leqq 96$	Specifies a musical note. N36 is 04C  N0 is a rest. The chromatic scale increases as n is increased by 1.
A – G (An – Gn)	$1 \leqq n \leqq 64$	Specifies the musical note within a specified octave.  # (or +) and – are used for a semitone. The sound length can be specified by n. (C4 is the same as L4C.) When omitted, it is the length specified by Ln.
Rn (rest)	$1 \leqq n \leqq 64$	Specifies a rest. R1 R2 R4 R8 R16 R32 R64 
•		Express a dot. The length is extended to 1.5 times by placing it by one. C4. = ♩. R8. = 𝄾.
Vn (volume)	$0 \leqq n \leqq 15$	Specifies the volume. The volume increases as n becomes larger. The inital setting is V8.

☐ FUNCTION AND UTILIZATION

```
PLAY "T80O3L4CDEFG2.RABO4CDC2."
```

Based on the above statement, the sound is played according to the following notes.

To express a subcommand with a variable

```
M$="T80O3L4CDEFG2.RABO4CDC2."
PLAY M$
```

A subcommand is assigned to a string type variable, M$, once, then M$ is specified in a PLAY statement as a subcommand.

To express a part of a subcommand with a variable (X variable;)

```
10 M$="CDEFG2.R"
20 PLAY "O4L4XM$;GAGAG2.R"
30 PLAY "XM$;ABO5CDC2."
```

When a subcommand assigned to a string type variable is used in " " of a play statement, add X before and ; after. In the example above, a subcommand assigned to M$ is used in two PLAY statements.

To express n in a subcommand with a variable (=variable;)

n which is specified in each subcommand can be a constant or a variable in a PLAY statement. When expressed as a variable, = is added before and ; after.

P

```
10 FOR I=1 TO 8
20 PLAY "O=I;CEG"
30 NEXT I
```

This program plays 8 octave music from PLAY "O1CEG" to PLAY "O8CEG".

Performance of chords

Up to 3 commands can be simultaneously played such as PLAY A$, B$, C$

```
10 A$="O4CDO3BO4E2R4"
20 B$="O4EFDG2R4"
30 C$="O4GAGO5C2R4"
40 PLAY A$,B$,C$
```

This program plays the following notes.

Function PLAY (play)

Checks if music is being played or not.

☐ FORMAT

PLAY(N)

N — Cond. Integers from 0 to 3.

Given value: Numeric type.

☐ FUNCTION AND UTILIZATION

Three different sounds can be simultaneously played in a PLAY statement.
In the case of PLAY A$, B$, C$;
the sound of subcommand A$ is output from Channel 1, the sound of B$ is output from Channel 2, and the sound of C$ is output from Channel 3.

The PLAY function checks if data is in the music data buffer of Channel 1 when N = 1, the same for Channel 2 when N = 2, and the same for Channel 3 when N = 3. When data is in the buffer, –1 is given, and when there is no data, 0 is given. When N = 0, the OR (logical sum) of the buffer status (0 or 1) of all channels is given. In other words, if one of them is –1, –1 is given.

Function POINT (point)

Gives the color code of a point at a specified location in the graphics screen.

☐ FORMAT

POINT(X, Y)

X, Y — Cond. Numeric type constants, variables, array variables, their expressions from –32767 to 32767.

Given value: Numeric type (–1 is given when a specified location is outside the display area.)

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 SCREEN 3
20 FOR I=1 TO 250
30 X=INT(RND(1)*255)
40 Y=INT(RND(1)*191)
50 PSET (X,Y),1
60 NEXT I
70 FOR Y=0 TO 191 STEP 4
80 FOR X=0 TO 255 STEP 4
90 C=POINT(X,Y)
100 IF C=4 THEN PSET (X,Y),15
110 NEXT X,Y
120 GOTO 120
```

The color code for the location (X, Y) is assigned to variable C in line 90, and changed into white in line 100 if C is 4 (dark blue).

P

POKE (poke)

Writes data to a specified memory address.

☐ FORMAT

POKE address, expression

Address	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 65535.
Expression	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
POKE 50000,255
```
Writes 255 as data to memory 50000.

```
POKE &HD000,&HA8
```
Writes $A8_H$ as data to memory address $D000_H$.

Function POS (position)

Gives the X-coordinate of the cursor position.

☐ FORMAT

POS(X)

X	Cond.	An arbitray numeric value (dummy argument)
Given value:		Integer type

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 INPUT A$
20 PRINT A$;.X=POS(X)
30 IF X>=5 THEN CLS
40 PRINT:GOTO 10
```

The value of the cursor X-coordinate is given to the variable X based on line 20, X=POS(X). As a result, the screen is cleared by inputting a string with 5 characters or more than 5 characters for A$.

P

PRESET (point reset)

Marks or erases a dot on the screen in the graphic mode.

☐ FORMAT

PRESET[STEP] (X-coordinate, Y-coordinate) [, color]

X, Y-coordinate	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 32767.
Color	Cond.	Integers from 0 to 15.
	Omit	Current background color.

☐ FUNCTION AND UTILIZATION

When executed with color omitted, a dot is marked with the same color as the background color. As a result, if something is drawn at a specified location with a color other than the background color, it looks as if a point at the same location was only erased.

- When a color is specified, it functions exactly the same as when a color is specified by PSET.
- See page 29 for STEP specifications.
- See PSET for a program example.

PRINT (print)

Displays numeric data or character data on the text screen.

☐ FORMAT

PRINT expression [separator] [expression] [separator] [expression] . . .

Expression	Cond.	Numeric type or string type constants, variables, array variables, their expressions.
Separator	Cond.	Comma (,) or semicolon (;).

P

☐ FUNCTION AND UTILIZATION

Expression (data) writing method

Numeric type constants, numeric type and string type variables are written as they are, and string type constants are written inside quotation marks (" ").

Separator function

When data is punctuated with a comma (,), spaces by a 14 digit tab function is inserted between the data, and when it is punctuated with a semicolon (;), it is followed by the next data.

If a separator is not written at the end, line feed is performed after the data display. If a separator is written at the end, data of the next PRINT statement continues on the same line without a line feed.

Numeric data and signs

In regard to signs that indicate positive or negative, "+" is omitted while "−" is displayed as it is. (If a ";" separator is used when positive numeric data is displayed, two spaces are inserted between data to provide space for a sign.

Omitted format

The same result can be obtained by inputting "?" instead of PRINT.

EXECUTION EXAMPLE

```
10 A$="ABC":B$="DEF"
20 PRINT A$;B$
30 PRINT A$,B$
40 PRINT
50 PRINT "MSX"
60 PRINT +50,-50
70 ?"PERSONAL COMPUTER"
RUN
ABCDEF―――――――――――――――――――――――― Result of line 20.
ABC             DEF―――――――― Result of line 30.
    ―――――――――――――――――――――――――――― Result of line 40.
MSX――――――――――――――――――――――――――― Result of line 50.
 50             -50―――――――― Result of line 60.
PERSONAL COMPUTER――――――――― Result of line 70.
```

P

PRINT USING (print using)

Outputs data to the screen in a specified format.

☐ FORMAT

PRINT USING format symbol; expression [expression] . . .

Expression [Cond.] String type and numeric type constants, variables, array variables, their expressions.

☐ FUNCTION AND UTILIZATION

The value of an expression is displayed in a format specified by a format symbol.

Format symbols for character type data

Symbol	Expression format and Execution example

"!"	Outputs the first 1 character.
	`PRINT USING "!";"United","Nation"`
	`UN`

"\ \"
n spaces
	Outputs n + 2 characters. When data is smaller than n + 2 characters, inserts spaces for the residual characters.
	`PRINT USING "\   \";"ABCDEF","GHI"`
	`,"JKLMN"`
	`ABCDGHI JKLM`

"&"	Outputs all character strings.
	`10 A$="North":B$="South"`
	`20 PRINT USING "& Pole ";A$,B$`
	`RUN`
	`North Pole South Pole`

Format symbols for numeric type data

<table>
<tr><th>Symbol</th><th>Expression format and Execution example</th></tr>
<tr><td>"#"</td><td>Writes # by the number of numeral digits to be displayed. Decimal point is ".".

