Business Computer

MZ-3500

BASIC LANGUAGE MANUAL



SHARP

This apparatus complies with requirements of EEC directive 76/889/EEC.

Das Apparat stimmt mit den Bedingungen der EG-Richtlinien 76/889/EWG überein.

Cet appareil répond aux spécifications de la directive CCE 76/889/CCE.

Dit apparaat voldoet aan de vereisten van EEG-reglementen 76/889/EEG.

Apparatet opfylder kravene i EF direktivet 76/889/EF.

Quest'apparecchio é stato prodotto in confornitá alle direttive CEE 76/889/CEE.

Reinhard Sötje
Badstr. 19
7600 Offenburg
Tel. 0781 / 7 51 81

Business Computer

MZ-3500

BASIC Language Manual

© SHARP CORPORATION

PRECAUTION

This manual is based on the FDOS master supplied with the MZ-3500 series Business Computer.

1) The system software for the MZ-3500 Series Business Computer is supplied as files on disk. The system software and this manual may be revised without prior notice. Check carefully the file version numbers.
2) The utmost care has been taken in the preparation of this manual. However, if there are anything unclear, or questionable portions, or any comments, contact your local SHARP dealers.
3) The system software (FDOS master file) of Business Computer Model-3500 Series is the property of SHARP Corporation, and is copyrighted by SHARP Corporation.
   It is the system software and this manual in whole or in part without permission from SHARP Corporation.

CONTENTS

I. **INTRODUCTION** 1
1. COMPUTER EXECUTION 1
(1) Start-up of BASIC 1
(2) Manual operation 2
(3) Constant 9
(4) Variables 11
(5) Numeric expressions and character strings 14
(6) Priority levels of arithmetic operations 15
(7) Output formats of numeric values 16
(8) Array variables 19
(9) Error messages 22
2. FUNCTIONS 23
(1) Character processing functions 23
(2) Numeric processing functions 26
3. TEXT AND GRAPHICS 31
(1) Text screen and graphic screen 31
(2) Coordinate systems and cursor operation 33
(3) Color specification 34
4. FILES 35
(1) What is a file 35
(2) File types 37
5. FDOS OPERATIONS 40
(1) What is FDOS 40
(2) Terms 41
(3) FDOS commands 43
BASIC 44
INIT 45
CAT 47
BACKUP 48
COPY 50
SCOPY 52
KILLALL 54
DEBUG 55
6. DEFINABLE KEY USES 58
(1) Definable key difinition 58
(2) Application examples of definable keys 59
(3) Listing and erasing definable keys 60

II. BASIC PROGRAMMING 61
1. PROGRAMMING IN GENERAL 61
(1) Program 61
(2) Programming 62
2. PROGRAM CORRECTING AND EDITING 66
(1) Program modification 66
(2) Correcting procedure 67
(3) Line insertion and deletion 69
3. DEBUG FUNCTION 71
(1) 1-line debugging 71
(2) Trace debugging 73

III. BASIC COMMAND WORDS 75
1. SYNTAX (RULE) SYMBOL DEFINITION 75
2. COMMANDS 77
RUN 78
VRUN 80
CONT 82
LIST 85
EDIT 90
NEW 92
DELETE 93
ERASE 95
AUTO 96
REN 98
TR 100
TR STOP 102
3. STANDARD STATEMENT 103
LET 104
DISP 107
PRINT 110
IMAGE 113
INPUT 117
DATA 120
READ 121
RESTORE 123
REM 125
STOP 126
END 127
DIM 128
GO TO 131
GO SUB 134
RETURN 136
ON...GO TO 138

ON...GO SUB 139
IF...THEN 141
IF... : ELSE 144
FOR...TO...STEP/NEXT 149
4. SPECIAL STATEMENTS 156
CALL 157
SUB 160
SUBEND 162
KEY 163
KEY IN 165
POP 171
ON ERROR 172
OFF ERROR 175
ON KEY 176
OFF KEY 180
ON TIME 181
OFF TIME 184
RESUME 185
MOVE 187
MAXS 190
MINS 192
SEARCH 194
DEFFN 198
MUSIC 201
WAIT 203
DEFINT 205
DEFREAL 206
DEFSTR 207
DEG 209
RAD 210
RANDOM 211
ERROR MUSIC 212
5. CRT DISPLAY (PRINTER OUTPUT) CONTROL STATEMENTS 213
DISP CLEAR 214
SET 216
RESET 218
POS 219
ENTER 220
CURSOR 222
MVCUR 224
LINE 225
BL 227
RV 228
RE 229

TABLE 231
SCROL 233
CHANGE DISP 235
CHANGE PRINT 236
COLOR 237
6. GRAPHIC PROCESSING STATEMENTS 238
SDISP 239
ODISP 241
GCOLOR 242
CLS 244
SCALE 246
PSET 248
PRESET 250
LINE 251
CIRCLE 255
PAINT 260
GENTER 265
GDISP 268
GPRINT 270
GCURSOR 271
GINPUT 273
GCHR 276
7. STATEMENTS RELATED TO MACHINE LANGUAGE 280
NEW 281
ERASE 282
POKE (PEEK) 283
USR 284
8. RS-232C INTERFACE CONTROL STATEMENTS 287
CHANNEL 288
SEND 290
RCV 294
SENREV 297
OPCHNL 299
POLLING 300
RLIST 302
TERM 304

IV. DISK BASIC INSTRUCTION WORDS 305
1. INSTRUCTION LIST 306
2. GENERAL FILE MANIPULATION COMMANDS 307
LOCK 308
UNLOCK 310
COPY 311
KILL 312

CAT 313
RNAME 316
LPRINT 317
LINPUT 319
BYE 321
3. DATA FILE MANIPULATION STATEMENTS 322
MAXFILE 323
CREATE 325
OPEN 327
CLOSE 329
3-1. Sequential file manipulation statements 330
BPRINT # 332
BINPUT # 334
PRINT # 335
INPUT # 338
SEP # 340
3-2. Randam file manipulation statements 341
RFORMAT # 343
PUT # 345
GET # 346
4. PROGRAM FILE MANIPULATION COMMANDS 347
SAVE 348
STORE 350
LOAD 352
CHAIN 354
MERGE 356
LOAD SUB 358
ALOAD 363
5. OTHER FILE MANIPULATION COMMANDS 365
MSAVE 366
MLOAD 367
VSAVE 368
VLOAD 369
GSAVE 370
GLOAD 372
KSAVE 373
KLOAD 374
INDEX

I. INTRODUCTION

1. COMPUTER EXECUTION

Operating a computer is not simply a market of switching it on and using it, a number of preparating operations must be carried out.
Also the system most be loaded with instructions or a set of instructions, i.e. a program, to execute a job. (For programs, see II. 1. "Programming in General" (page 61).)
Some kind of language is needed to give instructions BASIC, as (Beginner's All purpose Symbolic Instruction Code) described in this manual, is one of these languages.
However, computers can only understand machine language instructions.
When instructions are given in BASIC, the computer translates each instruction into machine language to be executed. This translation is done by a machine-language program called the BASIC interpreter, generally called BASIC.
BASIC of this system is supplied as a disk file*1).

(1) Start-up of BASIC

To set this system in a state in which BASIC can be used (BASIC operation state), start up the BASIC interpreter using the following steps. (Refer to IV. 2. "Power On" (page 19) in the Owner's Manual.)

① Insert the disk (FDOS master disk) supplied with this system in the drive (drive No.0) on the right side, and turn on the power.

② When the power is ON, the following message is displayed on the CRT.

```
FDOS   22007A(V1.0A)
READY
```

③ Then key in as follows with the keyboard.

BASIC [ENTER]

④ The following message is displayed on the CRT.

```
BASIC  22007B(V1.0A)
READY
>
```

Thus BASIC is read from disk into memory*2) and the initialization of BASIC is completed.

Note

To protect the original FDOS master disk, you are recommended to make a backup disk (see page 48).

---

*1) File: Program or data recorded on a disk.

*2) Memory: Program and data storage location.

When step ② is executed, IPL*1) (Initial Program Loader) is automatically started, FDOS*2) (Floppy Disk Operating System) is loaded into the memory, and control is passed to FDOS. This state is called the FDOS operation state in which FDOS commands*3) can be used.
When step ③ is executed in this state, BASIC is loaded into the memory and control is passed to BASIC. This state is called the BASIC operation state in which BASIC instructions can be used. "READY" means the instruction wait state; in the BASIC operation state, manual operations*4) and program execution in BASIC can be carried out.
The blinking symbol below "READY" is called a cursor, which indicates the character display position on the screen.

(2) Manual operation

Manual operation is the immediate execution of keyed-in instructions, not stored in the memory as a program. Its use somewhat similar to that the calculator. For example, when the following DISP statement*5) (see page 107) is keyed in, the answer of 240 x 1150 is obtained immediately.

```
DISP 240 * 1150 [ENTER]
```

DISP is an instruction to display the specified contents on the screen. The specified content in this case is the value of 240 * 1150, where the asterisk (*) means multiplication (x).
When the [ENTER] key is pressed, the instruction is executed and the result is displayed as shown below. If the [ENTER] key is not pressed, the computer does not execute the instruction displayed on the screen.
(Result)

```
DISP 240 * 1150
 276000
```

Program execution may be halted to carry out manual operation.
For manual operation, set the MODE CHANGE switch*6) to OP mode. Most manual operations can be done with the switch set to PRO mode. However, to protect the program, the switch should be set to OP mode before manual operation. For example, if a numeric value is entered by mistake instead of an expression, the value does not affect the program in OP mode; however, in PRO mode, the program line*7) indicated by the value entered is erased. (See II. 2. 3 "Line Insertion And Delection.")

---

*1) IPL: A program used to load the system program from the FDOS master disk. This program operates only at power ON.
*2) FDOS: See I. 6. (1) "FDOS" (page 40).
*3) Command: Instructions mainly used in manual operations.
*4) Manual operations: See (2) "Manual Operations".
*5) Statement: Instructions mainly used in programs.
*6) MODE CHANGE switch: Refer to page 23 in the Owner's Manual.
*7) Program line: See II. 1. (1) "Programs" (page 61).

1) Numeric value entry

Use the following keys to enter numeric values:

Sign ..... + or - (+ sign may be omitted.)
Digit ..... 0 to 9
Decimal point ..... .
Exponent ..... E
Hexadecimal ..... &

Example:

Numeric value	Key operation
-123.45	-123.45
1.2×10^{-34}	1.2E-34
$3A71_{16}$	&3A71

2) Arithmetic operations

Use arithmetic operators + (addition), - (subtraction), * (multiplication), / (division), and ∧ (exponentiation) for arithmetic operations.

Example:

Key operations for arithmetic operations (manual)

	Arithmetic expression	Key operation			Result
Addition	$8+3=$	DISP	8+3	ENTER	11
	$-1.2+4=$	DISP	-1.2+4	ENTER	2.8
Subtraction	$4-8=$	DISP	4-8	ENTER	-4
	$5.8-3.1=$	DISP	5.8-3.1	ENTER	2.7
Multiplication	$3\times5.1=$	DISP	3*5.1	ENTER	15.3
	$2.5\times(-1.6)=$	DISP	2.5*(-1.6)	ENTER	-4
Division	$35\div4=$	DISP	35/4	ENTER	8.75
	$-7.8\div(-3)=$	DISP	-7.8/(-3)	ENTER	2.6
Mixed	$(5+3-1)\div2=$	DISP	(5+3-1)/2	ENTER	3.5
	$3\times2-21\div3=$	DISP	3*2-21/3	ENTER	-1
Exponentiation	$3^4=$	DISP	3^4	ENTER	81
	$2.8^3=$	DISP	2.8^3	ENTER	21.952

An asterisk (*) is used for multiplication instead of the usual x. A slash (/) is used for division instead of ÷.

The DISP statement may be omitted, except for arithmetic operations where the first numeric value is in exponent format. In OP mode, DISP may be omitted for any arithmetic operation. (For details, see the DISP statement (page 107).)

3) Relational operations

The relational operators used for relational operations are as listed next. The result of a relational operation is 1 (true) if the relational expression is satisfied; otherwise, 0 (false). A relational expression is two sets of data connected by a relational operator.

Relational operator	Meaning
=	Left side is equal to right side.
>	Left side if greater than right side.
<	Left side is less than right side.
> = = >	Left side is greater than or equal to right side.
< = = <	Left side is less than or equal to right side.
< > > <	Left side is not equal to right side.

Example:

Key operations for relational operations (manual)

Key operation			Result
DISP	3>5	ENTER	0
DISP	3<5	ENTER	1
DISP	1.2>=2	ENTER	0
DISP	1.2<=2	ENTER	1
DISP	3<>5	ENTER	1
DISP	3<>3	ENTER	0

Note

Although the above example uses only numeric data, relational operation can be performed on character data also. Character data are compared using their ASCII value (equivalent code) beginning at the first character.

For ASCII codes, refer to "ASCII code table" (Appendix-1) in the Owner's Manual.

Example:

Relational operations with character data

Key operation	Result	Explanation
DISP "ABC"> "ABD" ENTER	0	Data up to the second character is the same. The ASCII code of the third character "C" on the left side is less than that of "D" on the right side; therefore, this relational expression is not satisfied and the result is 0.
DISP "MEMORY" >"MEMORIAL" ENTER	1	Data up to the fifth character is the same. The ASCII code of the sixth character "Y" is greater than that of "I"; this relational expression is satisfied and the result is 1. (Number of characters is not significant.)

4) Logical operations

Use logical operators AND, OR, and NOT for logical operation. AND and OR require two numeric expressions, whereas NOT require one numeric expression. Relational operators may be used in these numeric expressions. (When a relational operator is used in a numeric expression, the numeric expression must be enclosed in parentheses.) All logical operations process 16-bit binary data*). The most significant bit is used to indicate the sign: 1 is for minus and 0 is for plus. So, the value(s) of the numeric expression(s) which can be processed are in the range of −32767(1111111111111111 in binary representation) to 32767 (0111111111111111 in binary representation). (Decimals are ignored.) ignored.)

i) Numeric expression AND numeric expression

Calculates the logical product of the two numeric expressions.

If the two numeric expressions are relational expressions, the result is 1 only when both relational expressions are true.

Example:

1. `DISP 12 AND 10` [ENTER]

In this example, 12 and 10 are converted to binary data, their logical product is obtained, and the result is displayed on the screen. The logical product is calculated as shown below.

0000000000001100 ····· Converts 12 into 16-bit binary data.

××××

0000000000001010 ····· Converts 10 into 16-bit binary data.

|| || || ||

0000000000001000 ····· Logical product. (8 in decimal representation)

Product is 0 Decimal 8

To obtain the logical product, corresponding bits are multiplied by each other according to the rules: 1 x 1 = 1, 1 x 0 = 0, 0 x 1 = 0, and 0 x 0 = 0. The result is decimal 8 (binary 1000).

2. `DISP(3>1)AND(6>=4)` [ENTER]

In this example, both relational expressions are true; 1 is displayed as the result.

Note

When using an AND operation as a conditional expression in an IF statement (page 141), an arithmetic operator * may be used instead of AND.

ii) Numeric expression OR numeric expression

Calculates the logical sum of the two numeric expressions. When the two numeric expressions are relational expressions, the result is 1 when either or both relational expressions are true.

*) Binary data: Data represented by binary numbers.

Example:

1. DISP 6 OR 10 [ENTER]

In this example, 6 and 10 are converted into binary data, their logical sum is calculated, and the result is displayed on the screen. The logical sum is calculated as shown below.

0000000000000110 ····· Converts 6 into 16-bit binary data.

++++

0000000000001010 ····· Converts 10 into 16-bit binary data.

‖ ‖ ‖ ‖

0000000000001110 ····· Logical sum. (14 in decimal representation)

Sum is 0 Decimal 14

To obtain the logical sum, corresponding bits are added to each other according to the rules: 1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, and 0 + 0 = 0. The result is decimal 14 (binary 1110).

2. DISP (6>8) OR (7<>1) [ENTER]

In this example, one of the relational expressions (on the right side) is satisfied; 1 is displayed on the screen as the result.

Note

When using an OR operation as a conditional expression in an IF statement, arithmetic operator + may be used instead of OR.

iii) NOT numeric expression

Calculates the negation of the numeric expression result.

Example:

1. DISP NOT (-1234) [ENTER]

In this example, -1234 is converted into binary data, its negation is calculated, and the result is displayed on the screen. Negation is calculated as shown below.

1000010011010010 ····· Converts -1234 into 16-bit binary data.

0111101100101101 ····· Negation. (31533 in decimal representation.)

The value of the numeric expression is converted into binary data, and the 1's complement*) of each bit taken. (1 and 0 are reversed.) The result is 31533 (binary 0111101100101101).

2. DISP NOT (8>=7) [ENTER]

In this example, the relational expression is satisfied; the value in parentheses is 1. Negation of 1 is carried out and the result — 32766 (binary 1111111111111110) is displayed on the screen. (not 0)

Note

-0 (binary 1000000000000000) is regarded as 0 (binary 0000000000000000).

---

*) 1's complement: The value in which each bit is subtracted from 1. (0 → 1, 1 → 0)

3. DISP NOT ((-3) AND (-4)) [ENTER]

In this example, -32767 is displayed. (not 32767)

The following calculation is made.

1000000000000011 ····· Converts -3 into binary data.

1000000000000100 ····· Converts -4 into binary data.

1000000000000000 ····· The logical product is -0.

-0 is converted to 0. The negation of 0 is calculated, and the result is -32767 (binary 1111111111111111).

5) This system provides functions for specific operations and processes. For details, see 2. "Functions" (page 23).

6) Calculation using variables

Details of variables are given in (4) "Variables" (see page 11). Here, consider variables as locations where data is assigned. To assign a numeric value (characters) to a variable, specify variable = numeric expression (character string). Note that this equation does not mean that the left side is equal to the right side but that the data on the right side is assigned to the variable on the left side.

Example:

1. Assignment example

Assignment	Key operation
A←1.234	A=1.234 [ENTER]
A←A+1	A=A+1 [ENTER]
X←B	X=B [ENTER]
$C\leftarrow\sqrt{A^2+B^2}$	C=SQR (A^2+B^2) [ENTER]
B←A、C←A	B=A:C=A [ENTER]

2. Examples of trigonometric functions using variables (unit: degree)

Operation	Key operation	Display
X←30	X=30 [ENTER]	30
sinX	DISP SIN X [ENTER]	0.5
cosX	DISP COS X [ENTER]	0.8660254038
tanX	DISP TAN X [ENTER]	0.5773502692

7) Stack

For instance, when calculating 23 − 11 * SQR 4, SQR 4 is first calculated but data is read sequentially from 23; therefore, 23 − 11 * must be saved in memory. The memory area for storing this is called a stack. The stack for saving operands, such as 23 and 11, is called the operand stack. The stack for saving operators, such as − and *, is called the operator stack. The operand stack has 25 locations, and the operator stack has 48 locations. Note the number of locations in these stacks when writing a numeric expression.