```
PRINT USING "POINT:###.#";123.4
POINT:123.4
```

• When the number of integer digits is less than the specified # number, data is displayed with right justification, and if it is more, "%" is added before the data.

```
10 PRINT USING "####";12
20 PRINT USING "####";12345
RUN
  12
%12345
```

• When the number of digits in a fraction of numeric data is smaller than the specified # number, "0" is added, and when it is larger, it is rounded to the nearest whole number.

```
10 PRINT USING "##.##";25.3
20 PRINT USING "##.##";25.345
RUN
25.30
25.35
```

The "+" sign of numeric data is ignored and the "−" sign is counted as one digit.

```
10 PRINT USING "###";+123
20 PRINT USING "###";-123
RUN
123
%-123
```
</td></tr>
<tr><td>"+"</td><td>"+" is added if it is a positive numeral, and "−" is added if it is a negative numeral before or after the numeric data.

```
10 PRINT USING "+####";123,-123
20 PRINT USING "####+";123,-123
RUN
 +123 -123
 123+ 123-
```
</td></tr>
<tr><td>"−"</td><td>"−" is added after negative numeric data.

```
PRINT USING "###-";123,-123
123 123-
```
</td></tr>
</table>

P

"**"	The space before numeric data is filled with "*". One "*' in the format expresses one digit.
	`10 PRINT USING "**######";123`
	`20 PRINT USING "**######";-123`
	`RUN`
	`*****123`
	`****-123`

"£"	Adds "£" before numeric data. One "£" in the format is couted as one digit.
	`10 PRINT USING "££###";1234`
	`20 PRINT USING "+££###";-1234`
	`RUN`
	`£1234`
	`-£1234`

"**£"	Adds "£" just before the numeric data, and space before that is filled with "*".
	`PRINT USING "**£###.##";12.34`
	`***£12.34`

","	When this is specified somewhere before the decimal point, it is displayed by the insertion of commas between each 3 digits to the left of the decimal point.
	`PRINT USING "#,######.##";12345.67`
	`   12,345.67`

"^^^^"	Displays numeric data by floating point type.
	"^^^^" corresponds to the digits for exponent part.
	`PRINT USING "##.##^^^^";234.56`
	` 2.35E+02`

P

PRINT# (print number)

Writes data to a file opened by an OPEN statement.

☐ FORMAT

PRINT # file number, expression

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement.
Expression	Cond.	String type and numeric type constants, variables, array variables, their expressions.

☐ FUNCTION AND UTILIZATION

Outputs data to a file opened by an OPEN statement.

EXECUTION EXAMPLE

```
10 OPEN "CAS:DATA" FOR OUTPUT AS #1 —— Opens a file to write-in.
20 FOR I=0 TO 4
30 READ A$
40 PRINT #1,A$;",";——Writes data to a file.
50 NEXT I
60 CLOSE #1
70 DATA TOKYO,LONDON,PARIS,PEKING
,NEW YORK
```

This is a program which sequentially writes data written in line 70 to cassette tape with a file name "DATA".
(See page 42 for File Processing.)

PRINT# USING (print number using)

Writes data to a file opened by an OPEN statement in a specified format.

P

☐ FORMAT

PRINT # file number USING format symbol; expression

File number	Cond.	1 ≦ file number ≦ numeral specified by MAXFILES= statement.
Expression	Cond.	String type and numeric type constants, variables, array variables, their expressions.

☐ FUNCTION AND UTILIZATION

This format can be specified when data is output to a file. See PRINT USING for a format symbol.

PSET (point set)

Marks a dot on a graphic mode screen.

☐ FORMAT

PSET [STEP] (X-coordinate, Y-coordinate) [, color]

X, Y coordinates	Cond.	Numeric type constants, variables, array variables, their expressions from −32768 to 32767.
Color	Cond.	Integers from 0 to 15.
	Omit	Current foreground color.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 SCREEN 2                      P-144
20 FOR X=0 TO 255
30 PSET(X+1,100)———————— Draws a dot.
40 PRESET (X,100)——————— Erases the dot drawn before.
50 NEXT X
```

See page 29 for STEP specifications.

PUT SPRITE (put sprite)

Displays a specified sprite pattern at an arbitray location on a specified sprite plane.

☐ FORMAT

PUT SPRITE sprite plane number [[STEP] (X-coordinate, Y-coordinate)], [color], [sprite number]

Sprite plane number	Cond.	Integers from 0 to 31.
X-coordinate	Cond.	Numeric type constants, variables, array variables, their expressions from −32 to 255.
Y-coordinate	Cond.	Numeric type constants, variables, array variables, their expressions from −32 to 191.
STEP (X-coordinate, Y-coordinate)		
	Omit	Previous location specified by the last graphic instruction.
Color	Cond.	Integers from 0 to 15.
	Omit	Current foreground color.
Sprite number	Cond.	For 8 x 8 dots, it is from 0 to 255. For 16 x 16 dots, it is from 0 to 63.
	Omit	Same as the sprite plane number.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 SCREEN 2
20 SPRITE$(1)=CHR$(&H18)+CHR$(&H3
C)+CHR$(&H66)+CHR$(&HDB)+CHR$(&HE
7)+CHR$(&H7E)+CHR$(&H24)+CHR$(&H42)
30 X=0:Y=0:DX=1:DY=1
40 PUT SPRITE 0,(X,Y),,1
50 X=X+DX:Y=Y+DY
60 IF X>2500R X<0 THEN DX=-DX
70 IF Y>190 OR Y<0 THEN DY=-DY
80 GOTO 40
```

A UFO shape is defined in line 20 as a sprite pattern assigned to sprite number 1. The sprite pattern is displayed on the screen by a PUT SPRITE statement in line 40. The sprite plane number is 0. Since the display color is omitted, it is the same as the foreground color that was set. The UFO pattern appears to fly around the screen because the X, Y values that specify the display location are changed.

READ (read)

Reads data specified in a data statement.

☐ FORMAT

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

Variable	Cond.	Numeric type or string type.

☐ FUNCTION AND UTILIZATION

Reads data in a sequence starting from the first data in the DATA statement that has the smallest number in a program, and assigns them sequentially to variables in the READ statement.

- When a plural number of numeric type or string type variables are arranged in one READ statement, they are punctuated with a comma (,).
- The variable type must be in accord with the corresponding data.

```
10 READ A,B,C,D$,E$
20 PRINT A,B,C,D$,E$
100 DATA 5,10,20,ABC,XYZ
```

- When a plural number of READ statements exist in a program, the 2nd READ statement starts reading from data that is next to data read by a previous READ statement.
- When a RESTORE statement is executed, the READ statement readout executed next returns to the smallest DATA statement after the line number specified by the RESTORE statement.

EXECUTION EXAMPLE

```
10 READ A,B,C
20 READ D$,E$
30 PRINT A;B;C;D$;E$
100 DATA 10,20,30,ABC,DEF
RUN
 10  20  30 ABCDEF
```

REM (remark)

Inserts a comment statement in a program.

□ FORMAT

REM comment statement

□ FUNCTION AND UTILIZATION

A REM statement is used to insert a comment statement so that a program list can be easily read.

EXECUTION EXAMPLE

```
10 REM MUSIC
20 PLAY "T60CEGEC1"
```

Although a REM statement is displayed when a program is listed, it is skipped during program execution.

```
10 'MUSIC
20 PLAY "T60CEGEC1"
```

A single quotation mark (') can be used instead of REM.

```
10 PRINT "MSX":REM output
20 PRINT "PERSONAL COMPUTER" 'Output
RUN
MSX
PERSONAL COMPUTER
```

Although a colon (:) is required when REM is added after another statement, it can be omitted by using " ' "

R

RENUM (renumber)

Renumbers the lines of a program.

□ FORMAT

RENUM [new starting line number], [old starting line number], [increment]

New starting line number	Cond.	Integers from 0 to 65529.
	Omit	10
Old starting line number	Cond.	Integers from 0 to 65529.
	Omit	Smallest line number before execution.
Increment	Cond.	Integers from 0 to 65529.
	Omit	10

□ FUNCTION AND UTILIZATION

Used to renumber lines after a program correction.

- The line number jumped to in a GOTO or GOSUB statement can be correctly renumbered by executing a RENUM statement. However, if the specified line number jumped to in a GOTO statement, etc. does not exist when RENUM is executed, the line number jumped to in a GOTO statement is not changed and an error occurs.

EXECUTION EXAMPLE

```
RENUM ———————— Renumbers all lines from line 10 with an increment of 10.
RENUM 100,,100 ———— Renumbers all lines to the line numbers beginning
                    with line 100, having an increment of 100.
RENUM 100 ———————— Renumbers all lines to the line numbers beginning
                    with line 100, having an increment of 10.
RENUM 100,38,20 ——— Renumbers the line 38 and after to the line
                    numbers beginning with line 100, having an incre-
                    ment of 20.
LIST ———————————— Executes LIST.
15 FOR I=0 TO 10
20 A=A+1
23 PRINT A
35 NEXT I
Ok
RENUM ——————————— Executes RENUM.
Ok
LIST ———————————— Executes LIST again.
10 FOR I=0 TO 10
20 A=A+1
30 PRINT A
40 NEXT I
```

R

RESTORE (restore)

Specifies a DATA statement read by a READ statement.

☐ **FORMAT**

RESTORE [line number]

Line number	Cond.	Integers from 0 to 65529.
	Omit	DATA statement with the smallest line number.

☐ **FUNCTION AND UTILIZATION**

A RESTORE statement is used when the same data has to be read a plural number of times. When a RESTORE statement is executed, the next READ statement starts reading data from the DATA statement with the smallest line number after the line number specified by the RESTORE statement.

EXECUTION EXAMPLE

```
10 READ A,B,C
20 READ D,E,F
30 RESTORE 110
40 READ G,H,I
50 PRINT A;B;C;D;E;F;G;H;I
100 DATA 10,20,30
110 DATA 40,50,60
run
 10  20  30  40  50  60  40  50  60
```

RESUME (resume)

Returns execution to a main program after execution of the error processing routine.

R

☐ **FORMAT**

RESUME $\left[\left\{\begin{matrix}\textbf{0}\\ \textbf{line number}\\ \textbf{NEXT}\end{matrix}\right\}\right]$

Line number	Cond.	Integers from 0 to 65529.
	Omit	Line where an error occured.

☐ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

`RESUME 0` or `RESUME` ——— Returns to s statement where an error occurred.

`RESUME 100` ——— Returns to line 100.

(See the program example in ON ERROR GOTO.)

Function RIGHT$ (right dollar)

Gives an arbitrary number of characters taken from the right of string data as string data.

☐ FORMAT

RIGHT$(X$, N)

X$	Cond.	String type constants, variables, array variables, their expressions.
N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
Given value:		String type.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT RIGHT$("I LOVE TOKYO",5)
TOKYO
```

```
PRINT RIGHT$("I LOVE TOKYO",5.3)
TOKYO
```

When N is not an integer value, figures below the decimal point are omitted.

```
PRINT RIGHT$("I LOVE TOKYO",0)
Ok
```

When N is 0, a null string is given.

Function RND (random)

Gives a random positive number less than 1 (including 0).

R

☐ FORMAT

RND(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions.
Given value:		Numeric type.

☐ FUNCTION AND UTILIZATION

When X is larger than 0

Random numbers are always generated in the same sequence.

```
10 FOR N=1 TO 10
20 PRINT RND(1)
30 NEXT N
RUN
```

```
.59521943994623
.10658628050158
.76597651772823
.57756392935958
.73474759503023
.18426812909758
.37075377905223
.94954151651558
.63799556899423
.47041117641358
```

When X is negative
Generates a series that corresponds to the value of X, and after that generates random numbers with this series.

```
10 PRINT RND(-1)
20 FOR N=1 TO 10
30 PRINT RND(N)
40 NEXT N
RUN
 .04389820420821
 .0962486816692
 .21069655852301
 .3265173630504
 .47775124336581
 .3409147084636
 .12971184081661
 .0977770174288
 .35157860175541
 .835389696666
 .63902641386221
```

R

When X is 0
Gives the same value as that generated before.

```
10 PRINT RND(1)
20 PRINT RND(0)
30 PRINT RND(-1)
40 PRINT RND(0)
RUN
 .59521943994623
 .59521943994623
 .04389820420821
 .04389820420821
```

RUN (run)

Executes a program from a specified line.

□ FORMAT

RUN [line number]

Line number	Cond.	Integers from 0 to 65529.
	Omit	Executes from the starting line.

□ FUNCTION AND UTILIZATION

When RUN is executed, a program is executed after all variables are undefined (numeric variables are set to 0, and string variables are set to null strings). After program execution has been terminated, a command wait status occurs.

- Press STOP to temporarily stop program execution. Execution is resumed by pressing it again.
  Press CTRL and STOP to interrupt a program. In this case, it can be resumed by a CONT command.

SAVE (save)

Saves a BASIC program on a specified device.

□ FORMAT

SAVE "device name [file name]"

Device name	Cond.	CAS: . . . . Cassette tape CRT: . . . . Text mode screen GRP: . . . . Graphic mode screen LPT: . . . . Printer
File name	Cond.	String within 6 characters. If 7 or more characters are specified, the 7th character and after are ignored.
	Omit	Null string

□ FUNCTION AND UTILIZATION

When CAS: is specified as a device name, a BASIC program in memory is saved on cassette tape in an ASCII format.

EXECUTION EXAMPLE

```
SAVE"CAS:PROG2"
```

- A program to be merged with a program in memory by a MERGE statement must be saved with an ASCII format.

R
S

SCREEN (screen)

Sets the screen display mode, sprite size, key sound or no key sound, and the cassette interface baud rate, and also selects the type of printer.

□ FORMAT

SCREEN [mode], [sprite size], [key click switch], [baud rate], [printer type]

Mode	Cond.	0, 1, 2 or 3.
	Omit	Current mode.
Sprite size	Cond.	0, 1, 2 or 3.
	Omit	Current size.
Key click switch	Cond.	0 or integers from 1 to 255.
	Omit	Current state.
Baud rate	Cond.	1 or 2.
	Omit	Current baud rate.
Printer type	Cond.	Integers from 0 or 1 to 255.
	Omit	Current printer type.

Modes

Specified value	Mode
0	40 characters x 24 lines Text mode
1	32 characters x 24 lines Text mode
2	High resolution graphic mode
3	Multi-color mode

Sprite size

Specified value	Size
0	8 x 8 dot unmagnified
1	8 x 8 dot magnified
2	16 x 16 dot unmagnified
3	16 x 16 dot magnified

S

Key click switch

Specified value	Key depression sound
0	No
Other than 0. *	Yes

*Range from 1 to 255.

Baud rate

Specified value	Baud rate*
1	1200 baud
2	2400 baud

*Cassette interface baud rate.

Printer type

Specified value	Printer
0	MSX printer**
Other than 0*	Non MSX printer***

* Range from 1 to 255.
** A printer compatible with MSX personal computers with graphic characters.
*** For non MSX printers, graphic characters are converted to spaces.

Initial value specification and omission

When a specification is omitted, the presently selected mode is maintained. The initial state is as follows.