Example:

Data movement in the stack during execution of

10+3*SQR 7-LOG TAN(12+68)^2^3/EXP 2 [ENTER]

Operation	Operand stack				Operator stack				
	Location 1	Location 2	Location 3	Location 4	Location 1	Location 2	Location 3	Location 4	Location 5
10	10								
+	10				+				
3	3	10			+				
*	3	10			*	+			
SQR	3	10			SQR	*	+		
7	7	3	10		SQR	*	+		
−	1.793···E1				−				
LOG	1.793···E1				LOG	−			
TAN	1.793···E1				TAN	LOG	−		
(	1.793···E1				(	TAN	LOG	−	
12	12	1.793···E1			(	TAN	LOG	−	
+	12	1.793···E1			+	(	TAN	LOG	−
68	68	12	1.793···E1		+	(	TAN	LOG	−
)	80	1.793···E1			TAN	LOG	−		
^	7.536··E−1	1.793···E1			^	−			
2	2	7.536···E1	1.793···E1		^	−			
^	2	7.536··E−1	1.793···E1		^	^	−		
3	3	2	7.536··E−1	1.793···E1	^	^	−		
/	1.832··E−1	1.793··E−1			/	−			
EXP	1.832··E−1	1.793··E−1			EXP	/	−		
2	2	1.832··E−1	1.793···E1		EXP	/	−		
[ENTER]	1.792···E1								

8) Clearing and correction of keyed-in data and recovery from errors

Key-in errors in manual operations can be corrected by clearing the entry by pressing the [CL] key or by moving the cursor with the [←] or [→] keys.

i) Clearing with the [CL] key

Clear the entry with the [CL] key and enter the data from the beginning again. [CL] key can also be used to recover from an error state.

ii) Correction with [←] and [→] keys

Correction with the [CL] key requires keying-in from the beginning, whereas by moving the cursor using the [←] and [→] keys, partial correction of erroneous keyed-in or operation errors can be done.

○ When an error is caused by a keying-in error or by an operation error, press the [CL] key or the [↑] key to recover from the error and make the necessary correction.

iii) Correction with [↑] key

Press the [↑] key to recover from an error and move the cursor to the error location or to the position immediately after it, so that correction can be made immediately. When the [↑] key is pressed in a normal state, the previously keyed-in contents are displayed on the screen so they can be corrected.

(3) Constant

Constants are data elements with fixed contents.

1) Numeric constants

The system can handle four types of constant.

i) Decimal format

Example:

```
123
-12.34
.567 ..... The leading zero may be omitted. (0.567)
2345.0
```

ii) Exponential format (Exponent part is indicated by E.)

Example:

```
1.2E+13 ..... Represents 1.2 x 10^13. (+ in the exponent part may be omitted.)
4.56E-8 ..... Represents 4.56 x 10^-8.
21.3E15 ..... Represents 21.3 x 10^15.
```

iii) The ratio of the circumference of a circle to its diameter is expressed as PI in this system. The value of PI = 3.14159265358 is stored in memory. (Key in `PI` [ENTER] .)

Example:

The circumference of a circle of radius 5 can be obtained by keying-in the following:

`2*PI*5` [ENTER]

iv) Hexadecimal notation

Prefix hexadecimal notation with an ampersand (&). The range of hexadecimal notation this system can handle is &0 to &FFFF.

Hexadecimal	Decimal
&0	0
&1	1
&2	2
&3	3
⋮	⋮
&A	10
&B	11
&C	12
&D	13
&E	14
&F	15
&10	16
⋮	⋮
&1F	31
&20	32
⋮	⋮
&FFFF	65535

2) Character constants

A character constant is a string of characters and/or symbols enclosed by a pair of quotation marks (") or apostrophes ('). A character constant is used as is during execution and is enclosed by quotation marks or apostrophes.

Example:

```
"GOOD MORNING"
"$1,800"
'TOTAL ?'
'2+5'..... 2 + 5 is not calculated, but '2 + 5' is handled as a character string.
```

See the example shown below.

```
>DISP 2+5
 7
>DISP '2+5'
2+5
```

Note

All character strings which can be entered from the keyboard can be handled as character constants; however, quotation marks cannot be included in a character string enclosed by a pair of quotation marks, nor can apostrophes be included in a character string enclosed by a pair of apostrophes.

Example:

```
Incorrect   'BOY'S HAT'
Correct     "BOY'S HAT"
```

(4) Variables

Variables are the memory areas for storing (assigning) numeric constants and character constants previously discussed. The variables for storing numeric constants are called the numeric variables, and the variables for storing character constants are called the character variables. Numeric variables are divided into real-type numeric variables and integer-type numeric variables. Real-type numeric variables are simply called numeric variables in general. Variables are also classified into simple variables and array variables; variables usually refer to simple variables. (In this manual, simple variables are called variables.) For array variables, see 1. (8) "Array Variables" (page 19).

The organization of the variables explained here is shown below.

- Variables
  - Numeric variables
    - Real-type numeric variables
      - Real-type numeric (simple) variables
      - Real-type numeric array variables
    - Integer-type numeric variables
      - Integer-type numeric (simple) variables
      - Integer-type numeric array variables
  - Character variables
    - Character (simple) variables
    - Character array variables

The name assigned to a variable is called a variable name. Variable names must comply with the following rules.

i) A character name must consists of 1 to 8 characters.
(The 9th and the subsequent characters are ignored.)

ii) A character name must begin with an alphabetic character.

iii) The second and subsequent characters must be alphanumeric characters or number.

iv) Function names (INT, etc.), commands, statements, and other keywords are not allowed as, or in, variables.

Example:

Correct) ABC
E1
answer
Total
AtoB1

Incorrect) DATA1 ..... Includes a keyword DATA.
A* ..... Includes operator *.
COST ..... Includes the cosine function name COS.

1) Numeric variables (numeric simple variables)
Numeric variables include real-type numeric variables and integer-type numeric variables. Declare the type as real or integer by the type declaration symbol suffixed to numeric variables.

i) Real-type numeric variables
Suffix a variable with a cross-hatch symbol (#) for type declaration. # may be omitted in the normal state (power ON time).
Real-type numeric variables can handle numeric values 0 and positive and negative values which can be expressed in up to 12 significant digits and exponent -99 to 99.

Example:

```
A=12.34 ..... Assigns 12.34 to variable A. (#-omitted example)
Total#=2300 ..... Assigns 2300 to variable Total #.
AVERAGE#=-.031 ..... Assigns -0.031 to variable AVERAGE#.
```

ii) Integer-type numeric variables
Suffix a variable with a percent symbol (%) for type declaration. Integer-type numeric variables can handle integers in the range of -32767 to 32767. Integer-type numeric variables use less memory area than real-type numeric variables, and the processing speed is faster. Use of integer-type numeric variables is recommended when handling integers in this range.

Example:

```
ABC%=24 ..... Assigns 24 to variable ABC%.
I2%=-32700 ..... Assigns – 32700 to variable I 2%.
ANSWER%=76.21 ..... Assigns 76 to variable ANSWER.
                    (Since ANSWER% is an integer-type numeric variable, the
                    decimal part is ignored.)
```

2) Character variables (simple character variables)
Declare a character variable by suffixing a variable with a dollar sign ($) or an at-sign (@).

Example:

```
AB$="B.MANUAL" ..... Assigns "B.MANUAL" to character variable AB$.
DATE@="82.09.01" ..... Assigns "82.09.01" to character variable DATE@.
```

Note

1. Up to 16 characters can be assigned to a character variable (simple character variable).
2. The differences between the character variables suffixed with $ ($-type character variables) and the character variables suffixed with @ (@-type character variables) are are follows.

① An ASCII character CHR$ 0, null, (see the CHR$ function (page 23) explained later) can be assigned only to @-type character variables.

Example:

```
A@=CHR$ 0 ..... Null is assigned to variable A@.
A$=CHR$ 0 ..... Null cannot be assigned to variable A$ and an error
                occurs. (ERROR. 19)
```

② Up to 16 characters can be assigned normally to a $-type character variable and an @-type character variable. However, the maximum number of characters which can be assigned to an @-type character simple variable can be specified by the DIM statement (page 128). (The DIM statement can specify the maximum number of characters which can be assigned to a character array variable of $- and @-types.)

Example:

```
DIM A@*10 ..... Specifies the maximum number of characters which can
                be assigned to character variable A@ as 10.
```

③ $-type character variables use less memory area than @-type character. The use of $-type character variables is recommended when a null code is not required.

④ $-type character variables can be assigned to @-type character variables.

Example:

```
A@=A$ ..... Assigns A$ to A@.
```

3) Variable declaration

The identification of real number variables, integer variables, and $ and @ type string variables by use of declaration symbols are described in the above sections 1) Numerical variables, and 2) String variables.

Another method may be used to declare variable types, that is, to declare variable types by using the DEFREAL, DEFINT, and DEFSTR statements. (These statements may be used in manual operation or in programs.) For details, see the DEFREAL statement (page 206), the DEFINT statement (page 205), and the DEFSTR statement (page 207), which are explained later.

4) Special numeric variables (pseudo variables)

The following special numeric variables (pseudo variables) are also available. However, make sure they do not conflict when your own variable useage as they use reserved words.

i) TIME

Indicates the time with the integer part indicating the month, day, and hour (24-hour system), and the decimal part indicating the minute and second (base 60). (For example, 40813.5035 indicates April (month 4) 8th, 1:50:35 pm (13:50:35).)

Key in `DISP TIME` [ENTER]. The value of variable TIME keeps changing. To set a time, for example, May (month 5) 5th 10:58:30, key in `TIME=50510.583` [ENTER]

The time count is started at the moment when the [ENTER] key is pressed.

ii) ERL (ERror Line number)

Indicates the error line number when an error occurs in a program. For the details of the line numbers, see II. 1. (1) "Programs" (page 61).

iii) ERN (ERror Line number)

Indicates the error code number when an error occurred. For the error code numbers, refer to IV "Error code list" (page 51) of the Appendix.

Example:

Key in as follows.

3/0 [ENTER]

An error occurs, and ERROR 3 is displayed on the bottom line of the screen. Press the [CL] key to release the error state and key in the following:

ERN [ENTER]

3 is displayed as the value of pseudo variable ERN. Error code 3 means that the error was caused by "division by zero" (refer to the error code list).

(3 on the bottom line of the screen displayed upon occurrence of the error also indicates the error code number.)

(5) Numeric expressions and character strings

1) Numeric expressions

The following numeric expressions can be used in this system.

Type	Application example	Reference page
Numeric constant	A=-12.34	9
Numeric variable	A=B	12
Numeric function	A=SIN 30 B=ABS (SIN X)	26
ASC function	A=ASC "J"	23
VAL function	A=VAL "1,234.5"	24
LEN function	A=LEN "GOOD-BYE"	24
FN function-name (numeric expression) FN function-name (character-string) (Results are given as numeric values)	A=FN M(5,3) B=FN N(EXP 34.5)	198
numeric expression arithmetic-operator numeric expression	A=LEN "MAIN"+LEN "SUB" B=5.6^3	3
numeric expression relational-operator numeric expression character-string relational-operation character-string	A=B<=C+6 B="NAME">"NAMES"	3
numeric expression AND numeric expression numeric expression OR numeric expression NOT numeric expression	A=13 AND 11 B=6 OR 13 C=NOT (-1234)	5
sign numeric expression	A=+SIN (A+B) B=-ATN 2.5	3

2) Character strings

The following character strings can be used in this system.

Type	Application example	Reference page
Character constant	A$="SALES"	10
Character variable	A$=B$	12
CHR$ function	A$=CHR$ 37	23
STR$ function	A$=STR$ (5+2*3)	24
STRING$ function	A$=STRING$ (10"*")	24
LEFT$ function	A$=LEFT$ (B$,4)	25
RIGHT$ function	A$=RIGHT$ (B$,6)	25
MID$ function	A$=MID$ (B$,5,3)	25
SPA function	A$=SPA (3*N)	25
FN function-name (numeric expression) FN function-name (character-string) (The result of these functions is a characters string)	A$=FN B$(5,3,Y$)+Z$ B$=FN C$("MORNING")	198
Character-string + character-string	A$=B$+RIGHT$ (C$,5)	104

(6) Priority levels of arithmetic operations

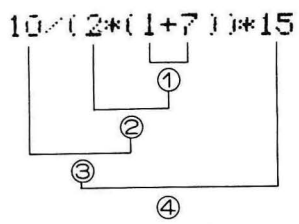
Arithmetic operations are executed in the order of the priority levels as shown below.

Highest priority — Parentheses ( )
Functions and NOT
Exponentiation ^
Signs + and -
*, /, and AND
+, -, and OR
Lowest priority — Relational operators =, <, >, <=, >=, <>

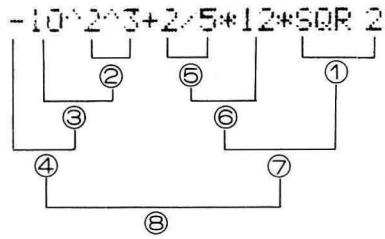
The operations on the same priority level are executed from left to right. When consecutive exponentiations (^) are specified, the exponentiation part alone is executed from right to left. All the operations in parentheses are executed first, which changes the execution sequence. When the operation execution sequence is not clear, it is better to use parentheses to clarify the sequence.

Example:

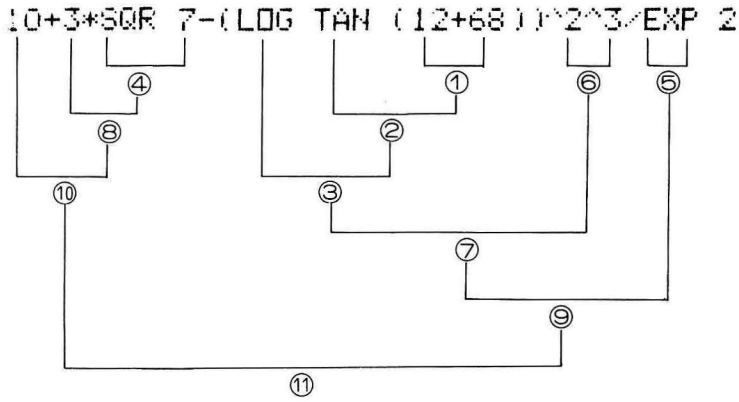
1. 10 ÷ (2 x (1 + 7)) x 15 is calculated in the order as shown below.



2. $-10^{2^3} + 2 \div 5 \times 12 \times \sqrt{2}$ is calculated in the order as shown below.



3. $10 + 3 \times \sqrt{7} - \{ \log \tan (12 + 68) \}^{2^3} \div e^2$ is calculated in the order as shown below.



(7) Output formats of numeric values

Three numeric formats can be used to output numeric values on CRT display or to the printer:

① Standard format
② Fixed-point format
③ Floating-point format

When the power is turned on or a NEW command (see page 92) is executed, the standard format is automatically set.

To change the numeric format; use the FIXED statement to set the fixed-point format, and the FLOAT statement to set the floating-point format. To return to the standard format, use the NORMAL statement.

1) Standard format

Standard format automatically selects the fixed-point or the floating-point format according to the numeric value to be output so that the operation result is displayed in a format easy to comprehend. The digits exceeding the significant digits are truncated.

Format: NORMAL

This statement may be used both in manual operations and in programs.

Explanation:

1. The maximum number of significant digits is 12; the 13th and the subsequent digits are truncated.
2. The leading zeros and the trailing decimal zeros are suppressed.
3. Fixed-point or floating-point format is automatically selected to suit the numeric value to be output.(x indicates the numeric value to be output.)

All values in the range $10^{-4} \leqq |X| < 10^{12}$
Values in the range $10^{-11} \leqq |X| < 10^{-4}$ with up to 11 decimal digits and X = 0 } Fixed-point format

All numeric values other than those above . . . . . Floating-point format

Example:

When NORMAL is executed (at power ON)

Numeric value in memory	Output format	
123456789012	123456789012	Fixed-point format
-123456789012	-123456789012	
0.12345678901	0.12345678901	
1.23E-9	0.00000000123	
12.34567890123	12.3456789012	The 11th decimal digit is truncated.
-1.00	-1	The trailing decimal zeros are suppressed.
0.000056789012	5.6789012E-05	Floating-point format
-1234567890123	-1.23456789012E+12	

2) Fixed-point format

Fixed-point format keeps the number of decimal digits constant; the number of decimal digits and rounding can be specified in this format. To set the fixed-point format, use the FIXED statement.

Format: FIXED m.n (FIXED numeric expression)

m: Number of decimal digits (integer 0 to 11)

n: Rounding specification (integer 0 to 9)

n	Rounding
0	Round down
1	8 down 9 up
2	7 down 8 up
3	6 down 7 up
4	5 down 6 up
5	4 down 5 up (round off)
6	3 down 4 up
7	2 down 3 up
8	1 down 2 up
9	Round up

This statement may be used both in manual operations and in programs.

Explanation:

1. Rounds the (m + 1) th decimal digit according to the rounding specification n, and outputs the numeric value concerned down to the m th decimal digit.
2. When the fixed-point format (FIXED m. n) is specified, the floating-point format (FLOAT m. n) is temporarily used for the output value whose absolute value exceeds 10^{12}.
3. FIXED 0. n format does not output decimal digits.
4. When the total of the specified number of decimal digits and the number of the integer digits (maximum 11) of the numeric value to be output exceeds 12, the number of digits displayed may not be as specified, or the rounding may not be performed.

Example:

When FIXED 3.5 is executed

(Rounds off the 4th decimal digit, and outputs the numeric value down to the 3rd decimal digit.)

Numeric value in memory	Output format
12	12.000
-123.45678	-123.457
0.1296	0.130
1234567890123	1.235E+12
1.234E-4	0.000
1234567890.5699	1234567890.56

···· The absolute value of the numeric value to be output is over 10^{12}; the value is output in the floating-point format.

···· The number of output digits is too large; the number of displayed digits is not as specified nor was rounding performed.

3) Floating-point format

The floating-point format is most useful when handling very large numeric values or very small numeric values. To set the floating-point format, use the FLOAT statement.

Format: FLOAT m . n (FLOAT numeric expression)

m: Number of decimal digits (integer 0 to 11)

n: Rounding specification (integer 0 to 9)

Specification of n is the same as that for the fixed-point format. The floating-point format outputs numeric values in the format as shown below.

(-) d. d . . . . . dE ± dd

↓ Sign — Mantissa — Exponent

This means,

(-) d. d . . . . . d x $10^{\pm dd}$

↓ Sign — Mantissa — Exponent

This statement may be used both in manual operations and in programs.

Explanation:

1. The first nonzero digit of the numeric variable concerned is displayed as the first digit of the mantissa part. When the numeric value is negative, the mantissa is prefixed with a minus sign. For a positive value or zero, the numeric value is prefixed with a space.

2. For FLOAT 0. n, the decimal point is not written (integer is displayed in the mantissa part); in other cases, the decimal point is written following the first digit.

3. After the decimal point, as many decimal digits as are specified by m are displayed.

4. The exponent part consists of a plus or minus sign and two digits, which indicates the exponentiation of 10. When the exponent part is 00, the sign is +.

Example:

When FLOAT 2.9 is executed

(Rounds up the 3rd decimal digit of the mantissa, and outputs the numeric value concerned down to the 2nd decimal digit.)

Numeric value in memory	Output format
123. 45	1.24E+02
-3. 2	-3.20E+00
0. 00000123	1.23E-06
98765432101	9.88E+10
5E6	5.00E+06

(8) Array variables

When handling multiple data, the processing steps become very complex if each data item is assigned to a separate variable for processing. In such cases, array variables are very useful.

1) What is an array variable

An array variable is a variable (array variable name) followed by a number (called a subscript) enclosed in parentheses. By changing the subscript, multiple data items can be processed using one array variable. For example, to assign 10 data items to array variable A, change the subscript from 1 to 10 and assign each data item to A(1), A(2), . . ., A(10). Each of A(1), A(2), . . ., A(10) is called an array element, i.e., an array variable consists of array elements.