Mode	:	40 characters x 24 lines text mode
Sprite size	:	8 x 8 dot unmagnified
Key click switch	:	Key click sound
Baud rate	:	1200 baud
Printer type	:	MSX printer

EXECUTION EXAMPLE

```
10 SCREEN 0,,1 ——— 40 character x 24 line text mode, no key
                   click sound.  (WIDTH 37,24)
10 SCREEN ,,,2 ——— Baud rate is selected as 2400 baud.
10 SCREEN 2,3 ——— High resolution graphic mode, Sprite is 16
                  x 16 dot magnified.
10 SCREEN 2
20 FOR I=0 TO 255
30 PSET (I,100)
40 NEXT I
50 GOTO 50
```

When program execution has been terminated, the screen returns to the text mode (SCREEN 0 or 1). As a result, when the graphic mode is to be maintained, program execution is as shown in line 50 of the above program. Press [CTRL] and [STOP] at the same time to stop execution.

Function SGN (sign)

Gives 1 when numeric data is positive, 0 when it is 0, and −1 when it is negative.

S

□ FORMAT

SGN(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions.
Given value:		Integer type.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 INPUT A
20 IF SGN(A)=-1 THEN PRINT "Negative"
30 GOTO 10
```

"Negative" is displayed in line 20 only when the value assigned to A is negative.

Function SIN (sine)

Gives the sine value for numeric data.

☐ FORMAT

SIN(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions. (Unit: Radian)
Given value:		Floating point type constants from −1 to 1.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT SIN(3.14/3)
 .86575983949239
PRINT SIN(60*3.14/180)
 .86575983949239
```

- To give X in degree units, use the formula SIN(X*π/180).

SOUND (sound)

Generates sound effects by writing data directly to the PSG (Programmable Sound Generator) register.

☐ FORMAT

SOUND register number, expression

Register number	Cond.	Integers from 0 to 13.
Expression	Cond.	Constants, variables, array variables, their expressions within the determined range for each register.

PSG register functions and the write data range

Register No.	Function	Data range
0	Channel A frequency	0 – 255
1		0 – 15
2	Channel B frequency	0 – 255
3		0 – 15
4	Channel C frequency	0 – 255
5		0 – 15
6	Noise frequency	0 – 31
7	Selects a channel for tone and noise generation.	0 – 63
8	Channel A volume	0 – 15 Volume variation occurs when 16 is selected.
9	Channel B volume	
10	Channel C volume	
11	Volume variation pattern frequency	0 – 255
12		0 – 255
13	Volume variation pattern selection	0 – 14

□ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
10 SOUND 0,56  }—— Sets the Channel A frequency to 400 Hz.
20 SOUND 1,1   }
30 SOUND 7,62 ——— Selects a Channel A tone.
40 SOUND 8,8 ———— Selects the Channel A volume.
```

When this program is executed, a 400 Hz sound is continuously output.
Press [CTRL] + [STOP] to stop this.

Function SPACE$ (space dollar)

Gives an arbitrary number of spaces as string data.

□ **FORMAT**

SPACE$(N)

N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
Given value:		String type.

□ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT SPACE$(5);"ABC"
     ABC
```

5 spaces

● When N is not an integer value, figures below the decimal point are omitted.

Function SPC (space)

Outputs an arbitrary number of spaces.

□ **FORMAT**

SPC(N)

N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
Given value:		String type.

□ **FUNCTION AND UTILIZATION**

The SPC function can only be used in PRINT and LPRINT statements.

EXECUTION EXAMPLE

```
PRINT "ABC";SPACE$(10);"DEF"
ABC          DEF
```

10 spaces

● When N is not an integer value, the decimal point are omitted.

S

SPRITE ON
SPRITE OFF
SPRITE STOP

Validates, invalidates, or holds an interrupt caused by a sprite overlap.

☐ FORMAT

SPRITE ON — Interrupt valid
SPRITE OFF — Interrupt invalid
SPRITE STOP — Interrupt hold

☐ FUNCTION AND UTILIZATION

A command used to actually validate, (SPRITE ON), invalidate (SPRITE OFF), or hold (SPRITE STOP) an interrupt after an interrupt caused by sprite overlap is declared by an ON SPRITE GOSUB statement.

(See chapter 2.)

SPRITE$ (sprite dollar)

Defines sprite pattern data.

☐ FORMAT

SPRITE$(sprite number)

Sprite number	Cond.	When 8 x 8 dots – Integers from 0 to 255. When 16 x 16 dots – Integers from 0 to 63.

☐ FUNCTION AND UTILIZATION

When the sprite pattern is defined for the SPRITE$ variable, it is maintained as a specified sprite number pattern. See chapter 2.

S

Function SQR (square root)

Gives the square root value of numeric data.

☐ FORMAT

SQR(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions over 0.
Given value:		Numeric type.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT SQR(100)
 10
```

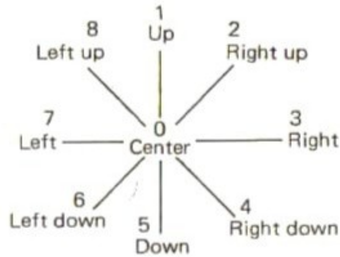
Function STICK (stick)

Gives the direction of cursor keys and joy sticks.

☐ FORMAT

STICK(N)

N	Cond.	0, 1 or 2.
Given value:		Integer type.



☐ FUNCTION AND UTILIZATION

Gives the direction of cursor keys when N=0, that for joystick 1 when N=1 and that for joystick 2 when N=2. The range of given values that indicate the direction is from 0 to 8. When no cursor key is pressed, or when joysticks are centered, 0 is given.

EXECUTION EXAMPLE

```
10 CLS
20 X=14
30 LOCATE X,10:PRINT " ";
40 D=STICK(0)
50 IF D=0 THEN LOCATE X,10:PRINT "*"
60 IF D=3 THEN X=X+1:IF X>28 THEN X=28
70 IF D=7 THEN X=X-1:IF X<0 THEN X=0
80 LOCATE X,10:PRINT "*";
90 GOTO 30
```

A program that moves "*" to the left and right on the screen by using the left and right cursor keys. The value given to variable D in line 40 depends on whether a cursor key is pressed or not. The X-coordinate, in which "*" is displayed by a given value, is modified in line 50, 60, and 70.

STOP (stop)

Interrupts program execution.

☐ FORMAT

STOP

☐ FUNCTION AND UTILIZATION

When a STOP statement is executed, program execution is interrupted.

- When a direct mode CONT statement is executed, execution restarts from the statement after the interrupted statement.

STOP ON (stop on)
STOP OFF (stop off)
STOP STOP (stop stop)

Validates, invalidates or holds an interrupt by the [CTRL] + [STOP] key.

☐ FORMAT

STOP ON — Interrupt valid
STOP OFF — Interrupt invalid
STOP STOP — Interrupt hold

☐ FUNCTION AND UTILIZATION

Commands that actually validate (STOP ON), invalidate (STOP OFF), or hold (STOP STOP) an interrupt after declaring an interrupt by [CTRL] + [STOP] using an ON STOP GOSUB statement. (See page 50 for Interrupts)

Function

STRIG (stick trigger)

Gives −1 when the space bar or a joystick trigger button is depressed, and 0 when they are not depressed.

☐ FORMAT

STRIG(N)

N	[Cond.]	Integers from 0 to 4.
Given value:		Integer type.

☐ FUNCTION AND UTILIZATION

Gives the space bar status when N=0, joystick 1 trigger button status when N=1, N=3, and the joystick 2 trigger button status when N=2, N=4. The given value is 0 when they are not depressed and −1 when they are depressed.

EXECUTION EXAMPLE

```
10 CLS
20 COLOR ,C,C
30 IF STRIG(0)=0 THEN GOTO 20
40 C=C+1:IF C>15 THEN C=0
50 GOTO 20
```

A program that changes the color of the screen every time the space bar is depressed.

STRIG ON (stick trigger on)
STRIG OFF (stick trigger off)
STRIG STOP (stick trigger stop)

Validates, invalidates or holds an interrupt by the sapce bar or a joystick trigger button.

☐ FORMAT

STRIG(n) ON — Interrupt valid
STRIG(n) OFF — Interrupt invalid
STRIG(n) STOP — Interrupt hold

n | Cond. | Numeric type constants, variables, array variables, their expressions from 0 to 4.

☐ FUNCTION AND UTILIZATION

Specifies the space bar, joystick 1 or 2 trigger buttons used for an interrupt by "n". The line number of the corresponding subroutine must be specified by an ON STRIG GOSUB statement.

Value of n	Specifies
0	Space bar
1	Joystick 1 trigger button 1
2	Joystick 2 trigger button 1
3	Joystick 1 trigger button 2
4	Joystick 2 trigger button 2

`STRIG(0) ON` — Validates a space bar interrupt.

`STRIG(1) OFF` — Invalidates a joystick 1 trigger button 1 interrupt.

`STRIG(2) STOP` - Holds a joystick 2 trigger button 1 interrupt.

(See page 50 for Interrupts.)

S

Function STR$ (convert to string)

Converts numeric type data to string type data.

☐ FORMAT

STR$(X)

X | Cond. | Numeric type constants, variables, array variables, their expressions.

Given value: String type.

☐ FUNCTION AND UTILIZATION

When numeric data is negative, the first character of the given string data is –. When it is 0 or positive, the first character of given string data is a space.

EXECUTION EXAMPLE

```
10 X=100:Y=200
20 X$=STR$(X):Y$=STR$(Y)
30 PRINT X+Y
40 PRINT X$+Y$
RUN
 300
 100 200
```

$\underbrace{\texttt{ 100}}_{X\$}\underbrace{\texttt{ 200}}_{Y\$}$

Function STRING$ (string dollar)

Gives the character of a given character code or the starting character of a given character string continuously by an arbitrary number as string data.

☐ FORMAT

STRING$(N, J)
STRING$(N, X$)

N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.
J	Cond.	An arbitrary character code (See the Character Code Table on page 165.)
X$	Cond.	String type constants, variables, array variables, their expressions.
Given value:		String type.

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT STRING$(10,70)
FFFFFFFFFF

PRINT STRING$(5,"ABC")
AAAAA
```

S

SWAP (swap)

Exchanges the value of two variables.

☐ FORMAT

SWAP variable, variable

variable	Cond.	Numeric type or string type variables, array variables. The two variables must have the same type.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
10 A=3:B=5
20 SWAP A,B
30 PRINT "A=";A
40 PRINT "B=";B
RUN
A= 5
B= 3
```

Function TAB (tab)

Moves the cursor from the beginning of a line to the right by the number of specified characters.

□ **FORMAT**

TAB(N)

N	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

□ **FUNCTION AND UTILIZATION**

The TAB function can only be used in PRINT and LPRINT statements. When N is 0, it is on the extreme left, and when it is a value in which 1 is subtracted from the number of characters on one line, it is on the extreme right.

EXECUTION EXAMPLE

```
PRINT TAB(5);"AAA"
     AAA
```

5 spaces

Function TAN (tangent)

Gives the tangent value for numeric data.

□ **FORMAT**

TAN(X)

X	Cond.	Numeric type constants, variables, array variables, their expressions. (Unit: radians)
Given value:		Floating point type constant.

T

□ **FUNCTION AND UTILIZATION**

EXECUTION EXAMPLE

```
PRINT TAN(3.14/3)
 1.72992922009

PRINT TAN(60*3.14/180)
 1.72992922009
```

- To give X in degree units, use the formula TAN ($X*\pi/180$).

TIME (time)

Holds the value of a built-in timer.

□ **FORMAT**

TIME
TIME=Expression

Expression	Cond.	Constants, variables, array variables, their expressions from 0 to 65535.

□ **FUNCTION AND UTILIZATION**

In regard to this variable, the value of a built-in timer is held during BASIC activation with the value advanced by 1 about every 1/50 second in a range from 0 to 65535. When 65535 is reached, it becomes 0 again.
The value of the variable can be rewritten with a LET statement. When the CPU is in an interrupt prohibition state (such as during cassette tape I/O), this timer is stopped. When the power is off, it does not operate.

EXECUTION EXAMPLE

```
10 CLS:TIME=0
20 LOCATE 12,8:PRINT INT(TIME/50)
30 GOTO 20
```

This program continuously displays the integer of the value, in which the value of TIME is divided by 50 after making the TIME variable value become 0 once. The numeral is advanced by 1 about every second.

T

TROFF (trace off)

Releases TRON to stop the display of executed line numbers.

□ **FORMAT**

TROFF

□ **FUNCTION AND UTILIZATION**

When a TROFF statement is executed in a direct or indirect mode during TRON statement execution, the display of a line number is released.

TRON (trace on)

Displays executed line numbers.

□ FORMAT

TRON

□ FUNCTION AND UTILIZATION

When a TRON statement is executed once by a direct or indirect mode, the line number executed after that is displayed on the text mode screen inside [ ]. It is used for program debug (correction), etc.

- When the screen in placed in a graphic mode by a SCREEN statement, the line number is not displayed.

EXECUTION EXAMPLE

```
10 TRON
20 FOR I=0 TO 3
30 A=I+1:PRINT A
40 NEXT I
50 TROFF
RUN
[20][30] 1
[40][30] 2
[40][30] 3
[40][30] 4
[40][50]
```

USR (user)

Gives the result obtained after the execution of a machine language routine that starts from an address defined by a DEFUSR statement.

□ FORMAT

USR [X] (I)

X	Cond.	Integers from 0 to 9.
	Omit	0
I	Cond.	Numeric type or string type constants, variables, array variables.
Given value:		Depends on the user function.

T
U

□ FUNCTION AND UTILIZATION

X is a user program number. The number specified by DEFUSR is used. I is a variable or constant that indicates the value to be transferred from BASIC to a subroutine.

EXECUTION EXAMPLE

```
DEFUSR0=&HE000
X=USR0(I)
```

Based on these statements, the subroutine after the address &HE000 is executed with the resultant value given to BASIC. (See page 56 for Machine language subroutines.)

Function VAL (value)

Gives string data as numeric data.

□ FORMAT

VAL(X$)

X$ — Cond. — String type constants, variables, array variables, their expressions that express numerals.

Given value: Numeric type.

□ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
PRINT VAL("5")
 5
PRINT VAL("   5")
 5
```

The space before string type data is ignored.

Function VARPTR (variable pointer)

Gives the starting address in memory where data assigned to a specific variable is stored.

□ FORMAT

VARPTR(variable)

variable — Cond. — Numeric type and string type variables, array variables.

□ FUNCTION AND UTILIZATION

Gives the decimal starting address in memory where a value assigned to a variable is stored. The given value ranges from −32768 to 32767. If it is negative, the actual address is one in which the value is added to 65536. The VARPTR function is used when an address in memory with data is transferred to a machine language subroutine for example.

EXECUTION EXAMPLE

```
10 MAXFILES=5
20 A=VARPTR(#1)
30 PRINT HEX$(A)
40 A%=15
50 X=VARPTR(A%)
60 N$=HEX$(X):PRINT N$
70 END
RUN
EE53
8060
```

This program checks the address in memory where the value assigned to a variable (A%) is stored, and displays it after converting it to hexadecimal.
Before calling the VARPTR, it is necessary to substitute numerical values for all the variables used in the program concerned.

V

VDP (video display processor)

Used to read and write the VDP register content.

□ FORMAT

VDP (register number)
VDP (register number) = expression

Register number	Cond.	Integers from 0 to 8.
Expression	Cond.	Constants, variables, array variables, their expressions from 0 to 255.

□ FUNCTION AND UTILIZATION

Used as a function to read the register content of the TMS9929A (VDP), the video display LSI of the MSX personal computer, or as a variable to write data directly to the register.

VDP registers

Followings are the bit assignment of the VDP registers.