Note

1. Array variable A and simple variable A are totally different variables; A(1) and A can be used at the same time as two different variables.

2. Bear in mind that when using array variables not only numeric constants but numeric variables, numeric array variables, and other numeric expressions (see page 14) may be used as a subscript.

Example:

```
A(I)
A(3*I)
```

Examples of the use of subscripts is included in the explanation of the DIM statement on page.

2) Declaration of array variables

Before using an array, the size of the array variable to be used must be declared (at the beginning of the program) by the DIM statement. The DIM statement sets the maximum value of the subscript of the array variable, and at the same time reserves the memory area for the array variable.

Example:

```
DIM X(5)
```

This declares that five array elements X(1) to X(5) are to be used.
One DIM statement can declare multiple array variables.

Example:

```
DIM X(5),Y(10)
```

Note

1. The DIM statement cannot redeclare the same array variable. For example, array variable C declared by DIM C(10) cannot be redeclared by DIM C(20). (ERROR 20).
2. Over 32K bytes cannot be declared for one array variable.

3) Multi-dimensional array variables

Array variables with one subscript such as A(1), . . ., A(10) in the previous example are called 1-dimensional array variables. Array variables with two or more subscripts (multi-dimensional array variables) can also be used.

Example:

```
A(3,5)
```
..... 2-dimensional array variable

```
ST(7,4,2)
```
..... 3-dimensional array variable

As for 1-dimensional array variables, the array size must be declared by using the DIM statement before using array variables with 2 or more dimensions.

Example:

```
DIM Y(7,5)
```

The following 35 (7 x 5) array elements can be used.

Y (1, 1)	Y (1, 2)	Y (1, 3)	Y (1, 4)	Y (1, 5)
Y (2, 1)	Y (2, 2)	Y (2, 3)	Y (2, 4)	Y (2, 5)
Y (3, 1)	Y (3, 2)	Y (3, 3)	Y (3, 4)	Y (3, 5)
Y (4, 1)	Y (4, 2)	Y (4, 3)	Y (4, 4)	Y (4, 5)
Y (5, 1)	Y (5, 2)	Y (5, 3)	Y (5, 4)	Y (5, 5)
Y (6, 1)	Y (6, 2)	Y (6, 3)	Y (6, 4)	Y (6, 5)
Y (7, 1)	Y (7, 2)	Y (7, 3)	Y (7, 4)	Y (7, 5)

4) Integer-type numeric array variables and character array variables

Although all the array variables explained so far are real-type numeric array variables, integer-type numeric array variables and character array variables are also available.

Example:

```
Add#(X,Y) ..... Real-type numeric array variable
                (# may normally be omitted.)
C%(1,2,K) ..... Integer-type numeric array variable
TEL$(A,B) ..... $-type character array variable
B@(M,N) ....... @-type character array variable
```

The maximum number of characters which can be stored (assigned) to a character array variable is 16 in each array element (same number as in character variable); however, this limit may be changed by using the DIM statement.

Example:

```
DIM Name$(10)*30
```

Specify the number of characters which can be assigned to each array element in the numeric expression DIM character-array-variable * numeric expression as shown in the above example. This example specifies that the number of characters which can be assigned to each array element of Name$(1), . . ., Name$(10) is 30. Up to 255 characters can be assigned to each array element of a character array variable. (See the DIM statement (page 128).). To refer to all the elements of array variable Name$ in a program, write Name$(*). In the above example, Name$(*) represents all the 10 array elements. An array variable written like this in a program is called a generalized array variable, which can be used in specific statements.

(9) Error messages

When an error occurs, this system emits a sound three times and displays the error code number on the bottom line of the CRT. The error code number indicates the cause of the error. (Refer to IV "Error code List" (page 51) of the Appendix. When an error occurs in a program, execution is stopped and the error code number and the error line number (the program line number where the error occurred) are displayed.
For example,

```
ERROR 4 IN LINE 100
      ↓           ↓
Error code No.  Program line No. (error line No.)
```

indicates that the error occurred because a statement without a valid jump address was executed on program line 100.
The error tone may be stopped, by executing the ERROR MUSIC statement (page 212).
For error recovery and error correction, see 1. (2) "Manual Operations" (page 2).

2. FUNCTIONS

The results of specific arithmetic operations and processes can be obtained easily by using various functions below.

Note

For function processing, the numeric expressions or character strings including signs, arithmetic operators, and/or relational operators must be enclosed in parentheses.

(1) Character processing functions

Character processing functions include character strings in their operand*) or their results are given by character constants.

i) ASC character-string (ASCII)
Converts characters into character codes. The ASCII code of the first character of the character string becomes the value of function ASC. This value is given by a decimal equivalent code. For the ASCII codes, refer to "ASCII Code Table" (Appendix-1) of the Owner's Manual.
To convert ASCII codes into characters, use the CHR$ function explained below.

Example:

```
A=ASC("SHARP")
```

The ASCII code "83" of the first character "S" is assigned to A.

ii) CHR$ numeric expression (Character dollar)
Converts character codes into characters. This is the reverse function of the ASC function. This function converts the decimal or hexadecimal equivalent code indicated by the value of the numeric expression in the range –32767 to 32767 into an ASCII character. (For hexadecimal equivalent code, prefix the code with &.) Decimal equivalent codes are 0 to 255; the values outside this range are divided by 256 and the absolute value of the remainder (MOD value (numeric expression, 256)) is converted into an ASCII character.

Example:

```
A$=CHR$ 78
```

Character "N", which has ASCII code "78", is assigned to A$.

Example:

```
A$=CHR$ &5A
```

Character "Z", which has ASCII code (hexadecimal) "5A", is assigned to A$.

---

*) Operand: Data on which an operation is performed, such as A in DISP A and PRINT A.

iii) LEN character-string (length)

The length of (number of characters) in the character string becomes the value of function LEN.

Example:

```
A=LEN "MODEL-3500"
```

The number of characters 10 is assigned to A.

iv) VAL character-string (value)

Converts the character data (character string) into numeric data (numeric constant). To convert numeric data into character data, use the STR$ function explained next.

Example:

```
A=VAL "1,234.5"
```

The converted value 1234.5 is assigned to A.

Example:

```
A=VAL "6.7*5"
```

Numeric value 6.7 only is assigned to A. ($A \neq 33.5$)

v) STR$ numeric expression (string dollar)

The STR$ function is the reverse function of VAL; it converts the numeric data (Value of the numeric expression) into character data (character constant).

Example:

```
A$=STR$ (.3*2+5)
```

Character data " 5.6" is assigned to A$.

vi) STRING$ (numeric expression, character-string) (string dollar)

Indicates the character constant obtained by repeating the first character of the character string the number of times specified by the numeric expression.

Example:

```
A$=STRING$ (10,"-")
```

Character data "----------" is assigned to A$.

vii) STRING$ (numeric expression-1, numeric expression-2) (string dollar)

Converts the absolute value of numeric expression 2 into a hexadecimal notation and returns the number of digits specified by the value of numeric expression 1 as hexadecimal character data. The value of expression 2 must be in the range –65535 to 65535. When the number of digits specified by numeric expression 1 is less than the number of the hexadecimal-converted value, the excess leading part of the converted character data is truncated. Conversely, when the number of digits specified by numeric expression 1 is greater than that of the hexadecimal-converted character data, zeros are added to the leading part.

Example:

```
A$=STRING$(4,4321)
```

Character data "10E1" is assigned to A$. Showing that the hexadecimal representation of 4321 is 10E1.

Example:

```
A$=STRING$(6,4321)
```

Character data "0010E1" is assigned to A$.

viii) SPA numeric expression (space)

Indicates a character constant obtained by assigning the number of spaces specified by the numeric expression.

Example:

```
A$=SPA(2*B)
```

When B = 4, eight character spaces are assigned to A$.

ix) LEFT$ (character-string, numeric expression) (left dollar)

Assigns as many characters as are specified by the numeric expression starting form the left end of the character string.

Example:

```
A$=LEFT$(B$,4)
```

When B$ is "COMPUTER", "COMP" is assigned to A$.

x) RIGHT$ (character-string, numeric expression) (right dollar)

Assigns as many characters as are specified by the numeric expression starting from the right end of the character string.

Example:

```
A$=RIGHT$(B$,4)
```

When B$ is "MEMORY", "MORY" is assigned to A$.

xi) MID$ (character-string, numeric expression-1, numeric expression-2) (middle dollar)

Assigns as many characters as are specified by numeric expression 2 from the character position defined by the value specified by numeric expression 1 from the left end of the character string.

Example:

```
A$=MID$(B$,3,4)
```

When B$ is "ABCDEFG", "CDEF" is assigned to A$.

(2) Numeric processing functions

Numeric processing functions use numeric constants as operand and result.

i) INT numeric expression (integer)
Returns the integer part of the numeric expression value, truncating the decimal part.

Example:

```
A=INT (3.14)
```

The integer value 3 obtained by truncating the decimal part is assigned to A.

ii) ABS numeric expression (absolute)
Calculates the absolute value of the numeric expression.

Example:

```
A=ABS (2*3-4^2/2)
```

The value of the numeric expression is -2; 2 is assigned to A.

iii) SGN numeric expression (signum)
Returns – 1 when the value of the numeric expression is negative.
Returns 0 when the value of the numeric expression is 0.
Returns 1 when the value of the numeric expression is positive.

Example:

```
A=SGN (16/4-6)
```

The value of the numeric expression is -2 (negative); -1 is assigned to A. If the value of the numeric expression is 0, 0 is assigned to A; and if the value of the numeric expression is positive, 1 is assigned to A.

iv) SQR numeric expression ([Square] root)
Calculates the square root of the value of the numeric expression ($\sqrt{\text{numeric expression}}$).
When the value of the numeric expression is negative, an error results.
results.

Example:

```
A=SQR (3^2+8*2)
```

The value of the numeric expression is 25; $\sqrt{25}$ is calculated and the result 5 is assigned to A.

v) SIN numeric expression (sine)
Calculates the sine of the value of the numeric expression. The unit of the value *) is in degrees in the normal state (at power ON).

Example:

```
A=SIN 30
```

The 0.5, result of the calculation, is assigned to A. (When the unit is in degrees)

---

*) Unit of value: Normally degrees (after power ON) when trigonometric functions (SIN, COS, TAN etc.) are used. To use radians, execute the RAD statement (page 210); then the values used in trigonometric functions are set in radians. To switch from radians back to degrees, use DEG statement (page 209). For conversion between radians (R) and degrees (D), D = R * 180/PI (PI = 3.14159265358) is used.

vi) COS Numeric expression (cosine)
Calculates the cosine of the value of the numeric expression.

Example:

```
A=COS 30
```

The result of the calculation 0.8660254038 is assigned to A.
(When the unit is in degrees)

vii) TAN numeric expression (tangent)
Calculates the tangent of the value of the numeric expression.

Example:

```
A=TAN 14.5
```

The result of the calculation 2.634069132 is assigned to A.
(When the unit is in radians)

viii) ASN numeric expression (arc sine)
Calculates the arc sine ($\sin^{-1}$) of the value of the numeric expression.
This is the inverse of SIN.

Example:

```
A=ASN (-.5)
```

The result of the calculation –30, is assigned to A.
(When the unit is in degress)

ix) ACS numeric expression (arc cosine)
Calculates the arc cosine ($\cos^{-1}$) of the value of the numeric expression.
This is the inverse of COS.

Example:

```
A=ACS (.5-.2)
```

The result of the calculation 1.266103673, is assigned to A.
(When the unit is in radians)

x) ATN numeric expression (arc tangent)
Calculates the arc tangent ($\tan^{-1}$) of the value of the numeric expression.
This is the inverse of TAN.

Example:

```
A=ATN (-1)
```

The result of the calculation, –45, is assigned to A.
(When the unit is in degrees)

xi) EXP numeric expression (exponential)
Calculates the exponential numeric expression of the value of the numeric expression.

Example:

```
A=EXP 2.5
```

$e^{2.5}$ is calculated, and the result 12.18249396 is assigned to A.

xii) LN numeric expression (natural logarithm)
Calculates the natural logarithm loge of the value of the expression. This is the inverse of EXP.
(Expression value ≦ 0 results in an error.)

Example:

```
A=LN 2.5
```

$\log_e 2.5$ is calculated, and the result 0.9162907319 is assigned to A.

xiii) LOG numeric expression ([common] logarithm)
Calculates the common logarithm $\log_{10}$ of the value of the numeric expression.
(Expression value ≦ results in an error.)

Example:

```
A=LOG 2.5
```

$\log_{10}$ 2.5 is calculated, and the result 0.397400087 is assigned to A.

xiv) RND numeric expression (random)
Generates random numbers (pseudo random numbers) in the range 0 to the value of the numeric expression (exclusive). The number of significant digits of the random numbers is 12. Before using this function, the use of the RANDOM statement (page 211) is recommended because, if a RANDOM statement is not executed, the initial value of random numbers remains the same when the RND function is executed with the same numeric expression value after power ON or when memory is cleared.

Example:

```
A=RND 10+20
```

A random number in the range of 0 to 9.99999999999, generated by RND 10 plus 20 is assigned to A, i.e., a random number in the range of 20 to 29.9999999999 is assigned to A.

xv) MODIFY numeric expression (modify)

Modifies the value of the expression depending on the FIXED or FLOAT statement (page 18) specification.

Example:

```
10 A=56.789
20 FIXED 2.0
30 B=MODIFY A
```

The 3rd and the subsequent decimal digits of value A are truncated as specified by the statement on line 20, and 56.78 is assigned to B as specified on line 30.

xvi) CDMS numeric expression (change to DMS)

Converts the value of the expression from decimal into degrees-minutes-seconds.

Example:

```
A=CDMS 12.3456
```

12.3456 is converted, and the result 12.2044 is assigned to A.
(12.3456° = 12°20'44")

xvii) CDEG numeric expression (change to degrees)

Converts the value of the numeric expression from degrees-minutes-seconds to decimal. This is the inverse of CDMS.

Example:

```
A=CDEG 31.4545
```

31.4545 is converted, and the result 31.7625 is assigned to A.
(31°45'45" = 31.7625°.)

xiix) STATUS numeric expression (status)

When the value of the expression is 2, displays the number of unused bytes in the user area.

When the value of the expression is 1, displays the number of used bytes in the program area.

When the value of the expression is 0, displays the current line number.

STATUS 2 displays the total number of bytes in the user area at power ON and after the NEW command (page 92) is executed.

xix) PEEK numeric expression (peek)

Outputs the contents of the memory address indicated by the value of the numeric expression in decimal. Refer to "ASCII code table" (Appendix-1) in the Owner's Manual to convert the codes into ASCII characters. The value of the numeric expression must be in to the range of 38400 (&9600) to 53247 (&CFFF). Use the PEEK statement in association with the POKE statement (see page 283).

xx) MOD (numeric expression-1, numeric expression-2) (modulus)
Calculates the remainder of numeric expression 1/numeric expression 2.
The sign of the remainder is the same as the sign of the expression 1.

Example:

```
A=MOD (19,3)
```

19/3 is calculated, and the remainder 1 is assigned to A.

Example:

```
A=MOD (-35,6)
```

-35/6 is calculated, and the remainder -5 is assigned to A.

xxi) POINT (numeric expression-1, numeric expression-2) (point)
Checks if a point (numeric expression 1, numeric expression 2) on the CRT is lighted up, displaying "1" if it is lighted up, and "0" if it is not. The coordinates (numeric expression 1, numeric expression 2) here are the same as those used by the SET and RESET statements (see page 218) during text screen display. For graphic screen display, coordinates are specified in dots. (See 3. (2) "Coordinate System and Cursor Operation" (page 33).)
When the text screen and the graphic screen are overlaid, specify the coordinates in dots as for the graphic screen display.

Coordinate range for text screen display
$0 \leqq$ numeric expression $1 < 160$ (numeric expression 1 : 0 to 159)
$0 \leqq$ numeric expression $2 <$ 50 (numeric expression 2 : 0 to 49)

Coordinate range of graphic screen display
$0 \leqq$ numeric expression $1 < 640$ (numeric expression 1 : 0 to 639)
$0 \leqq$ numeric expression $2 < 400$ (numeric expression 2 : 0 to 399)

3. TEXT AND GRAPHICS

(1) Text screen and graphic screens

The following screens are displayed on the CRT.

CRT screen
- Text screen
- Graphic screen
  - Screen 1 (Blue when color monitor is used)
  - Screen 2 (Red when color monitor is used)
  - Screen 3 (Green when color monitor is used)

1) Text screen

The text screen displays characters. Blinking is, reverse, ruler line, and color can be specified for the character display. Blinking (blinking of the displayed characters) is specified by the BL statement (see page 227), and reverse (highlighting by reversing the color of the character display and its background) is specified by the RV statement (see page 228). Ruler line is used for table-making, which is specified by the TABLE statement (see page 231).

For color display, this system must be set in the color mode*). For color display, see (3) "Color Specification" (page 34). On the text screen, display is in character units.

*) Color mode: A mode in which color display is possible, as opposed to monochrome mode.

2) Graphic screens

Graphic screens display dots, lines (sets of dots), and shapes (sets of lines). Such displays are generally called graphics.

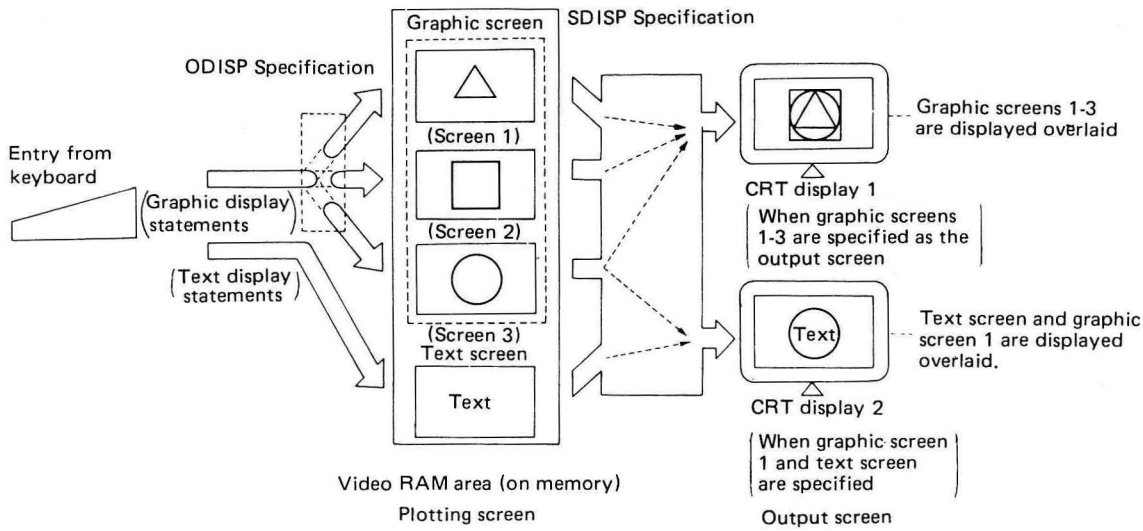
On graphic screens, display is in dot units.

A screen is composed of 640 dots (W) x 400 dots (L).

Three types of graphic screens are available: screen 1, 2, and 3. One or two or all three of these screens may be selected to be displayed on the CRT.