Register 0

MSB 7	6	5	4	3	2	1	0 LSB
0	0	0	0	0	0	M3	EV

EV — External video signal (input if 1)
M3 — Mode selection bit 3

Register 1

MSB 7	6	5	4	3	2	1	0 LSB
4/16K	BLANK	IE	M1	M2	0	SIZE	MAG

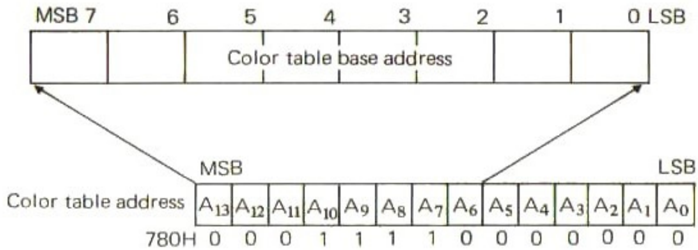
MAG — Sprite magnification (magnification if 1)
SIZE — Sprite size (8 x 8 if 0)
M1, M2 — Mode selection bit 1, 2
IE — Interrupt output specification
BLANK — Screen display ON/OFF (display if 1)
4/16K — 4K/16K byte memory selection (4K if 1)

Register 2

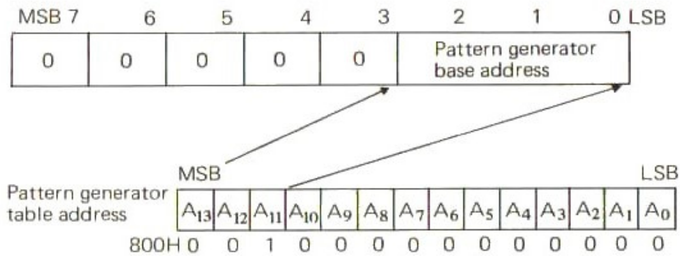
MSB 7	6	5	4	3	2	1	0 LSB
0	0	0	0	Pattern name table base address			

	MSB													LSB
Pattern name table base address	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
400H	0	0	0	1	0	0	0	0	0	0	0	0	0	0

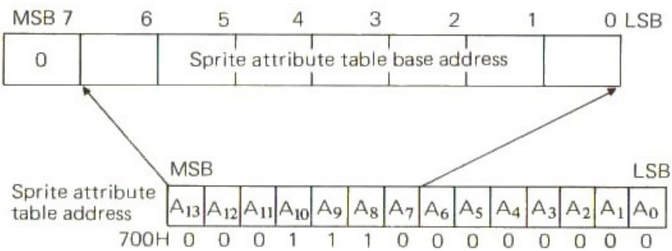
Register 3



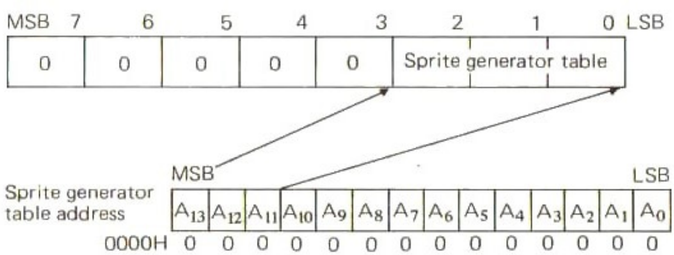
Register 4



Register 5



Register 6



V

Register 7

MSB 7	6	5	4	3	2	1	0 LSB
Text color "1"				Text color "0"/back drop color			

16 color codes from 0H to FH.

Register 8

MSB 7	6	5	4	3	2	1	0 LSB
F	5S	C	5th sprite No.				

F — Interrupt flag
5S — 5th sprite
C — Overlap flag
5th sprite No.

Register 8 is a read-out dedicated status register while the other registers are write-in dedicated.

Precautions

To accomplish screen operation with a VDP variable and by rewriting the VDP register value, adequate knowledge of the TMS9929A is necessary. If the VDP register is carelessly rewritten, the screen display is not correctly performed. Therefore, precautions shall be taken to avoid this.

Function VPEEK (video RAM peek)

Reads data in the video RAM.

□ FORMAT

VPEEK(address)

Address [Cond.] Integers from 0 to 16383.

□ FUNCTION AND UTILIZATION

Gives data written at a specified video RAM address.
Since the base address of each table can be found by the BASE function, use the BASE function to check the video RAM address when the VPEEK function is used.

V

VPOKE (video RAM poke)

Writes 1 byte data to video RAM.

☐ FORMAT

VPOKE address, expression

Address	Cond.	Integers from 0 to 16383.
Expression	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

Writes arbitrary data to a specified video RAM address. In regard to the video RAM address map, since the base address of each table can be found with the BASE function, check the video RAM address with the BASE function when a VPOKE statement is used.

WAIT (wait)

Waits until the I/O port input reaches a certain value.

☐ FORMAT

WAIT port number, expression 1 [, expression 2]

Port number expression 1, expression 2	Cond.	Numeric type constants, variables, array variables, their expressions from 0 to 255.

☐ FUNCTION AND UTILIZATION

When a WAIT statement is executed, data is input from a specified I/O port and XOR (exclusive OR) with the value of expression 2 is given, then AND (logical product) of the result and the value of expression 1 is given. If the value obtained as explained above is 0, data from the I/O port is continuously input and if it has a value other than 0, an advancement is made to the next line number. If expression 2 is omitted, its value is considered to be 0.

WIDTH (width)

Specifies the number of characters per line in the text mode.

☐ FORMAT

WIDTH(number of characters)

Number of characters	Cond.	Integers from 1 to 40 in the Screen 0 text mode. Integers from 1 to 32 in the SCREEN 1 text mode.

V
W

☐ FUNCTION AND UTILIZATION

EXECUTION EXAMPLE

```
SCREEN 0
WIDTH 40
```

In the SCREEN 0 text mode, 40 characters are set per line.

CHAPTER 4

SAMPLE PROGRAM

4

SAMPLE 1

A display color adjustment program is made using the SPRITE function and COLOR statements.

```
10 ' *** COLOR ***
20 COLOR 15,1,1:SCREEN2,2
30 OPEN "GRP:" FOR OUTPUT AS#1
40 FOR S=1 TO 2:A$=""
50 FOR P=1 TO 32:READ D$
60 A$=A$+CHR$(VAL("&H"+D$)):NEXT
70 SPRITE$(S)=A$:NEXT
80 FOR K=15 TO 2 STEP -1:Y=K*11+13
90 FOR X=10 TO 75+K*5 STEP 2
100 PUT SPRITE K,(X,Y),K,1
110 PUT SPRITE K+15,(X+8,Y-16),K,2
120 LINE(X-4,Y+3)-(X-2,Y+12),K,BF:NEXT
130 READ D$:PSET(X+30,Y-13),1:PRINT#1,D$
140 S=50+K*2:PLAY"V9N=S;32":NEXT
150 DRAW"BM15,15":PRINT#1,"Transparent"
160 DRAW"BM15,26":PRINT#1,"Black"
170 DRAW"BM27,0":PRINT#1,"Press RETURN K
ey."
180 IF INKEY$<>CHR$(13) THEN 180
190 COLOR 15,4,7:END
200 DATA 1,2,4,D,17,13,21,23,47,4C,F0
210 DATA C0,0,0,0,0,3F,7E,FC,F8,F0,E0
220 DATA C0,80,0,0,0,0,0,0,0,0
230 DATA 0,0,0,0,0,0,0,0,1,2,4,9,13,27
240 DATA 4F,9F,0,0,0,0,10,28,4C,9E,3F
250 DATA 7E,FC,F8,F0,E0,C0,80
260 DATA White,Gray,Magenta,Dark Green
270 DATA Light Yellow,Dark Yellow,Light
Red
280 DATA Medium Red,Sky Blue,Dark Red,Li
ght Blue
290 DATA Dark Blue,Light Green,Medium Gr
een
```

SAMPLE 2

Eight measures of Chopin's "Grande Valse Brillante" are played using the PLAY statement. Here the measure-by-measure music data are prepared in the DATA statements, and are read out successively by the READ statements for triple-chordal performance.

```
10 CLS:PRINT"WALTZ"
20 READ A$,B$,C$
30 IF A$="" THEN END
40 PLAY A$,B$,C$
50 GOTO 20
60 '
70 'DATA
80 '
85 DATA V13,V10,V10
90 DATA O4L4B-O5D8E-8F
100 DATA RRO5L4D
110 DATA RRR
120 DATA O4L4B-O5E-8F8G
130 DATA RRO5L4E-
140 DATA RRR
150 DATA O4L4B-O5F8G8A-
160 DATA RRO5L4F
170 DATA RRR
180 DATA O5L16B-4B-8R8B-R48B-R48
190 DATA O5L16G4G8R8GR48GR48
200 DATA O5L16D-4D-8R8D-R48D-R48
210 DATA O5L4B-O6C8O5B-8A-
220 DATA O5L2GR
230 DATA O5L2D-C4
240 DATA O5L4A-B-8A-8G
250 DATA O5L2C-O4B-4
260 DATA RRR
270 DATA O5L4GA-8G8F
280 DATA O4L2B-A-4
290 DATA RRR
300 DATA O5L4FG8F8E-
310 DATA O4L2A-G4
320 DATA RRR
330 DATA ""
340 DATA ""
350 DATA ""
```

CHAPTER 5

5

1. ERROR MESSAGES

When an error occurs, program execution is stopped, a command wait status occurs, and an error message is displayed. The cause of an error is concisely displayed as an error message. Error messages and actual examples of error causes are explained below. The numerals inside parentheses are error numbers.

Bad file name (56)

- File name is improper
- A device name that cannot be specified by an OPEN, SAVE or LOAD statement, was specified.

Bad file number (52)

- A file number was used that exceeds the range specified by a MAXFILES = statement.
- PRINT# statement execution was attempted with an unopened file number.

Can't CONTINUE (17)

- After an interruption, program was attemped to be restarted after modification.
- A program does not exist.
- A CONT statement was used in a program

Device I/O error (19)

- Load prevented due to cassette tape or tape recorder.
- Improper tape recorder level.
- Command interrupted before load completion.
- I/O unit error.

Direct statement in file (57)

- A statement in an ASCII program being loaded does not have a line number.
- An attempt was made to load a file other than that of a BASIC program (such as a data file).

Division by zero (11)

- Execution of division by zero was attempted.
- Execution of division by an undefined variable was attempted.

File already open (54)

- An attempt was made to reopen an opened file.

File not open (59)

- Execution of a PRINT# or INPUT# etc. statement was attempted by using a file number that was not opened by an OPEN statement.

Illegal direct (12)

- Execution of a statement that can only be used in a program, such as a DEF FN statement, was attempted by a direct command.

Illegal function call (5)

- A wrong value was used in a command.
- Value of a function is outside the tolerance range.

Input past end (55)

- Although all file data was read, read was attempted again.
- A file does not contain data.

Internal error (51)

- BASIC interpreter is abnormal.

Line buffer overflow (25)

- Input line buffer is full.

Internal error (51)

- BASIC interpreter is abnormal.

Line buffer overflow (25)

- Input line buffer is full.

Missing operand (24)

- No parameter exists after a command.
- Required parameters are incomplete.

NEXT without FOR (1)

- An executed NEXT statement has no corresponding FOR statement.
- Execution was transferred by a GOTO statement to somewhere inside a FOR – NEXT loop.

NO RESUME (21)

- An error processing routine has no RESUME statement. (An error processing routine must end with END, RESUME, or ON ERROR GOTO 0.)

Out of DATA (4)

- During READ statement execution, either no data or insufficient data exists.

Out of memory (7)

- Program too long.
- Too many variables used.
- Array too large.
- The multi-structure of a FOR – NEXT or GOSUB – RETURN statement is too long.

Out of string space (14)

- Character area is exceeded.
- The character area specified by a CLEAR statement is too small.

Overflow (6)

- Numeric type data or an arithmetic result exceeds the range that can be handled.
- An address parameter is outside a specified range.

RESUME without error (22)

- A RESUME statement has no corresponding ON ERROR statement.
- A transfer to an error processing routine by a GOTO statement.
- Since no END statement exists at the end of a main routine, an error processing routine is continuously executed.

RETURN without GOSUB (3)

- A RETURN statement has no corresponding GOSUB statement.
- Transfer to a subroutine by a GOTO statement.
- Since no END statement exists at the end of a main routine, a subroutine was continuously executed.

Redimensioned array (10)

- An attempt was made to define overlapping arrays with the same name.
- Array variables were used without being defined by a DIM statement, then they were defined.

String formula too complex (16)

- A one line character expression is too complicated.

String too long (15)

- A character variable was assigned a value that exceeded 255 characters.

Subscript out of range (9)

- A subscript was used that exceeded the size declared by a DIM statement.
- A subscript exceeding 11 was used for an array variable not declared by a DIM statement.

Syntax error (2)

- An input statement is not in accordance with MSX-BASIC grammar.

Type mismatch (13)

- The types of the left and right sides of LET, INPUT and READ statement are different.
- A logical operation was attempted to string type data.
- The type of data specified by a function is a mismatch.

Undefined line number (8)

- A non existing line number was specified in a GOTO, GOSUB, or RESUME statement.
- At RENUM statement execution, a non existing line number was specified with a GOTO statement etc.

Undefined user function (18)

- An attempt was made to use a user function not defined by a DEFFN statement.

Unprintable error (23, 26-49, 60-255)

- An error occurred that has no error number.
- An error occurred because the number of an unprintable error was specified in an ERROR statement.

Verify error (20)

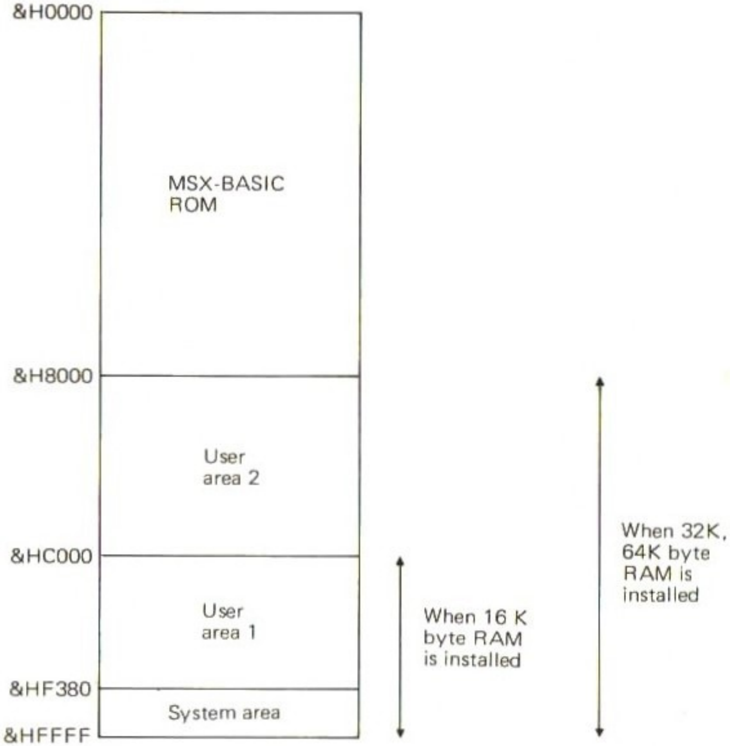
- The program on cassette tape is different from the program in memory.