When two or more screens are selected, all the screens selected are displayed overlaid. (Graphic screen(s) and text screen can be displayed at the same time.)

When there are two CRT displays, screen(s) can be selected for each CRT so different contents can be displayed on each CRT. The plotting screen and the output screen are explained here:



When information to be displayed on CRT is entered with the keyboard, it is stored in the video RAM area as shown above. This area is called the plotting screen, which is specified by the ODISP statement (see page 241).

Then the information stored in the video RAM area (plotting screen) is displayed on the CRT.

The screen displayed then is called the output screen, which is specified by the SDISP statement (see page 239). For the plotting screens in the above figure, △, □, and ○ are plotted on screens 1 to 3, respectively, and text is written in the center of the text screen.

Graphic screens 1 to 3 are specified as the output screen of CRT-1, and graphic screen 1 and the text screen are specified as the output screen of CRT-2.

On each plotting screen, separate information can be plotted, or the same information can be plotted on different plotting screens. In the latter case, specify multiple plotting screens in an ODISP statement (see page 241).

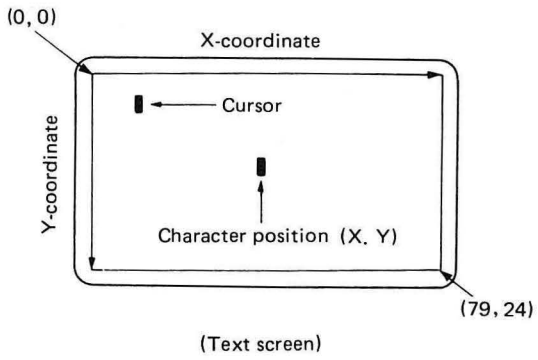
When the color mode is set, graphic screens 1 to 3 are blue, red, and green screens respectively. Color display (eight colors) on graphic screens is enabled by displaying these screens on CRT overlaying each other.

(2) Coordinate systems and cursor operation

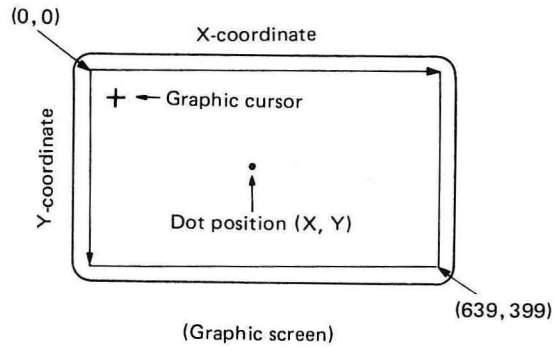
1) Coordinate systems

This system uses two coordinate systems, i.e. the text coordinate system and the graphic coordinate system to indicate positions on the CRT.

The text coordinate system is used to indicate the character positions on the text screen in character units. The graphic coordinate system is used to indicate the dot positions on the graphic screen(s) in dot units. Character positions are indicated by text coordinates (X, Y), and dot positions are indicated by graphic coordinates (X, Y). The text coordinate system and the graphic coordinate system for 80 characters (W) x 25 lines (L) screen (80 x 25 mode) are shown below.



(a) Text coordinate system



(b) Graphic coordinate system

2) Cursor operation

The cursor indicates the character or dot position to be displayed. On the text screen, a square is displayed as the cursor as shown in (a) of the above figure, and on the graphic screen, + is displayed as the cursor as shown in (b) of the above figure. (The graphic cursor + is displayed when the GCURSOR statement (see page 271)or the GINPUT statement (see page 273) is executed.) The text cursor is the size of a character. It can be moved by pressing the [→] key and/or [←] key to the position of the character to be changed from the keyboard. The center of the graphic cursor + indicates a dot.

It can be moved in four directions by pressing the [↑], [↓], [←], and [→] keys.

Dot input on the graphic screen is not effect with key input.

Enter a GINPUT statement to enable dot input with the keyboard. (For details, see the GINPUT statement.)

(3) Color specification

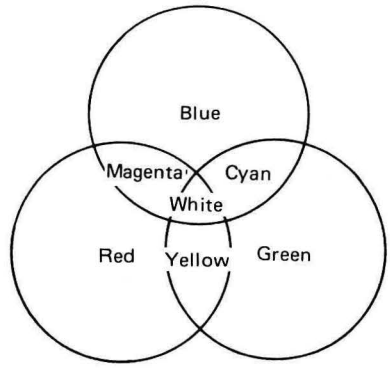
Color specification differs for the text screen and for the graphic screen. In either case first set the screen in a state in which color display can be carried out using the CHANGE DISP statement (see page 235) or the SDISP statement (see page 239). For the text screen, specify the color of the characters by using the COLOR statement (see page 237).

Blue, red, green, magenta, cyan, yellow, white, and black (total of eight colors) can be specified. For the graphic screen, specify the color of the graphics to be displayed and the background color by the GCOLOR statement (see page 242). The same colors as those for the text screen can be specified. On the graphic screen, the graphic information is automatically plotted on the blue, red, and/or green plotting screens according to the color specification.

The relationship between the specified colors and the screens on which graphics are plotted is shown below.

Graphic color	Graphic screen		
	Blue screen	Red screen	Green screen
Blue	O	X	X
Red	X	O	X
Green	X	X	O
Magenta	O	O	X
Cyan	O	X	O
Yellow	X	O	O
White	O	O	O
Black	X	X	X

O : Plotted
X : Not plotted



Three primary colors (red, green, and blue) and the colors obtained by overlaying these primary colors are shown on the right.

Note

In addition to the color specification, the output screen must be specified correctly to obtain color display on CRT. For details, see the SDISP statement (page 239). A color CRT must be used for color display.

4. FILES

(1) What is a file

1) File name

A program or data stored on an external storage media is called a file. A file of a program is called a program file, and a file of data is called a data file. The name assigned to a file is called a file name.

Floppy disks and cassette tapes are typical external storage media. Floppy disks have the following advantages over cassette tapes.

> ① Higher read/write speed.
> ② Larger capacity.
> ③ Data can be accessed quickly. (Random access)

In order to access disk files they have to be referenced by name.

Example:

To store a program with a file name NAME, key in SAVE "NAME" [ENTER] .

When a program or data is recorded on a disk as a file, the file name and the location where the file is recorded are automatically written in the file directory, which is used to check the file location when reading the file from the disk into the memory.

2) Volume name

Several floppy disks may be used at the same time, or floppy disks may be changed in the middle of program execution. To distinguish between different disks, each disk is given a name called a volume name.

For example, the supplied disk (FDOS master disk) has a volume name "SYSTEM". Before a new disk is used, it is initialized (see page 45); the volume name is assigned at this time.

Example:

To record a program with file name "NAME" on a disk with volume name "DISK1", key in as follows.

SAVE "NAME, DISK1" [ENTER]

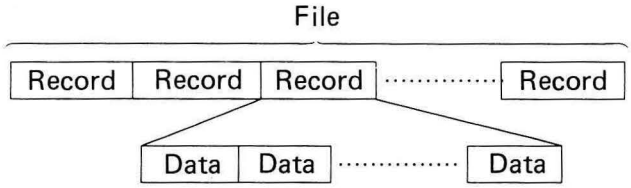
(This example specifies a volume name, not a channel drive No.)

The disk file format is as follows:

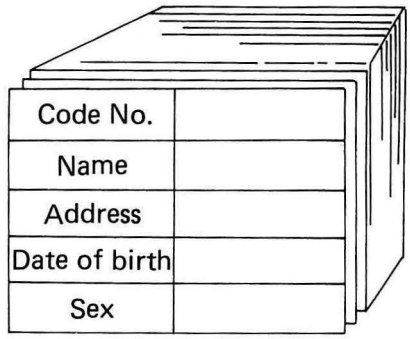
Volume name	← A name assigned to the entire floppy disk.
File directory	← This directory manages the file areas.
File 1	
File 2	
File 3	
⋮	
File n	

3) Record

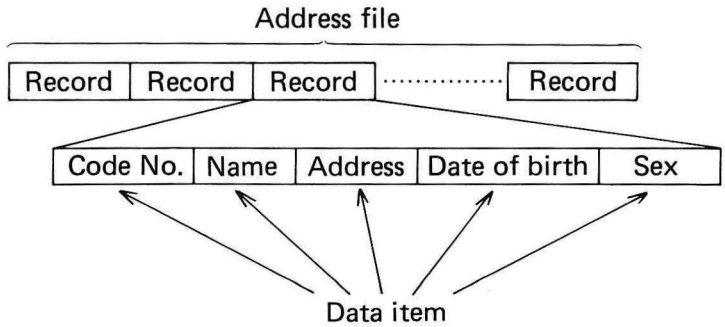
The unit of data written to/read from data files (see page 37) is called a record; a record can be defined as a group of data items read/written by input/output instructions. This means that a file is a group of records, where a record is a group of data items.



Let us record the address cards as shown below on a disk.



The code No., name, address, date of birth, and sex of each person are recorded as a record, i.e. one card corresponds to a record. Each item of a record is called a data item.



(2) File types

Files are classified into data files, program files, and other files.

1) Data files

Sequential files and random files are used as data files. Sequential files store records in the order they are written. Read/write operations cannot be begun in the middle of a file. Random files cannot record locations by numbering records; read/write can be begun anywhere in a file.

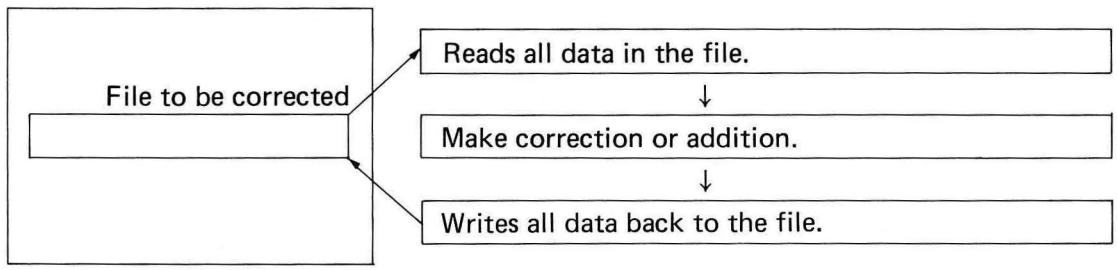
i) Sequential file

The length of a record (called a record length) is variable, and is determined by the length of the data. Variable record length makes more efficient use of a disk.

Data items and the number of items are determined by the record write statement (PRINT #statement (see page 335)) each time a record is written, which makes it possible to change the data items and the number of items in each record.

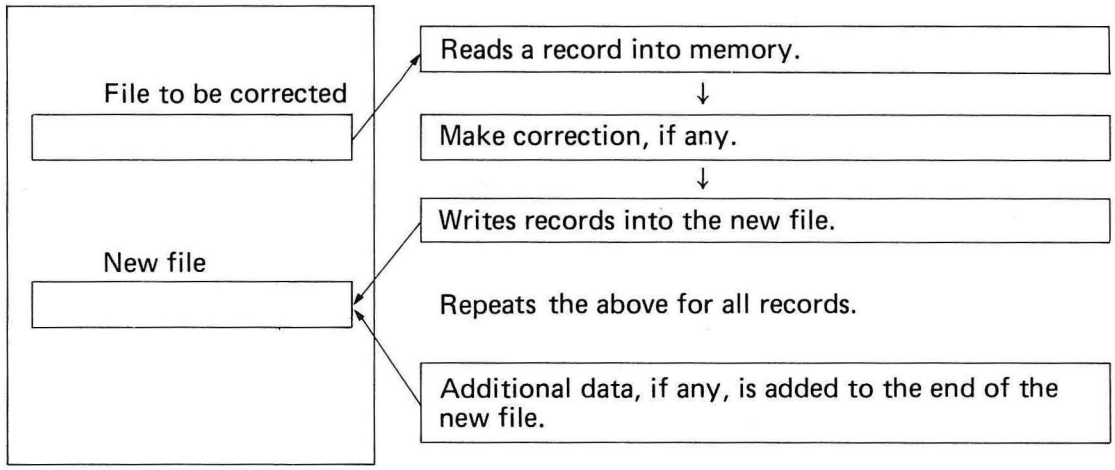
Records cannot be read/written starting at the middle of a sequential file. To read a record, all records preceding the required record must also be read.

For data correction and addition, all the data in the file is read into the memory where correction or addition is made, then all the data is written back to the same file.



When all the data in the file cannot be read into the memory, correction and addition are made by transferring all the data in the file into a new file. The concrete procedure is that a new file is set up and records in the file to be corrected are read sequentially one by one. Then the data not to be corrected is written into the new file as is, and the data which need to be corrected is corrected and then written into the new file.

Additional data is placed at the end.



Data can be added directly to the end of a file (data addition) by the access specification of the file open statement (OPEN statement (see page 327).

ii) Random file

To create a random file, the file are must first be reserved by using the CREATE statement (see page 325).

The file size must also be determined for this purpose.

The record length is fixed, and reading/writting of a disk is done in record units. Each record is numbered for record management. Therefore, any record can be read/written directly by specifying the record number.

This is because the record location can be calculated by the record length (fixed) and the record number.

Record location = File starting location + (record length) x (record number−1)

Random files allow direct reading/writting of any record for convenient data retrieval and correction.

However, the record length is fixed to the maximum length of the data items involved so that the disk utilization efficiency is not as good.

2) Program files

Program files are classified into intermediate language format program files and ASCII format program files.

i) Intermediate language format program file

A disk file which stores a program in the intermediate language format is called an intermediate language format program file. The intermediate language format is the format of a program in memory, which occupies less disk area than the ASCII format program. Usually, programs are stored on disks in this format.

ii) ASCII format program file

ASCII format program files store programs in the ASCII format (on disks).

The syntax rules of the BASIC programs are more or less the same as those for other computers; programs for this computer can be used by other computers when this format is used. The file format differs from machine to machine. The file format therefore needs to be converted by a utility program.

3) Other files

In addition to data files and program files, the following files are also used:

Machine language file
Text video RAM file
Graphic video RAM file
Definable key file

For details, see page 365.

5. FDOS OPERATIONS

(1) What is FDOS

FDOS (Floppy Disk Operating System) is a kind of system program, consisting of routines*) whose main function is file management.
FDOS is started by placing the supplied FDOS master disk in the drive (drive No. 0) on the right side and turning the power ON.
FDOS component routines.

① Floppy disk control routine ② File management routine ③ FDOS command execution control routine, RAM buffer management routine, I/O management routine ④ FDOS command execution routine

①This routine reads/writes data by physically specifying the physical sector and track on the disk. It is the basis of the floppy disk control.

②This routine manages file areas to make it easy for the user to create files without being concerned about the actual track and sector to be used.

③Some FDOS commands reside in memory and others are stored on the system disk files. When one of the latter commands is used, it is either loaded into the overlay area for execution, or replace the command by another command.
File data is exchanged between a disk and memory via a buffer (see page 323). This routine also manages the buffer area in memory.

④The FDOS command execution routine resides in memory.
DISK BASIC operates under control of routines 1, 2, and 3.
FDOS uses the above routines to execute the following commands.

FDOS commands
- BASIC
- INIT
- CAT (The only command which resides in memory)
- BACKUP
- COPY
- SCOPY
- KILLALL
- DEBUG

The commands other than CAT are executed when the corresponding execution routine is read from a disk file into the overlay area. Languages other than BASIC can also be operated in this system under control of FDOS.

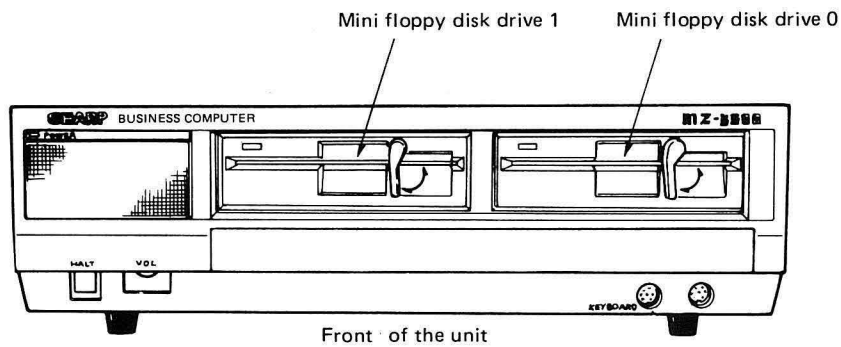
*) Routine: A program that has some general or frequent use.

(2) Terminology

i) Channel-drive No.

In addition to the built-in disk drives, additional disk drives can be attached to this system. Each drive is distinguished by its channel drive number.

Channel-drive number consists of the channel code and the drive number, where channel code is A or B and the drive No. is 0, 1, 2, or 3. Channel B is used when a standard floppy disk drive is connected. The drives in the main unit are A0 on the right side and A1 on the left side.



Front of the unit

ii) File specifications

File specifications are given by a character string including the file name, etc., to specify a file.

File specification consists of the file name, extender, volume name, channel-drive number, written in this order. (The volume name and channel-drive number may be omitted.)

Write a slash (/) before the extender, a comma (,) before the volume name, and a colon (:) before the channel-drive number.

Example:

```
"SAMPLE/TXT,SHARP:A1"
```

↑ ↑ ↑ ↖

File name Extender Volume name Channel-drive No.

File name ..... The name of a file, which consists of up to eight alphanumeric characters beginning with an alphabetic character.

Extender ..... Consists of up to three alphanumeric characters beginning with an alphabetic character.
The extender has a special meaning in some FDOS commands; however, it can be considered as a kind of file name here.

Volume name ..... The name of a disk, which consists of up to six non space alphanumeric characters beginning with an alphabetic character.

When the volume name is omitted, the disk in the drive indicated by the channel-drive number is assumed by default.
When the channel-drive number is omitted, the drive in which the disk indicated by the volume name is set is assumed by default.
When both the volume name and the channel-drive number are omitted, the disk which is set in drive A0 (right-hand drive on the main unit) is assumed by default.
When the channel code is omitted in the channel-drive number, channel A is assumed and, when the drive number is omitted, drive 0 is assumed.

Note

1 DISK BASIC gives the file specifications as a character string. However, in FDOS commands, they are given as characters which are not enclosed in quotation marks [" "].

2. File specifications "SAMPLE/TXT" and "SAMPLE" indicate different files because the former file specification includes extender TXT.

(3) FDOS commands

The commands which can be used in the FDOS operation state are called the FDOS commands to distinguish them from the commands used in the BASIC operation state. The following FDOS commands are explained here:

BASIC
INIT
CAT
BACKUP
COPY
SCOPY
KILLALL
DEBUG

BASIC command

Reads BASIC from the disk into the memory to set the system in the BASIC operation state.

Format: BAS I C

Explanation:

Insert the FDOS master disk into channel A drive 0 (right-hand drive) and turn the power ON. The system then enters the FDOS operation state in which various FDOS commands can be executed.

When this command is executed in this state, BASIC is read from the disk into memory and the system enters the BASIC operation state.

To return the operation state from BASIC to FDOS, use the BYE statement (see page 321).

When the FDOS master disk is set in drive A0 and the power is turned ON, the following message is displayed on the CRT.

Application example: CRT display

```
FDOS   22007A(V1.0A)
READY
```

Enter BASIC [ENTER] ; then the display changes as follows:

CRT display

```
BASIC  22007B(V1.0A)
READY
>
```

Now BASIC can be used. In this state, three buffers are reserved in the memory for input/output between the memory and disk. (See the MAXFILE statement (page 323).

INIT command

Initializes disks

〈Purpose of use〉
To format a disk for subsequent use.

Format: I N I T

Explanation:

Before using a new disk, the disk must be initialized using this command. This command checks each track to see if the disk is defect-free and writes on the disk the track numbers, sector numbers, addresses, etc.

The volume name is assigned at this time also. Disks in any drive of any channel can be initialized by this command.

Application example:

Set the FDOS master disk in the drive with channel-drive number A0.

Enter the INIT command.

INIT [ENTER] ..... Then the following messages are displayed one after another on the CRT. Enter your response in the order of 1 to 6.

```
FD INITIALIZE 22007C(V1.0A)

* CH.A(0) OR CH.B(1)? ①
* DRIVE NUMBER? ②
* VOLUME NAME? ③
* PRESS ENTER KEY ④

    TRACK : 00 ⑤


* END Y(0) OR N(1)? ⑥
```

① Channel code specification
Enter 0 for A, and 1 for B. 0 or 1 [ENTER]

② Drive number specification
Enter the drive number in which the disk to be initialized in set. 0 to 3 [ENTER]

③ Volume name specification
When only the [ENTER] key is pressed, "SHARP" is registered. To specify your own volume name, enter up to six alphanumeric characters (not including a space) beginning with an alphabetic character. Enter, for example, SAMPLE [ENTER]

④ Start of initialization
Before pressing the [ENTER] key to start initialization, check if the disk is really the one to be initialized and if the channel-drive number is correct. [ENTER]

⑤ Display of the track numbers during initialization
When an error occurs, the displayed track number shows the track where the error occurred.

⑥ When initialization is completed, this message is displayed.
Enter 1 to return to step ① and enter 0 to return to the FDOS state.

0 or 1 [ENTER]

Enter 1 if there is another disk to be initialized.
When an error occurs, the corresponding error message is displayed on the CRT.
In such a case, press the [CL] key or [CTRL] [HOME] keys to return to step ① to retry. Hold down [CTRL] and press [HOME] key during entry to return to step ① and press [CTRL] and [A] keys to return to the FDOS operation state.

Note

1. When entering this command, the FDOS master disk must be set in drive A0.
2. When a disk storing programs and data is initialized, all the files are erased.
   Attach a write-protect seal to disks to be retained. (Refer to page 27 of the Owner's Manual.)
3. Disks with volume name "SYSTEM" cannot be initialized.

CAT command

Outputs the catalog list of the files on a disk

〈Purpose of use〉
① To see what files are cataloged on a disk.
② To see the disk and file specifications.
③ To see the size of the unused areas on a disk.

Format: CAT␣CH m 〔*〕〔/〕

␣ : Space. In FDOS commands, the specified space must be entered.
CH : Channel code (A or B)
m : Drive No. (0, 1, 2, 3)
* : Printer output specification. When this is omitted, the catalog list is displayed on CRT.
/ : Date and time output specification.

Explanation:

1. Displays the catalog list of the files on the disk which is set in the specified drive onto the CRT. For the details of the catalog list, see CAT statement (page 313).
2. Enter * after the drive number to output the catalog list to a printer.
3. Enter a slash (/) at the end to output the file creation date and time (month, day, hour) or the last rewrite date and time.

Application example:

1. CAT␣A1 [ENTER] ····· Outputs the catalog list of the files on the disk in drive A1 onto the CRT.
2. CAT␣A0*/ [ENTER] ····· Outputs the catalog list of the files on the disk in drive A0 to a printer. The file update or creation date is also output.

Note

Use the [CR] key or [↑] key to release the error state.

BACKUP command

Copies the entire disk for back-up purpose.

〈Purpose of use〉
To save the original disks or to update the disk contents.

Format: BACKUP

Explanation:

1. Copies the entire disk to back up the original disk.
   When this command is entered, a sequence of messages is displayed on the CRT.
   Specify the channel-drive number according to the messages.
2. The FDOS master disk should be backed up onto another disk to protect the original disk.

Application example:

Set the FDOS master disk on the drive with channel-drive number A0.
Then enter BACKUP [ENTER]

CRT display

```
FD BACKUP  22007D(V1.0A)

( SOURCE DISK )          TRACK : 00 ⑥
* CH.A(0) OR CH.B(1)? ①  } Source disk information
* DRIVE NUMBER? ②        }

( DESTINATION DISK )  TRACK : 00 ⑥
* CH.A(0) OR CH.B(1)?  ③  } Copy disk information
* DRIVE NUMBER?  ④        }

* PRESS ENTER KEY  ⑤


* END Y(0) OR N(1)?  ⑦
```

Source disk: Disk to be copied
Copy disk : Disk on which a copy is made.

① and ③ specify the channel codes of the source and copy disks respectively. Enter 0 for A, and 1 for B. 0 or 1 [ENTER]

② and ④ Specify the drive numbers of the source and copy disk respectively. Enter 0, 1, 2, or 3. 0 to 3 [ENTER]

⑤ Before pressing the [ENTER] key to start copying, check if the specified disks are correctly set in both drives and if the channel-drive number is specified correctly.
[ENTER]

⑥ Display of the track number during copying.

⑦ When copying is completed, this message is displayed. Enter 1 to return to step ① or 0 to return to the FDOS operation state.

0 or 1 [ENTER]

Enter 1 to make back-up copies on several disks.

If the source disk information (①and②) is the same as the copy disk information (③ and④), i.e. when only one drive is used, the following message is displayed.

```
* INSERT SOURCE DISK AND PRESS ENTER KEY
```

Now set the copy disk in the drive and press the [ENTER] key.

Then the following message is displayed.

```
* INSERT DESTINATION DISK AND PRESS ENTER KEY
```

Remove the source disk from the drive, set the copy disk into the drive, then press the [ENTER] key.

Repeat these operations until message ⑥ is displayed.

When an error occurs, the cause is on the copy disk if the track number of the source disk is equal to that of the destination disk or on the source disk if track number of the source disk is greater than that of the destination disk.

Press the [CL] key or [CTRL] [HOME] keys to return to step ①.

Press the [CTRL] [HOME] keys during entry to return to step ①, and press [CTRL] [A] keys to return to the FDOS operation state.

Note

1. When entering this command, the FDOS master disk must be set in drive A0.
2. When the source disk includes a secure program, the source disk cannot be copied.
3. A copy cannot be made onto a disk with volume name "SYSTEM".
   (To copy onto such a disk, change the volume name by using the LPRINT command (see page 317).

COPY command

Copies a files

〈Purpose of use〉
To copy program files and data files for storage purpose.

Format: COPY␣C␣D

␣ : Space
C : File specifications (source file)
D : File specifications (copy file)

Explanation:

1. Copies the file (source file) with file specifications C into a file with file specifications D.
2. Unlike the file specifications in DISK BASIC, the file specifications in FDOS commands are specified without enclosing them in quotation marks [" "].

Simple example:

1. COPY␣CALC:A0␣DISK:A0 [ENTER]
   ..... Copies the file with file name CALC on the disk set in drive A0 into the file with file name DISK on the disk in the same drive (A0).
2. COPY␣CALC:A0␣TEST,SHARP [ENTER]
   ..... Copies the file with file name CALC on the disk set in drive A0 into the file with file name TEST on the disk with volume name SHARP.

Application example:

Let us copy the file with file name A on the disk set in drive A0 into the file with file name TEST on the disk with volume name SHARP set in drive A1.
Enter COPY␣A:A0␣TEST,SHARP:A1 [ENTER]

CRT display

```
FILE COPY  22007E(V1.0A)

* INSERT DISK AND PRESS ENTER KEY ⬚ ← Cursor blinks here
```

Set the specified disks in the specified drives, respectively, according to the above message, then press the [ENTER] key.

When the file copy is completed, the following message is displayed on the CRT and the system waits for a command.

```
    END
READY
```

The contents of the file with file name A remain unchanged.

Note

1. To enter this command, the FDOS master disk must be set in drive A0.
2. The files stored as secure programs by DISK BASIC cannot be copied.

SCOPY command

Copies a files

〈Purpose of use〉
To copy files with a single drive.

Format: SCOPY␣C␣D

␣ : Space
C : File specifications (source file)
D : File specifications (copy file)

Explanation:

1. Copies the file (source file) specified by file specifications C onto a different disk with file specifications D, using a single floppy disk drive.
2. When different channel-drive numbers are specified in file specifications C and D, the channel-drive number specified by C is used.

Simple example:

SCOPY␣CALC:A0␣TEST2,SHARP [ENTER]
..... Copies the file with file name CALC on the disk in drive A0 onto the disk of volume name SHARP, with file name TEST 2.

Application example:

Let us copy the file with file name A on the disk with volume name SYSTEM onto the disk with volume name SAMPLE, with file name SUB2, using only one drive.
Enter SCOPY␣A, SYSTEM␣SUB2, SAMPLE [ENTER]

(CRT display)

```
FILE COPY  2Z007F(V1.0A)
* INSERT SOURCE DISK OK  ← Cursor blinks here.
```

Now set the source disk in the specified drive, and press the [ENTER] key.

```
* INSERT DESTINATION DISK OK  ← Cursor blinks here.
```

Then remove the source disk from the drive, set the copy disk into the drive, and press the [ENTER] key.
Repeat the above steps until the following message is displayed.

(CRT display)

```
    END
READY
```

The system displays indicates the end of processing, and waits for a command.

The contents of the file with file name A remain unchanged.

Note

1. To enter this command, the FDOS master disk must be set in drive A0.
2. The files stored as secure programs by DISK BASIC cannot be copied.

KILLALL command

Erases all files on a disk

〈Purpose of use〉
To erase all files when they are not longer needed.

Format: KILLALL␣CH m

␣ : Space
CH : Channel code (A or B)
m : Drive number (0, 1, 2, or 3)

Explanation:

Erases all the files on a disk, except the locked files.

Application example:

Set the FDOS master disk in the drive with channel-drive number A0.
KILLALL ␣ A1 [ENTER] ..... Input of command.
CRT display

```
* INSERT DISK AND PRESS ENTER KEY
```

The above message is displayed. Now set the disk whose files are to be erased into the specified drive (A1 in this example).
Then press the [ENTER] key to start erasing files.

Note

1. To enter this command, the FDOS master disk must be set in drive A0.
2. Before starting to erase files, check that the correct disk is set in the specified drive.
3. Use the [CL] key or [↑] key to release the error state.

DEBUG command

Debugs machine-language programs.

〈Purpose of use〉
To output or change the memory contents.

Format: DEBUG

Explanation:

Sets this system into the debug state so that debug commands can be used.
Each debug command is further explained in the explanation of functions 1 to 4 below.
When DEBUG [ENTER] is entered in the FDOS state, the following messages are displayed and the system enters the debug state.

```
DEBUG  22007H(V1.0A)

USER MICRO AREA ..... ①
     START 9600 ..... ②
     END   9FFF ..... ③

* ⬚ ←Cursor blinks here.
```

① The user micro area is set up by the NEW command (page 281) or the ERASE command (page 282) in the BASIC operation state, and store user programs written in machine language.

② User micro area start address. (Refer to V "Memory Map" (page 50) of the Appendix.)
This start address may be difference according to the version number of BASIC which will be improved there after.

③ The end address of the stored machine-language program.

Now the commands explained in functions 1 to 4 below can be executed.
To return to the FDOS operation state, enter [A] [CTRL] .

● **Function 1:** Displays the memory contents on CRT.

Format: D␣AAAA␣BBBB

␣ : Space
AAAA : Start address (hexadecimal)
BBBB : End address (hexadecimal)

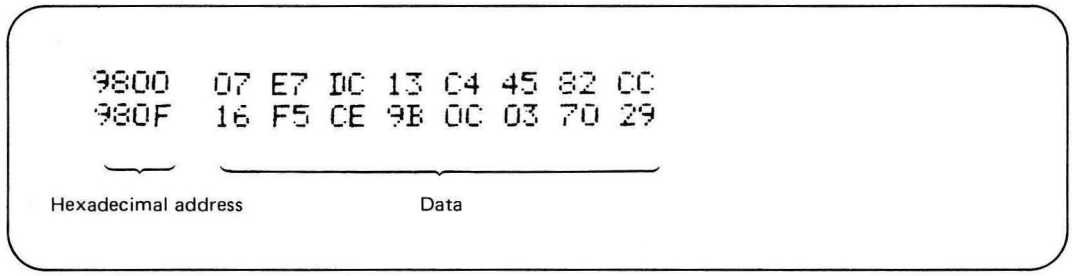
Explanation:

1. Displays the contents of the memory area from AAAA to BBBB (hexadecimal) on the CRT.
2. Press the SPACE key to stop output.
   To restart output, press the SPACE key again.
   When the [ENTER] key is pressed during output or while output is halted, the system waits for the DEBUG command to be input.
3. Only the user micro area can be output.

Example:

D␣9800␣980F [ENTER] ..... Displays the memory contents from address &9800 to &980F on CRT.

CRT display



● **Function 2:** Prints out the memory contents to the printer.

Format: P␣AAAA␣BBBB

␣ : Space

AAAA : Start address (hexadecimal)

BBBB : End address (hexadecimal)

Explanation:

1. Prints out the contents of the memory area from address AAAA to BBBB (hexadecimal).
2. Press the SPACE key to stop output.
   To restart output, press the SPACE key again.
   When the [ENTER] key is pressed during output or while output is halted, the system waits for the DEBUG command to be input.
3. Only the user micro area can be output.

Example:

P␣9800 980F [ENTER] ..... Prints out the memory contents from address &9800 to &980F.

● **Function 3:** Changes the memory contents.

Format: M␣AAAA

␣ : Space

AAAA : Memory address (hexadecimal) whose contents are to be changed

Explanation:

1. Changes the memory contents at address AAAA in the user micro area.
   When this command is entered, the contents at the specified address AAAA are displayed on the CRT.
   Now enter the new data as a 2-digit hexadecimal number and press the [ENTER] key; then the memory contents are changed.

2. When no change is to be made, the next address is displayed when the [↓] key is pressed and the previous address is displayed when the [↑] key is pressed.
When the [ENTER] key is pressed without entering data, the system waits for the DEBUG command to be entered.
In such cases, the memory contents remain unchanged.

Example:

M␣9A00 [ENTER] ..... Changes the memory contents at address 9A00.

(CRT display)

```
* M 9A00
  9A00   F3 -
```

● **Function 4:** Writes the memory contents onto a disk.

Format: S␣C␣AAAA␣BBBB

␣ : Space
C : File specifications
AAAA : Execution start address (hexadecimal)
BBBB : Execution end address (hexadecimal)

Explanation:

Writes the memory contents from hexadecimal address AAAA to BBBB onto a disk with file specifications C.
The files written in by this command can be read by the MLOAD command(page 367)of DISK BASIC.

Example:

S␣MICRO:A1␣9000␣9FFF [ENTER]
..... Writes the data at the memory address 9000 to 9FFF onto the disk set in channel A, drive 1, with file name MICRO.

Note

1. To release the error state, use the [CL] key or the [↑] key.
2. Press [CTRL][A] keys to return to the FDOS operation state.

6. DEFINABLE KEY USES

(1) Definable key definition

The DFK statement is used to define definable keys.
Definable keys 1 to 10 are defined as follows after execution of the BASIC command (page 44) after power ON. These definitions are effective until they are redefined by the DFK statement.

Key No.	1	2	3	4	5	6	7	8	9	10
Definition	+	-	*	/	(	)	=	"	$	,

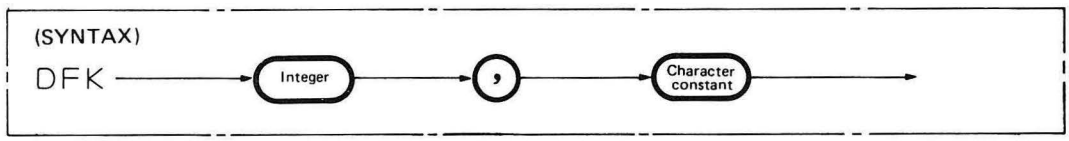
The definition for definable keys 1 to 10 are the same as when DFK0 is executed.

The DFK statement is explained here.

● **Function 1:** Allocates a user-defined Vol.

Format: DFK m, S

m : Definable key number (1 to 20)
S : Definition



Explanation:

Defines character constant S for definable key m. @ in character constant S indicates [ENTER] key operation.
By the use of @, the carriage return function can be assigned to a definable key.
Up to 12 characters can be defined by a definable key.
To define over 12 characters, attach + to the end.
Then the definition of the next key is added to the first key.
In the example below, ABC . . . XYZ is defined by definable key 1, LMN . . . XYZ is defined by definable key 2, and WXYZ is defined by definable key 3.

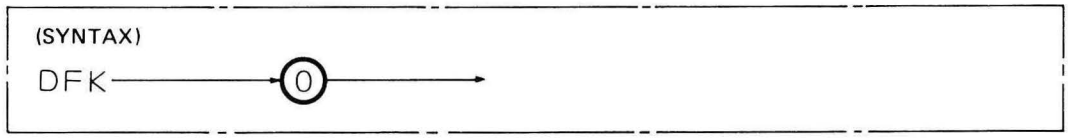
Example:

```
DFK 1,"ABCDEFGHIJK"+ [ENTER]
DFK 2,"LMNOPQRSTUV"+ [ENTER]
DFK 3,"WXYZ" [ENTER]
```

Note The definition after @ are invalid.

● **Function 2:** Defines the prescription.

Format: DFK O



Explanation:

+ , - , * , / , (,) , = , " , $, and, are automatically defined by DEF1 to DEF10 keys. The same values are defined during execution of BASIC command after power ON. These symbols can be entered by using definable keys regardless of the key mode, i.e. without switching modes.

(2) Application examples of definable keys

The ordinary keys on the main keyboard enter characters one at a time; however, with definable keys, any character string can be entered and any command can be executed. Such a function is indispensable in key operation.

When the same character string appears in a program many times, it is convenient to define the character string by a definable key and use the definable key instead of keying in the character string every time it appears.

Example:

```
10 DISP TAB 10;"LIST "
20 READ A,B
30 DISP TAB 10;A
40 DISP TAB 10;B
50 READ C$,D$
60 DISP TAB 10;C$
70 DISP TAB 10;D$
```

In the above program example, character string, i.e. DISP TAB and 10; is frequently used. Let us define this character string as a definable key, for example, DEF1 key.

Enter the following command.

DFK 1,"DISP TAB 10;" ENTER

Now, when entering the program, line 10 can be entered as shown above by simply keying in the following: 10 DEF1 "LIST" ENTER

Key in lines 30, 40, 60, and 70 in the same way.

It is convenient to define the commands frequently used in manual operations and the statements frequently used in programs as definable keys.

Example:

To define RUN, LIST and DISP to DEF2, DEF3, and DEF4 respectively, key in as follows.

```
DFK 2,"RUN@"
DFK 3,"LIST@"
DFK 4,"DISP@"
```

By adding @ at the end of each command, the command execution function can be assigned to a DEF key; simply pressing the [DEF3] key performs the same function as when the following is keyed.

LIST [ENTER]

These definitions enable program execution and program list output by a single key touch.