CHAPTER 6

APPENDIXES

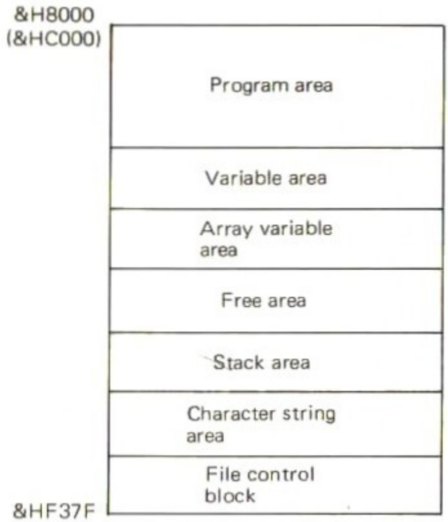
6

1. MEMORY MAP



See the Operating Instructions for the RAM capacity.

USER AREA CONFIGURATION



Program area
A program is stored with line numbers.

Variable area
Stores numeric type data and pointers for string type data.

Array variable area
Stores array variable data. Stores the pointer for the character string area if it is a string type.

Free area
Unused area. The size can be known with the FRE function.

Stack area
The stack area is used to save a return address.

Character string area
Stores a character string included in a string type variable or array variable.
The size can be specified with a CLEAR statement.

File control block
Used during file Input/Output.

2. I/O PORT ALLOCATION

Utilization	Port No.	Application
RS-232-C	&H80	Data read-out/write-in
	&H81	Mode set (during write-in) Status (during read-out)
Printer	&H90	Strobe (during write-in) Status (during read-out)
	&H91	Data write-in
VDP	&H98	Data read-out/write-in with video RAM.
	&H99	Command, address set (during write-in) Status (during read-in)
PSG	&HA0	Address latch (write-in)
	&HA1	Data write-in
	&HA2	Data write-out
PPI	&HA8	Data read-out/write-in for port A (Memory slot select) use.
	&HA9	Data read-out/write-in for port B (key borad scan) use.
	&HAA	Data read-out/write-in for port C (cassette).
	&HAB	Mode set (write-in)

- I/O addresses from &H00 to &H7F are not used. Addresses other than the above addresses of the address among &H80 to &HFF are reserved for system use.

3. CHARACTERS

CHARACTERS HANDLED BY MXS-BASIC

The characters shown in the following character code table can be displayed.

Hexa-decimal code	00 – 1F		20 – 3F		40 – 5F		60 – 7F	
	code	character	code	character	code	character	code	character
0	0	(null)	32	(space)	64	@	96	`
1	1	☺	33	!	65	A	97	a
2	2	☻	34	"	66	B	98	b
3	3	♥	35	#	67	C	99	c
4	4	♦	36	$	68	D	100	d
5	5	♣	37	%	69	E	101	e
6	6	♠	38	&	70	F	102	f
7	7	▪	39	'	71	G	103	g
8	8	◘	40	(	72	H	104	h
9	9	○	41	)	73	I	105	i
A	10	◙	42	*	74	J	106	j
B	11	♂	43	+	75	K	107	k
C	12	♀	44	,	76	L	108	l
D	13	♪	45	-	77	M	109	m
E	14	♫	46	.	78	N	110	n
F	15	☼	47	/	79	O	111	o
0	16	+	48	0	80	P	112	p
1	17	┴	49	1	81	Q	113	q
2	18	┬	50	2	82	R	114	r
3	19	┤	51	3	83	S	115	s
4	20	├	52	4	84	T	116	t
5	21	┼	53	5	85	U	117	u
6	22	│	54	6	86	V	118	v
7	23	─	55	7	87	W	119	w
8	24	┌	56	8	88	X	120	x
9	25	┐	57	9	89	Y	121	y
A	26	└	58	:	90	Z	122	z
B	27	┘	59	;	91	[	123	{
C	28	X	60	<	92	\	124	¦
D	29	/	61	=	93	]	125	}
E	30	\	62	>	94	^	126	~
F	31	+	63	?	95	_	127	(blank)

Hexa-decimal code	80 – 9F		A0 – BF		C0 – DF		E0 – FF	
	code	character	code	chracter	code	character	code	character
0	128	Ç	160	á	192		224	α
1	129	ü	161	í	193		225	β
2	130	é	162	ó	194		226	Γ
3	131	â	163	ú	195		227	π
4	132	ä	164	ñ	196	▪	228	Σ
5	133	à	165	Ñ	197		229	σ
6	134	å	166	ª	198		230	μ
7	135	ç	167	º	199		231	γ
8	136	ê	168	¿	200		232	Φ
9	137	ë	169	⌐	201		233	θ
A	138	è	170	¬	202		234	Ω
B	139	ï	171	½	203	///	235	δ
C	140	î	172	¼	204	\\\	236	∞
D	141	ì	173	¡	205	▼	237	φ
E	142	Ä	174	≪	206	▲	238	ε
F	143	Å	175	≫	207	▶	239	∩
0	144	É	176	Ã	208	◀	240	≡
1	145	æ	177	ã	209	⧗	241	±
2	146	Æ	178	Ĩ	210	⧓	242	≥
3	147	ô	179	ĩ	211	▪	243	≤
4	148	ö	180	Õ	212	▪	244	⌠
5	149	ò	181	õ	213	▪	245	⌡
6	150	û	182	Ũ	214	▪	246	÷
7	151	ù	183	ũ	215		247	≈
8	152	ÿ	184	Ĳ	216	△	248	°
9	153	Ö	185	ĳ	217	‡	249	•
A	154	Ü	186	¾	218	ω	250	·
B	155	¢	187	~	219		251	√
C	156	£	188	◊	220		252	^{n}
D	157	¥	189	‰	221		253	^{2}
E	158	Pt	190	¶	222		254	■
F	159	ƒ	191	§	223		255	

Characters whose character code consists of 2 bytes

Characters of codes 1 to 31 (decimal) in the above table have 2-byte character codes. Their codes in the table should be preceded by the code 1 and the codes listed in the table should be added by 64 (decimal).

Input/output of character codes

Input from the keyboard

Normal characters ..... 1-byte code is input.
Example: Code 65 (decimal) for the character "A"

2-byte code characters ..... 1 and the other code are input.
Example: Code 1 and 67 for the character "♥"

Output using CHR$ function

Normal characters ..... 1-byte code is used as a parameter.
Example: CHR$ (66) for the character "B"

2-byte code characters ..... 2 CHR$ functions are used, of which the first one is CHR$(1), and the following one is a CHR$ function using the above listed code as a parameter.
Example: CHR$(1); CHR$(68) for the character "◆"

4. CTRL KEY FUNCTIONS

In addition to the edit key, MSX-BASIC is provided with special functions just by pressing the CTRL key simultaneously with another key.

Key pressed	Function
CTRL + B	Moves the cursor to the beginning of a word (character group punctuated by a space). When the cursor is at the beginning of a word, it is moved to the beginning of the word just before.
CTRL + C	Releases the input wait state or automatic line number generation by the AUTO command to return to the command wait state.
CTRL + E	Provides erasure from the cursor location to the last line.
CTRL + F	Moves the cursor to the beginning of the next word.
CTRL + G	Generates a beep sound.
CTRL + H	Same as the BS key.
CTRL + I	Same as the TAB key.
CTRL + J	Moves the cursor 1 line below.
CTRL + K	Same as HOME.
CTRL + L	Same as SHIFT + HOME.
CTRL + M	Same as the RETURN key.
CTRL + N	Moves the cursor to a location next to the last character in a line.
CTRL + R	Same as the INS key.
CTRL + U	Erases all the characters on a line.
CTRL + X	Same as SELECT. Undefined in MSX-BASIC.
CTRL + \	Same as ⇒ cursor key.
CTRL + [	Same as ESC. Undefined in MSX-BASIC.
CTRL +]	Same as ⇐ cursor key.
CTRL + ^	Same as ⇑ cursor key.
CTRL + _	Same as ⇓ cursor key.

INDEX

W

X

SANYO

Printed in Japan

3P02813 b (Y3BA)