(3) Listing and erasing definable keys

i) Listing definable keys
Key in the following to display the definable key definition on CRT (see page 88).
LIST DFK [ENTER]

ii) Erasing definable keys
To erase the definition of all definable keys, key in as follows (see page 93).
DELETE DFK [ENTER]

II BASIC PROGRAMMING

1. PROGRAMMING IN GENERAL

(1) Programs

A program is a sequence of instructions to be performed by the computer, expressed in a computer language. The language explained in this manual is BASIC, (see I. 1 "Computer Execution" (page 1)) whose programs consist of a series of lines made up of a line number followed by a statement or statements.

①A program line (subsequently, this is called a line) is expressed as follows:

Example: `10 DISP "START"`

`10` — Line number; `DISP "START"` — Statement; both together — Program line

②When two or more statements (multistatements) are listed in a line, separate these statements with a colon (:).

Example: `10 DISP "START": INPUT A`

`DISP "START"` — Statement; `INPUT A` — Statement

③Write the line number at the beginning of the line. This indicates a line.

④Use an integer in the range 0 to 99999 as a line number.

⑤Approximately 250 characters can be written in a line. The line number is included in these characters.

Note

1. Spaces are included in the number of characters that can be written on a line. When writing many cahracters on a line, do not use a space after a keyword.
2. The upper limit of the number of characters that can be written in a line depends on the command words, etc. used in the line. If characters are input exceeding the upper limit, an error (ERROR 8) occurs.
3. If there are many characters in a line, up to 245 characters not including the line number are displayed in the list or edit state, and beyond 246 characters are displayed with symbols (¦).

A simple programming example is given below.

```
10 INPUT "RADIUS =";R ..... Assigns the numeric value input with
                             the keyboard to numeric variable R.
20 IF R=9999 GO TO 80 ..... If the input numeric value is 9999,
                             processing jumps to line 80.
30 P=2*PI*R ..... Assigns the value of 2 x π x R to
                  numeric variable P.
40 DISP "CIRCUMFERENCE =";P ..... Displays CIRCUMFERENCE = (Pvalue)
                                  on the screen.
50 S=PI*R^2 ..... Assigns the value of π x R² to numeric
                  variable S.
60 DISP "AREA =";S ..... Displays AREA = (S-value) on the
                         screen.
70 GO TO 10 ..... Jumps to line 10.
80 DISP "END" ..... Displays END on the screen.
90 END ..... Ends the program.
```

When a radius is input, the program in the above example calculates the circumference and area of the circle, and displays the results. This program repeats the same processing until 9999 is input as the radius.
For the meanings of the each statement, see the explanations given on the right side of the program.
For details, see III. 3. "Standard Statements ".

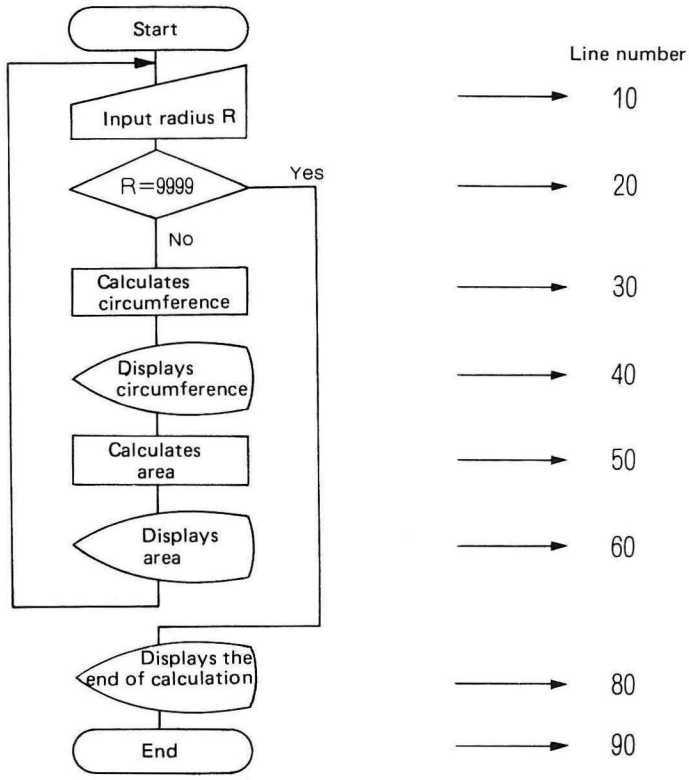
(2) Programming

Programming means to write a program. Normally, programming requires the following procedure:

① Carry out a thorough analysis of the job to be performed by the computer.
② Decide the processing procedure of the job.
③ Draw a flowchart.
④ Perform coding (write the program on coding forms).
⑤ Input the program to the computer with the keyboard, then execute the program.
⑥ Correct programming errors.
⑦ Documentation (Document of the program.)

The above programming example can be started with step ⑤, but steps ①, ②, and ③ are important for a large program.

(Flowchart)
The flowchart in step ③ indicates the program flow in a diagram. This is useful for under standing the program processing flow.
For example, the above program is indicated in the following flowchart:



This manual does not explain how to draw a flowchart and flowchart symbols.

(Program key input)
The BASIC program input procedure for this system is as follows:

① Set the mode change switch to PRO.
② First input the line number, then the statement with the keyboard. When input of one line is completed, be sure to press the [ENTER] key.
③ Repeat the operation in step ② above.

In step ②, for example, input the line number 10 of the above program as follows.
(For selection of the key mode, refer to the OWNER'S MANUAL.)

[1][0][ ][I][N][P][U][T][ ]["][R][A][D][I][U][S][ ][=][ ]["][;][R][ENTER]

[ ] indicates a space key. This is input to make the CRT display easy to read but a space does not change the meaning of program lines. The part enclosed in quotation marks (") is a character constant (page 11) and this is displayed as shown (with the specified space) when displaying the characters on the screen with a DISP statement (see page 107).

When input of the program is completed, input the following:

```
LIST [ENTER]
```

This LIST command displays the list of programs currently being stored in the main memory on the CRT in the order of the line numbers. Whether the program is input correctly or not can be checked when the program is listed.

Note

1. If the [ENTER] key is not pressed at the end of a program line, this line is omitted or combined with the next line input with the keyboard. This is because the program lines input with the keyboard are stored in the main memory by pressing the [ENTER] key.
2. The program lines do not need to be input in the same order as the line numbers.

(Program execution)

To execute the program, press the [RUN] key or input RUN [ENTER] (see page 78) with the keyboard. For the program in the above example, the processing is as follows:

1. When the [RUN] key is pressed, the screen displays the following:

```
RUN
RADIUS= ■
```

2. When 10 [ENTER] is input with the keyboard, the circumference and area are calculated and displayed as follows:

```
RUN
RADIUS =10
CIRCUMFERENCE =62.8318530716
AREA =314.159265358
RADIUS = ■
```

3. When a numeric value is input as the radius, the same processing is repeated.
4. When 9999 is input as the radius, the screen displays the following and the program ends.

```
RADIUS =9999
END
READY
>■
```

If the program is input incorrectly from the keyboard or there is an error in the program itself, an error occurs, or an incorrect result is displayed. In such cases, correct the program.
See 2 "Program Correction and Edit" (page 66) for the correction method.

(Documentation)
It is advisable to document the completed program so that this program can be understood by other persons, or can be referred to at a later date for checking or correction.

(Good programming)
It is preferable to make programs simple so that they can be checked, corrected and excuted easily. Therefore, note the following:

1. Write the program when a simple structure.

2. Do not use a complicated logical structure.

3. Use the program with REM statements (see page 125) and modularise, separate the program.

4. If the same processing is repeated, use a subroutine for this and call it from the main program. (In this way, the program outline can be easily understood by reading the main program. In addition, the number of program lines will be reduced and the program can be more easily corrected.)

The BASIC language is suitable for the above purpose, because it can be modified easily. This system provides debugging commands for detecting and correcting programming errors. Utilize these commands fully and enjoy your programming.
When programming, the first priority is to ensure that the program works correctly get used to it can be modified so that it executes more quickly and efficiently.

2. PROGRAM CORRECTION AND EDITING

There may be an error in the key operation for inputting a program or in the program itself. In such cases, the program must be corrected and edited.

Generally a program can be corrected and edited by the following operations:

① Correcting a program line already input from the keyboard (modification)
② Inserting a new program line (addition)
③ Deleting an unnecessary program line (deletion)

Note

A program can be corrected or edited only when the mode change switch is set to PRO.

(1) Program modification

To correct a program line, select the program line to be corrected, then use the [CL] or [←], [→] key.

i) Selecting a program line to be corrected

○ If a program is input erroneously from the keybaord, an error occurs at the incorrect program line during execution of the program, and the program execution is stopped. If the [↑] key is pressed in this state, the error state is reset and the line is displayed automatically on the CRT display.
For example, assume that INPUT in line number 10 is erroneously input as INPUY in the program in the above example (see page 61). If this program is executed, an error occurs at line 10 and SYNTAX ERROR IN LINE 10 is displayed on the bottom line of the CRT display.

```
>RUN


  SYNTAX ERROR IN LINE 10
```

If the [↑] key is pressed, the error state is released, line 10 is displayed, and the cursor blinks at the position after the error position.

```
>RUN
>10  INPUY["]RADIUS ="; R
```

The cursor blinks.

In this state, the program line can be corrected.

○ To recall the program line immediately after input (after inputting the program line by pressing the [ENTER] key and before pressing the next [ENTER] key), press the [↑] key. In this case, the cursor blinks at the end of the line. Correction is then possible.

○ To recall program line that cannot be called with the above [↑] key, use the [EDIT] key. (See EDIT command (page 90).)
For example, to place line number 10 in the edit state, press the following keys:

[EDIT] [1] [0] [ENTER]

The cursor blinks at the command word following the line number.

```
>10 INPUT "RADIUS =";R
    ↑
Cursor position
```

This line can be corrected in this state. When correcting the next line, recall it by pressing the following keys. It is not necessary to input the line number.

[EDIT] [ENTER]

To recall the preceding line, use an alphabetic character (A to Z) instead of the line number. For example, press the following keys:

[EDIT] [D] [ENTER]

(2) Correction procedure

i) To correct an entire line

① When the [CL] key is pressed, the line is cleared.
② Input the correct line beginning with the line number with the keyboard.
③ Press the [ENTER] key at the end.

ii) To correct a part of a line

① Move the cursor to the character position to be corrected with the [←] or [→] keys.
② Input the correct character with the keyboard.
③ Press the [ENTER] key.

iii) To delete a part of a line

① Move the cursor to the first character of the part to be deleted.
② Press the DEL key the same number of times as the number of characters to be deleted.
(Each time the DEL key is pressed, one character is deleted.)
③ Press the ENTER key.

iv) To insert characters in a line

① Move the cursor to the character position following the part where the insertion is to be made.
② Press the INS key the same number of times as the number of characters to be inserted.
(Each time the INS key is pressed, a space is inserted.)
③ Input the characters to be inserted with the keyboard.
④ Press the ENTER key.

Example:

```
10 INPUT A,B,C
20 D=B*B-A*C
30 PRINT D
40 END
```

Correct this program as follows. (Correct lines 20 and 30.)

```
10 INPUT A,B,C
20 D=B*B-4*A*C
30 DISP D
40 END
```

Step number	Operation	Display (changed lines only)
	Mode change switch: PRO	
1	EDIT 2 0	>EDIT 20 (Cursor)
2	ENTER	> 20 D =B*B-A*C
3	→ → → → → →	> 20 D =B*B-A*C
4	INS INS	> 20 D =B*B- A*C
5	4 *	> 20 D =B*B-4*A*C
6	ENTER	>
7	EDIT	>EDIT
8	ENTER	> 30 PRINT D
9	D I S P	> 30 DISP T D
10	DEL	> 30 DISP D
11	ENTER	>

Whether the line has been properly corrected or not can be checked by calling the line with the EDIT key or by outputting the program list with a LIST command (see page 85).

In step 3, the → key is pressed six times. The cursor can also be moved by keeping the key depressed (repeat key function).

(Refer to the OWNER'S MANUAL (page 23).)

(3) Line insertion and deletion

i) Line insertion

Program lines need not be entered in line number order. Therefore, lines can be inserted between existing line numbers.

To insert a line, input a line number that is between the preceding and following line numbers of the inserted line position in the PRO mode. For a program calculating the circumference and area of a circle (see page 62), for example, insert a line for calculating and displaying the diameter between lines 20 and 30 as follows:

follows:

25 DISP "DIAMETER =";2*R ENTER

(In this case, the line number may be any number in the range 21 to 29.)

Using these procedures, the above program is changed as follows:

```
10 INPUT "RADIUS =";R
20 IF R=9999 GO TO 80
25 DISP "DIAMETER =";2*R
30 P=2*PI*R
40 DISP "CIRCUMFERENCE =";P
50 S=PI*R^2
60 DISP "AREA =";S
70 GO TO 10
80 DISP "END"
90 END
```

If the line numbers are consecutive and a new line number cannot be inserted, make these line numbers non-consecutive with a REN command (see page 98).
(see page 98).

ii) Line deletion

To delete a line, input the line number to be deleted in the PRO mode, then press the [ENTER] key.

For example, delete line 100 by pressing the following keys:
100 [ENTER]
To delete a set of lines, use a DELETE command (see page 93).

Whether or not a line is inserted or deleted correctly can be checked by outputting the program with a LIST command.

3. DEBUG FUNCTION

In some circumstances, the program may generate an incorrect result, or the program execution may not end because of an endless loop (repetition). Unlike a syntax error, it is very difficult to detect the reason. In this case the debug function can be used effectively. The debug function is available for 1-line debugging and trace debugging.

(1) 1-line debugging

In the 1-line debug mode, the program is executed line by line. For the 1-line debugging, set the mode change switch to OP.

i) Procedure for debugging from the first line

①Press the DEB key while pressing the CTRL key.
②Then, each time the DEB key is pressed, the program is executed line by line.

The program for calculating the circumference and area of a circle (see page 62) is debugged in the following way:

```
10 INPUT "RADIUS =";R
20 IF R=9999 GO TO 80
30 P=2*PI*R
40 DISP "CIRCUMFERENCE =";P
50 S=PI*R^2
60 DISP "AREA =";S
70 GO TO 10
80 DISP "END"
90 END
```

Operation	Display (changed lines only)	Remarks
Mode change switch: OP		
while pressing the CTRL key. DEB	>RUN/	
(do not press the CTRL key). DEB	>CONT/	Executes line 10.
	RADIUS =□	Executes line 10.
10 ENTER	RADIUS =10	Executes line 10.
DEB	>CONT/	Executes line 20.
DEB	>CONT/	Executes line 30.
DEB	>CONT/	Executes line 40.
	CIRCUMFERENCE =62.8318530716	
DEB	>CONT/	Executes line 50.
DEB	>CONT/	Executes line 60.
	AREA =314.159265358	
DEB	>CONT/	Executes line 70.
DEB	>CONT/	Executes line 10.
	RADIUS =□	
9999 ENTER	RADIUS =9999	Executes line 10.
DEB	>CONT/	Executes line 20.
DEB	>CONT/	Executes line 80.
	END	
DEB	>CONT/	Executes line 90.
	READY	
	>	

ii) Procedure for debugging from a line within the program

① RUN/line-number ENTER

With the above specification, debugging starts from the specified line number.

② Then each time the DEB key is pressed, a line is debugged.

Note

In the 1-line debug in i) or ii) above, CONT/is displayed on the CRT. In debugging of a program that displays a graph, for example, the graphical display separated up and down by this CONT/display. In such cases, press the following keys in the PRO mode before the operation i) or ii):

(PRO mode)

EDIT ENTER

Using this, the debugging line is at the bottom line and debugging can be performed without destroying the display except the bottom line.

Example:

```
10 C$="ABC"
20 FOR I=1 TO 5
30 CURSOR 10,I:DISP I,C$
40 NEXT I
50 END
```

If this program is debugged by 1-line debugging, the display is as shown in the upper part of the following figure. If EDIT / ENTER has been specified, the display is as shown in the lower part.

```
>RUN
>CONT        1          ABC
>CONT        2          ABC
>CONT        3          ABC
>CONT        4          ABC
>CONT        5          ABC
>CONT
>CONT
READY
```

```
>EDIT
             1          ABC
             2          ABC
             3          ABC
             4          ABC
             5          ABC
READY
```

(2) Trace debugging

In the trace debug mode, the program is debugged while tracing the execution steps sequentially. (This is called a trace.)

In this type of debugging, the contents of the program line currently being executed and the execution results are displayed sequentially.

Use a TR command (see page 100) for trace debugging.

○ Procedure for trace debugging.

① TR ON ENTER

② RUN

Simple example:

When the same program example as that used for 1-line debugging is debugged by trace debugging, the CRT displays the following:

```
>TR ON
>RUN
    10 INPUT "RADIUS =";R
RADIUS =10
 10

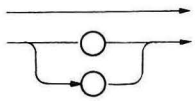
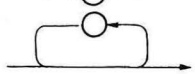
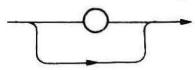

   20 IF R=9999 GO TO 80
   30 P=2*PI*R
 62.8318530716
   40 DISP "CIRCUMFERENCE =";P
CIRCUMFERENCE =62.8318530716
   50 S=PI*R^2
314.159265358
   60 DISP "AREA =";S
AREA =314.159265358
   70 GO TO 10
   10 INPUT "RADIUS =";R
RADIUS =9999
 9999


    20 IF R=9999 GO TO 80
    80 DISP "END"
END
    90 END
READY
>
```

III. BASIC COMMAND WORDS

1. SYNTAX(RULE) SYMBOL DEFINITION

To simplify explanation of the syntax and rules of commands and statements symbols are defined. The meaning of these symbols is explained in the following table.
These symbols are valid only in the description of the syntax and rules. This table does not apply to the symbols used in Simple examples and Application examples.

Symbol	Meaning
|	Indicates that one of the specified items can be selected.
〔 〕	The part enclosed in brackets 〔 〕 can be omitted. When this is omitted, the function is merely invalidated or a different function is validated.
{ }	The part enclosed in braces { } can be written repeatedly using a comma (,).
《 》	The part enclosed in angles 《 》 can be written repeatedly using a semicolon (;).
{{ }}	The part enclosed in double-braces {{ }} can be written repeatedly using a semicolon (;) or comma (,).
m, n	Indicates an integer. (Example: 10)
A, B	Indicates a numeric variable (including a numeric array variable).
S, T	Indicates a character constant. (Example: "NAME")
A$	Indicates a $- or @-type character variable (including character array variable).
M, N	Indicates a variable (numeric or character variable) (including an array variable).
X, Y, Z, U, V, W, R	Indicates a numeric expression.
C, D	Indicates a character string.
SF	Indicates a subprogram or function name.
CH, DH	Indicates an alphabetic character.
. . . . .	Indicates characters, digits, or symbols. (Example: *)
statement	Indicates a statement.
	Indicates the syntax flow.
	Indicates selection.
	Indicates repetition.
	Indicates omission.

Note

1. In the command word explanation, [ENTER] is omitted for syntax description. Actually, this key must be pressed at the end of the specification.
2. A program line can be assigned a label name; but the following statements must be used carefully:

○ SEP # statement or RFORMAT # statement

These program lines cannot be assigned label name. Use these commands immediately following the line number. (Multistatement cannot be set before the line number.) Do not include statements after the RFORMAT,# command or statements preceeding either of these commands.

○ DATA statement, SUB statement, or DEFFN statement

A label name cannot be assigned preceding these commands.

For a label name, specify a character constant of eight or fewer characters following the line number, and insert a colon (:) as a separator between the label name and the following statement.

3. A subprogram or function name comprises one to eight characters and symbols. The first character must be an alphabetic character. The following characters or symbols must be alphanumeric characters, %, #, or $.

2. COMMANDS

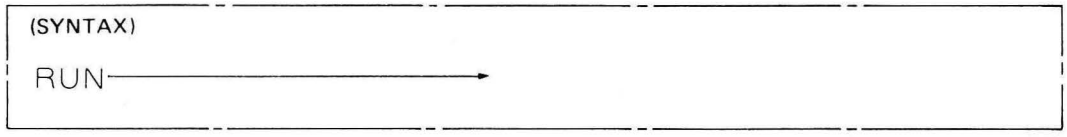
RUN	
VRUN	
CONT	
LIST	※
EDIT	
NEW	※
DELETE	
ERASE	※
AUTO	
REN	
TR	※
TR STOP	※

This chapter explains the above commands.
The commands marked with ※ can also be used in a program.

RUN

Executes a program.

● **Function 1:** Executes a program from the line with the lowest line number.
Format: RUN

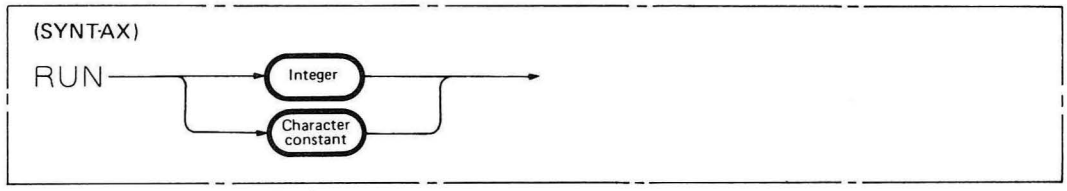


Explanation:

1. This command executes the program from the line with the lowest line number.
2. This command is stored in the RUN key.

● **Function 2:** Executes the program from a certain line number or line label name.
Format: RUN n | S

n: Line number
S: Line label name



Explanation:

1. Executes the program from the line with line number n or line label name S.
2. If a line number or line label name does not exist in the program, an error (ERROR 4) occurs. (This is the same for Function 4.)

Simple example:

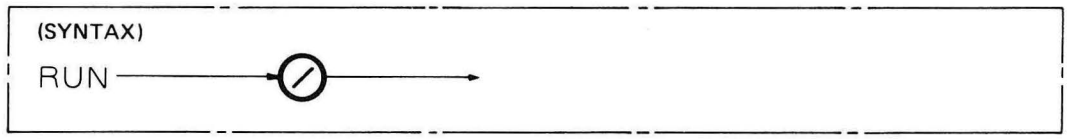
```
200 "A-1" :INPUT A$
300 PRINT A$
```

To execute the above program from line number 200, perform one of the following:

RUN 200 [ENTER] ..... A line number is used.
RUN "A-1" [ENTER] ..... A line label name is used.

● **Function 3:** Places the program in the execution mode*) beginning with the lowest line number.

Format: RUN /



Explanation:

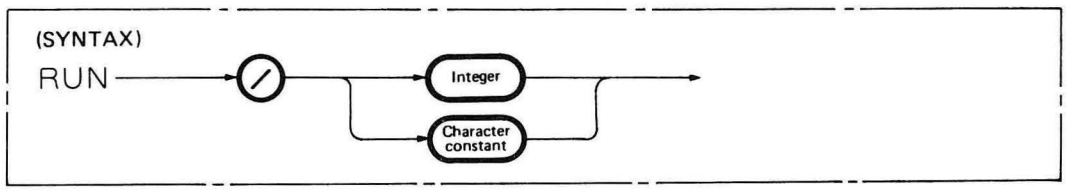
1. Places the program in the execution mode from the line with the lowest number.
   The program cannot be executed with this command only.
2. The program is executed line by line by pressing the DEB key (this key stores a CONT/command) after this command is input.
   This method is used to correct (debug) a programming error (bug).
   See II. 3 (1) 1-line debugging (page 71).
3. This command can be issued by pressing the DEB key while pressing the CTRL key.

● **Function 4:** Places the program in the execution mode from the line with a given line number or line label name.

Format: RUN / n | S

n: Line number

S: Line label name



Explanation:

1. Places the program in the execution mode from the line with line number n or line label name S.
2. Use this command for debugging from the line with line number n or line label name S.
   See Function 3 and II. 3 (1) "1-line debugging (page 71) for details.

Note

When this command is executed, initialization is performed and the contents of all variables are cleared for execution of the program.

Use the VRUN command explained next to retain the variable contents.

*) Execution mode: In this mode, the program can be executed with a CONT command (see page 82).

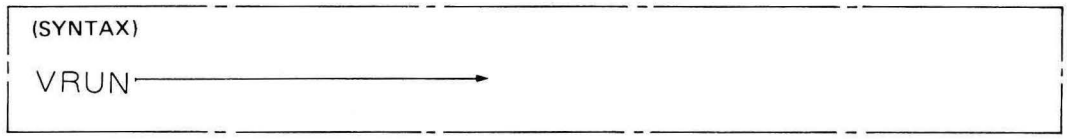
VRUN

Executes the program without clearing the variable contents.

〈Purpose of use〉
To use the execution results (variable contents) of a program to execute another program.

● **Function 1:** Executes a program from the line with the lowest line number.

Format: VRUN



Explanation:

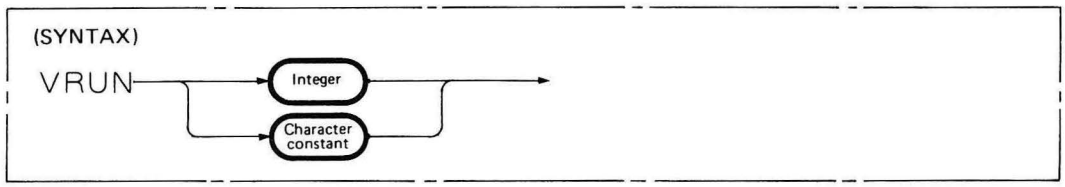
1. Executes the program from the line with the lowest line number.
2. This command can also be input by pressing the following keys: [V] [RUN]

● **Function 2:** Executes the program from the line with a given line number or line label name.

Format: VRUN n | S

n: Line number

S: Line name

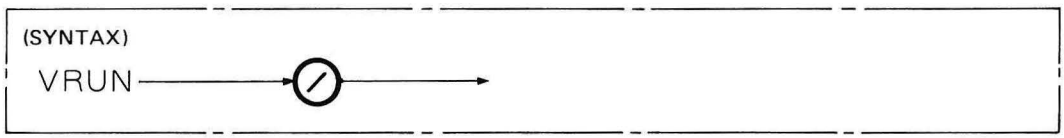


Explanation:

1. Executes the program from the line with line number n or line label name S.
2. If a line number or line label name does not exist in the program, an error (ERROR 4) occurs. (This is also the same for Function 4.)

● **Function 3:** Places the program in the execution mode beginning with the lowest line number.

Format: VRUN /



Explanation:

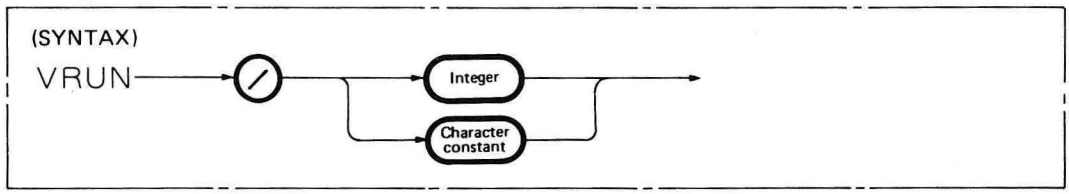
1. Places the program in the execution mode from the line with the lowest line number. This command merely places the program in execution mode and does not commence execution.
2. After this command is input, the program is executed line by line each time the DEB key (this key stores the CONT/command) is pressed.
   This function is used to correct (debug) a programming error (bug).
   See II. 3 (1) "1-line debugging (page 71) for details.
3. This command can be issued by pressing the DEB key while pressing the V and CTRL keys.

● **Function 4:** Places the program in the execution mode from the line with a given line number or line label name.

Format: VRUN / n | S

n: Line number

S: Line label name



Explanation:

1. Places the program in the execution mode from the line with line number n or line label name S.
2. Use this command for debugging from the line with line number n or line label name S.
   See Function 3 and II. 3 (1) "1-line debugging (page 71) for details.

Note

Unlike the RUN command (see page 78) explained previous, this command executes the program without clearing the variable contents.

CONT

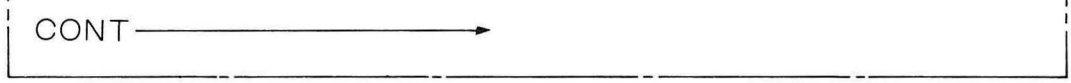
Continues execution of the suspended or stopped program.

〈Purpose of use〉
① To continue execution of a program which was suspended by the BRK key or HALT button, or stopped by a STOP statement or TR STOP command.
② To execute a program placed in the execution mode with a RUN/ or VRUN/command.

● **Function 1:** Continues execution of the program from the line following the line where the program was suspended or stopped.

Format: CONT

(SYNTAX)



Explanation:

1. Continues execution from the line following the statement where the program was suspended or stopped.
2. If a suspension was caused by the HALT button, execution is continued from the beginning of the line where the suspension occurred.
3. This command is stored in the CONT key.

Simple example:

```
400 STOP
450 INPUT A$
```

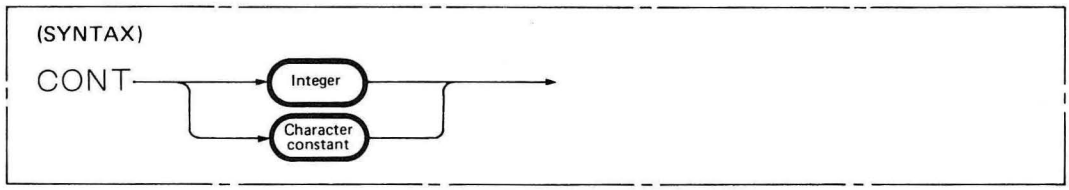
If this program is stopped at line number 400, the execution is restarted with the CONT command from line number 450.

● **Function 2:** Continues execution from the line with a given line number or line label name.

Format: CONT n | S

n: Line number

S: Line label name

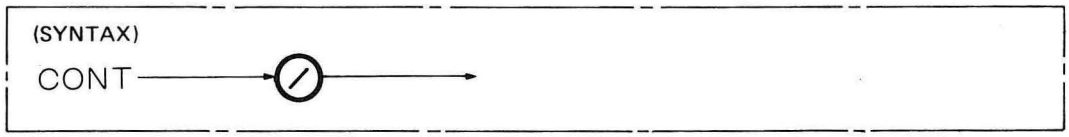


Explanation:

1. Continues execution of the program from the line with line number n or line label name S.
2. If a line number or line label name does not exist in the program, an error (ERROR 4) occurs.

● **Function 3:** Executes only the line following the statement where the program was suspended or stopped.

Format: CONT /



Explanation:

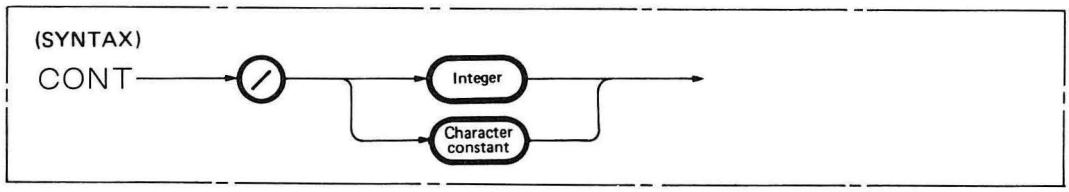
1. Executes only the line following the statement where the program was suspended or stopped.
2. If the program is suspended by pressing the HALT button, only the line where the program was suspended or stopped.
3. Use this command when debugging a program.
   See II. 3 (1) "1-line debugging (see page 71) for details.
4. This command is stored in the DEB key.

● **Function 4:** Executes only the line with a given line number or line label name.

Format: CONT / n | S

n: Line number

S: Line label name



Explanation:

1. Executes only the line with line number n or line label name S.
2. If a line number or line label name does not exist in the program, an error (ERROR 4) occurs.

Note

If this command is executed while the program has not been suspended or stopped, or while the execution mode is not set with a RUN command (see page 78) or a VRUN command (see page 80), and an error (ERROR 34) occurs.

LIST

Outputs the list of (Lists) the programs currently in the memory.

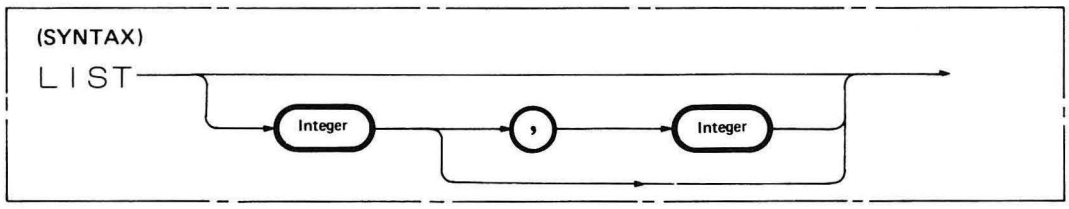
〈Purpose of use〉
① When listing the program on the CRT.
② When listing the program on a printer.
③ When listing the cataloged definable key contents on the CRT.
④ When checking whether or not programs are stored in the memory.

● **Function 1:** Displays the program list on the CRT.

Format: L I S T 〔m 〔, n〕〕

m : First line number

n : Last line number



Explanation:

1. Lists program lines with line numbers m to n on the CRT.
   To list only one line with line number n, use the format LISTn, n (the same line number is specified twice).
2. When 〔, n〕 is omitted, all program lines beginning with line number m are displayed.
3. When 〔m, n〕 is omitted, the whole program is displayed.
4. If the list to be displayed is too big to fit on one screen, the list can be scrolled.
   To stop the display before the end, press the space bar.
   (This function is the same as that of the CAT command (see page 313).)
5. If m or n indicates a line number that does not exist in the program, all program lines in the range are displayed.
   IF m > n, no error occurs, but the command is ignored.

Simple example:

```
100A=1
110"A":B=5*A+13
120DISP"A=";A,"B=";B
130A=A+.5
140IFA>10THEN"Z"
150GOTO"A"
160"Z":END
```

To list all program lines of the previous program, input the following with the keyboard:

LIST ENTER

The list is displayed as follows:

```
100 A=1
110*"A":B=5*A+13
120 DISP "A=";A,"B=";B
130 A=A+.5
140 IF A>10 THEN "Z"
150 GO TO "A"
160*"Z":END
```

For a line which has a label, an asterisk (*) is displayed automatically following the line number. (Line numbers 110 and 160)

● **Function 2:** Displays the listing a secure program on the CRT.

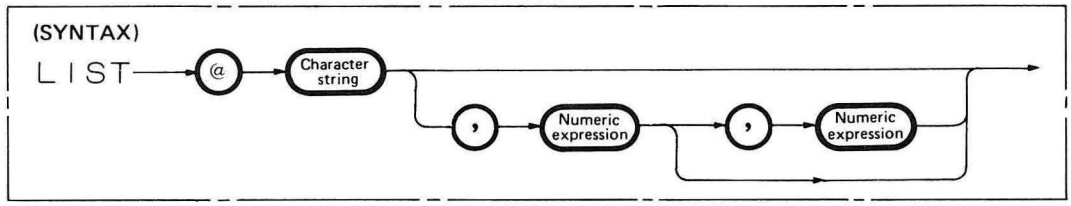
Format: **LIST @C [, m [, n]]**

@ : Specifies a secure program.

C : Password

m : First line number

n : Last line number



Explanation:

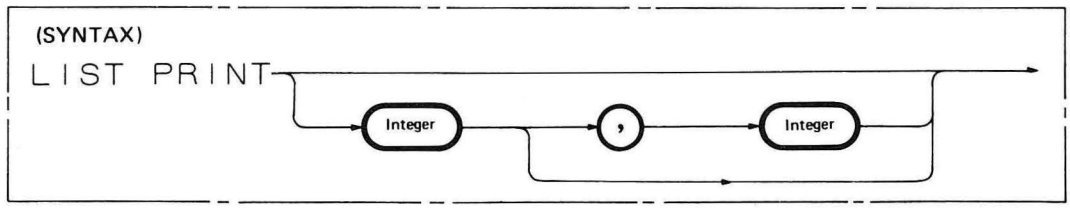
1. Displays the list of a secure program which has a password with character constant C, from line numbers m to n on the CRT.
2. The details are the same as those of Function 1.

● **Function 3:** Outputs the program list to the printer.

Format: **LIST PRINT [m [, n]]**

m : First line number

n : Last line number



Explanation:

1. Outputs the listing which has line numbers m to n, to the printer.
   The output is a maximum of 80 characters.
2. The details are the same as those of Function 1.

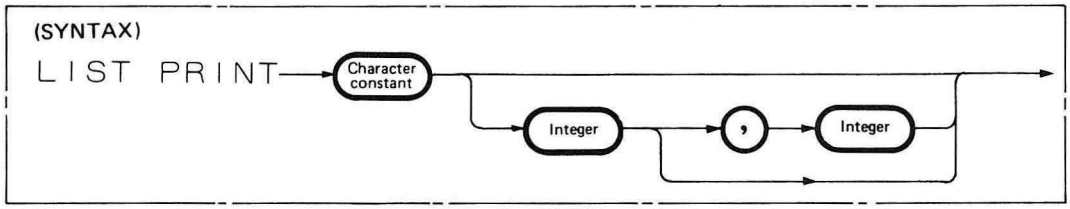
● **Function 4:** Outputs the program list to the printer with the title and paging, etc.

Format: LIST PRINT S [, m [, n]]

S : Output title name

m : First line number

n : Last line number



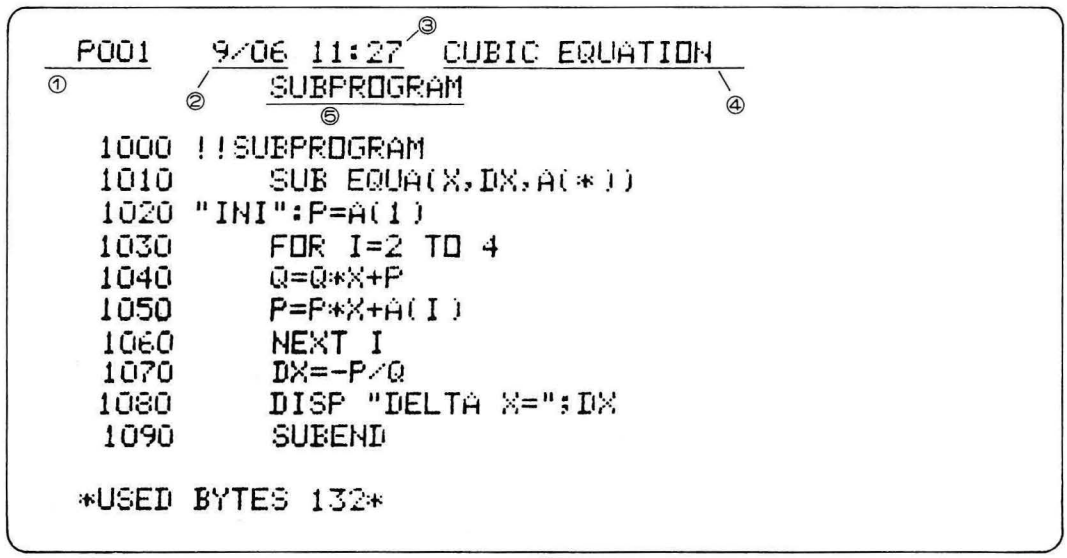
Explanation:

1. Outputs the page number, month/day, hour: minute, and title indicated by character constant S, then outputs the program list having line numbers m to n, to the printer. The output list is a maximum of 80 columns.
2. The details are the same as those of Function 1.
3. A page comprises 59 lines including the title, and a space of 7 lines is output between pages. Before outputting the list, adjust the print paper so that the print head is not more than 7 lines below the perforation.
4. This command is useful for classifying and filling various programs.
5. If !! is specified after the line number, the page is fed and the contents on the same line after !! are output as a subtitle. This subtitle is output on each page until the next !! is specified.
   (This function is useful for arranging subroutines.)
6. At the end of the program list output, the number of bytes used in the memory user area (value of STATUS 1) is output.
7. Specify the title or subtitle using 20 or fewer characters.
   (If more than 20 characters are input, only the first 20 characters are output.)
8. If the HALT button is pressed while outputting to the printer, the list output is suspended, and the number of bytes used is output, then the page is fed. (Form feed occurs.)

Simple example:

```
1000 !!SUBPROGRAM
1010 SUB  EQUA(X,DX,A(*))
1020*"INI":P=A(1)
1030 FOR I=2 TO 4
1040 Q=Q*X+P
1050 P=P*X+A(I)
1060 NEXT I
1070 DX=-P/Q
1080 DISP "DELTA X=";DX
1090 SUBEND
```

If LIST PRINT"CUBIC EQUATION",1000 [ENTER] is executed for the above program, the following list is output:



① Page number
② Month/day
③ Hour: minute
④ Title name specified in the LIST PRINT command
⑤ Subtitle name given after !!

!! is a kind of REM statement (see page 98).
This does not affect the execution of the program.

● **Function 5:** Displays the contents of the definable keys on the CRT.
Format: LIST DFK

(SYNTAX)
LIST DFK ——→

Explanation:

1. Displays the contents currently defined by the definable keys on the CRT.
   See I. 7. "Use of Definable Keys" (page 58) for the method of defining the definable keys.
2. The contents of DEF1 to DEF10 are displayed on the left side of the CRT and those of DEF11 to DEF20 are displayed on the right of the CRT.

Display example:

```
LIST DFK
  1 +                  11 RUNG
  2 -                  12 LIST
  3 *                  13 DISP
  4 /                  14 INPUT
  5 (
  6 )
  7 =
  8 "
  9 $
 10 ,
READY
```

If there are no contents defined, nothing is displayed (for example, DEF15 to DEF20). If the contents of the next definable key are added, symbol ↓ is displayed at the end of the definition. (See I. 7. (1) "Definable Key Definition Method" (page 58).)

EDIT

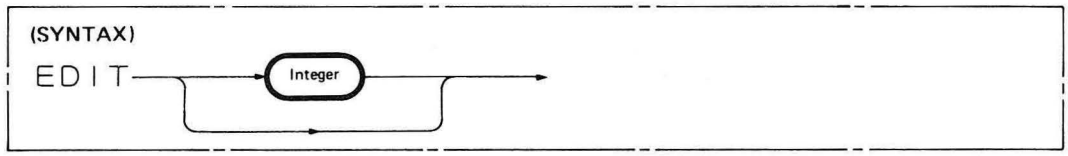
Calls a program line to enable correction.

〈Purpose of use〉
When correcting a program.

● **Function 1:** Specifies and calls a line number.

Format: EDIT (X)

X: Line number



Explanation:

1. Displays the line which has line number X (the fractions are truncated) on the CRT and enables correction (places the program in the edit state).
2. If this command is executed omitting line number X after correcting line number X, the line which has the line number following line number X is placed in the edit state. Therefore, for the correction of consecutive program lines, the second and subsequent line numbers need not be specified.
3. If a line number which does not exist in the program is input, the line which has the nearest line number is placed in the edit state.
4. The EDIT key stores characters EDIT.

Simple example:

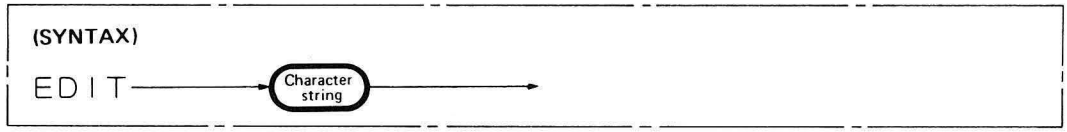
To correct program lines which have line numbers 300 and 400 in a program beginning with line number 100 and which have increments by 100 for the subsequent line numbers, perform the following operations:

① Set the mode change switch to PRO.
② EDIT 300 ENTER ..... Places the line which has line number 300 in the edit state.
③ Correct program line 300 and press the ENTER key at the end.
④ EDIT ENTER ..... Places the next line which has line number 400 in the edit state.
⑤ Correct program line 400, then press the ENTER key.

● **Function 2:** Calls the line preceding the corrected line.

Format: EDIT CH

CH: Alphabetic character



Explanation:

1. When X is omitted for Function 1, the line succeeding the line which has line number X is placed in the edit state. For Function 2, the program line preceding the line which has line number X is placed in the edit state.
2. The same function can be get for the input of arbitrary alphabetic characters in character string CH.

Simple example:

Assume that a program begins with line number 100 and has an increment by 100 for the subsequent program lines.

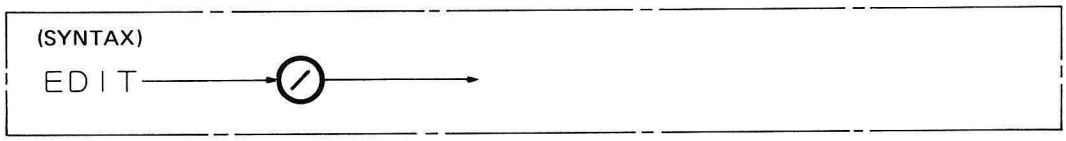
If correction of line 300 is found to be necessary while correcting line 400, correct line 400, then perform the following operations:

① EDIT D ENTER ····· Places line 300 preceding this line in the edit state.
(The alphabetic character may be other than D.)

②Correct program line 300, then press the ENTER key.

● **Function 3:** Use the CRT bottom line as the debug line in the debug mode.

Format: EDIT /



Explanation:

1. If the 1-line debug function (see page 71) is used, a manual command such as CONT/ is displayed on the CRT together with the execution results.
   To avoid this, this function displays manual commands only on the bottom line of the CRT.
2. Execution of this command prevents the information in the upper 15 lines on the CRT from being destroyed by the display of the manual commands.
3. To input this command with the keyboard, set the mode change switch to PRO, press the EDIT key, then input /.
   To reset the debug mode, press the following keys: EDIT ENTER CL
4. See II. 3. (1) "1-line debugging (page 71) for program debugging.

Note

1. The mode change switch must be set to PRO before correcting a program.
2. See II. 2. "Program Correction and Edit" (page 66) for program correction method.

NEW

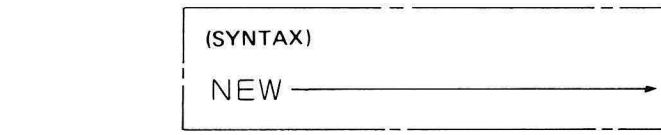
Erases all program lines and variables.

〈 Purpose of use 〉

To erase all programs and variables currently being stored in the memory to allow input of a new program.

Format: NEW

(SYNTAX)



Explanation:

1. Erases all program lines and variables currently being stored in the memory.
   This command also invalidates various declarations.
   This command does not erase the definable key definitions.
2. As a special case, all program lines and variables can be erased during or after execution of the program by using this command in the program.

DELETE

Deletes the specified program line from the memory.

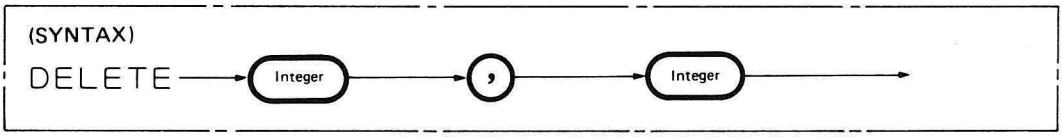
〈Purpose of use〉
① To delete a part of the program.
② To delete the preceding definition to define new contents for the definable keys.

● **Function 1:** Deletes the program lines.

Format: DELETE m, n

m : Start line number
n : End line number

(SYNTAX)



Explanation:

1. Deletes the program lines beginning at line number m and ending at line number n.
2. When deleting only one line, specify the line number of the line to be deleted for both m and n.
   If only the line number of the line to be deleted is input without using this command, the line is deleted. (See simple example 2.)
3. If a line number not existing in the program is input, line numbers between m and n are deleted.
   (If there are no lines between line numbers m and n, an error (ERROR 4) occurs.
4. If m > n, an error (ERROR 4) occurs.

Simple example:

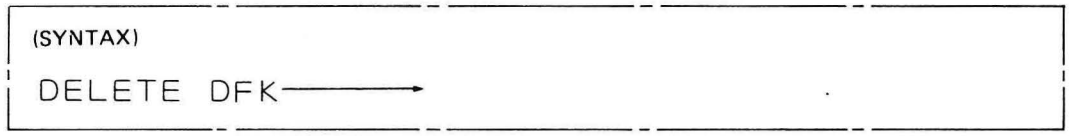
1. DELETE 100, 200 [ENTER] ..... Deletes lines from line number 100 to line number 200.
2. DELETE 100, 100 [ENTER]
   or
   100 [ENTER]
   ..... Deletes the line which has line number 100.

Note

Deletion of a program line is valid only when the mode change switch is set to PRO.
(If this command is executed in the OP mode, an error (ERROR 34) occurs.)

● **Function 2:** Deletes the definition of definable keys.

Format: DELETE DFK



Explanation:

1. Deletes the definition in all definable keys (DEF 1 to DEF 20).
2. After execution of this command, the definition in all definable keys are invalidated. Therefore, definable keys DEF1 to DEF20 must be redefined if they are needed. (For the definable key definition method, see I. 7. "Use of Definable Keys" (see page 58).)

ERASE

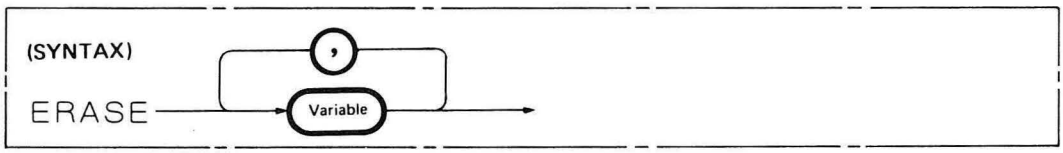
Erases variables according to the specification given.

〈Purpose of use〉
To use memory efficiently by erasing array variables and so on which will not be used any more.

● **Function 1:** Erases the specified variable.

Format: ERASE 〔N〕

N : Variable to be erased.



Explanation:

Erases variable N.

This command erases unwanted variables (especially, array variables) to use the memory space more efficiently.

Simple example:

To erase the array variable declared with DIM A (20, 20), specify the following:

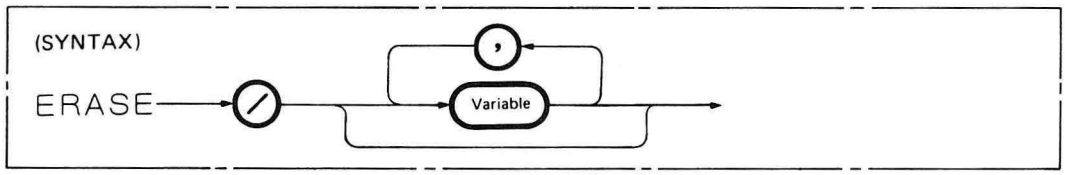
ERASE A(*) ENTER

A (*), in the above example, is called a collective array variable (see page 21).

● **Function 2:** Erases all variables apart from the specified variable.

Format: ERASE /〔〔N〕〕

N : Variable not to be erased



Explanation:

1. Erases all variables except variable N.
2. When 〔N〕 is omitted, all variables in the memory are erased.

Note

1. A variable in this case indicates the variable itself including the variable contents. (If an array variable is erased, also the preceding declaration for the array variable is invalidated.)
2. This command can also be used in a program.
   This command cannot be used in a subroutine (see page 20) or in a FOR/NEXT loop (see page 149).

AUTO

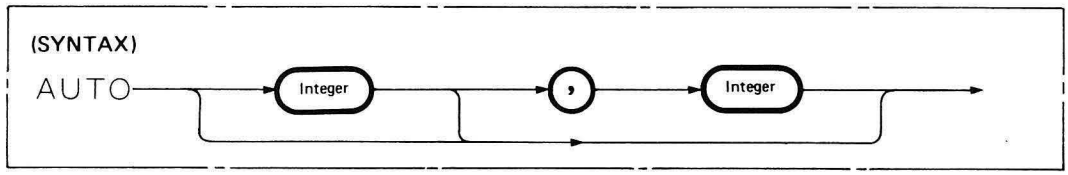
Automatically generates program line numbers.

〈Purpose of use〉
When the user wishes the programming to generate line numbers.

Format: AUTO [m [, n]]

m : Start line number

n : Line number increment



Explanation:

1. When line number m is displayed, input the statement, then press the [ENTER] key. The next line number with increment n is displayed automatically. Subsequently, the statement can be input in the same way. Each time the [ENTER] key is pressed, the line number is generated automatically. Input of the line numbers can be omitted using this command.
2. When increment specification n is omitted, 10 is set as the increment.
3. To clear the AUTO function, press the [CL] key; or delete the line number by pressing the [←] key, then specify the following:
   AUTO [ENTER] , or [CTRL] [HOME]
   (If [CTRL] and [HOME] keys are pressed, the display on the CRT is cleared.)

Simple example:

1. Operation		Line number generation sequence
AUTO 10	[ENTER]	10, 20, 30 .....
AUTO 5,5	[ENTER]	5, 10, 15 .....
AUTO 100	[ENTER]	100, 110, 120 ...
AUTO 100,20	[ENTER]	100, 120, 140 ...

2. AUTO 20,20 [ENTER]

If the above operation is performed, line number 20 is displayed and the cursor moves to the position following the line number.

If the statement is input and the [ENTER] key is pressed, line number 40 is displayed and the cursor moves to the position following this line number. If input of all program lines is completed at line number 1000, by repeating this operation, line number 1020 is displayed and the cursor moves to the position succeeding this line number. Erase the line number by pressing the [CL] key or by operating the [←] key, then input the following:

AUTO [ENTER]

Thus, the AUTO function is reset.

(The AUTO function can also be reset by pressing the [CTRL] and [HOME] keys.)

Note

1. Assume that the following program is generated by the operation of AUTO 20,20 [ENTER] :

```
20 A1=5:B1=10
40 C1=A1*B1*7
60 DISP A1,B1,C1
80 END
```

40 IC=A1*B1*7

If line number 40 is input erroneously as shown above, the next line number is displayed as421.

This is because line number 40 is considered to be 401 and 20 is added to this number.

40 IC=A1*B1*7
└─ This is considered to be 401.

To recover from this error, erase line number 421 by pressing the [CL] key or by operating the [←] key, then reinput line number 40.

Thus, 60 is displayed as the next line number by adding 20 to the line number 40.

If a digit is input erroneously following a line number, unexpected line number is generated as shown in this example.

Later, delete line number 401 by the following operation: 401 [ENTER]

REN

Changes program line numbers.

〈 Purposes of use 〉

① To arrange line numbers which are not in correct numberical order after program line insertion or deletion.

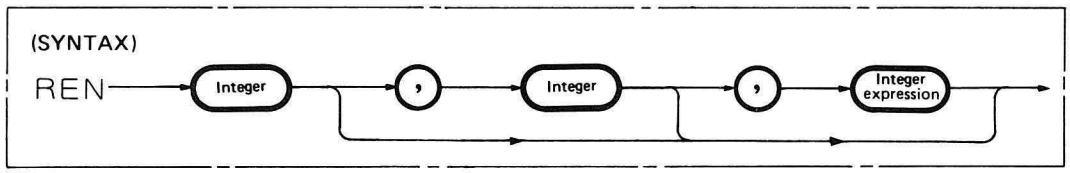
② When there are no unused line numbers between lines after insertion of program lines.

Format: REN ℓ [, m [, n]]

ℓ : First line number to be renumbered.

m : New line number

n : Line number increment



Explanation:

1. Renumbers the current lines beginning with line number ℓ to begin with line number m and having a line number increment by n.
   In this case, ℓ ≦ m must be satisfied. If ℓ > m and there are program lines before line number m, an error (ERROR 64) occurs.
   (If the smallest line number is ℓ, ℓ ≦ m must be satisfied.)
   If line number ℓ is not in the program, an error (ERROR 4) occurs.
2. When increment [, n] is omitted, the increment is set to 10.
3. When [, m, n] is omitted, the line numbers are renumbered to begin with ℓ and with an increment by 10.

Simple example:

1.

```
10 A=5:B=10
15 C=A*B+7
18 DISP A,B,C
20 END
```

If command REN 10,100,20 [ENTER] is executed for the above program, the line numbers are renumbered as follows:

```
100 A=5:B=10
120 C=A*B+7
140 DISP A,B,C
160 END
```

To check that the line numbers are renumbered, use the LIST command, which was explained earlier. (see page 85).

2.

```
10 A=1
15 DISP A^3,
20 A=A+1
21 IF A=2 THEN (10+10)
23 IF A<=20 THEN 15
25 END
```

If command REN 10 [ENTER] is executed for the above program, the line numbers are arranged as follows:

```
10 A=1
20 DISP A^3,
30 A=A+1
40 IF A=2 THEN (10+10)          ①
50 IF A<=20 THEN 20             ②
60 END
```

① ···· Not changed.
② ···· Changed.

Note

1. When this command is executed, the jump destination line number of the GO TO, GO SUB, THEN, ELSE, ON . . . GO TO, ON . . . GO SUB, RESTORE, and USING statements is changed.
   If the jump destination is specified in an numeric expression, the jump destination line number remains unchanged. (see Simple example 2.)
2. The line numbers specified in the TR, TR STOP, or LIST command are not changed.
3. This command is valid only when the mode change switch is set to PRO. (If this command is executed in the OP mode, an error (ERROR 34) occurs.)

TR

Traces *) the program.

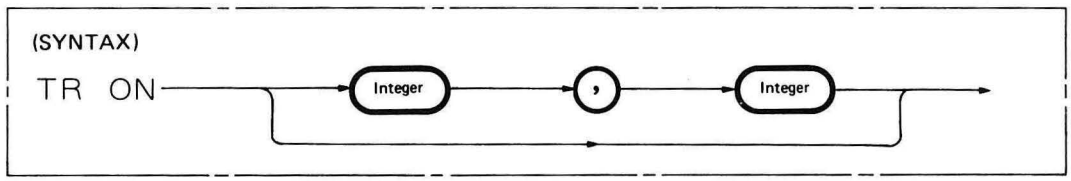
〈Purpose of use〉
When debugging the program to check the program flow and the execution results line by line.

● **Function 1:** Turns on the trace function for the CRT display.

Format: TR ON [m, n]

m : Start line number

n : End line number



Explanation:

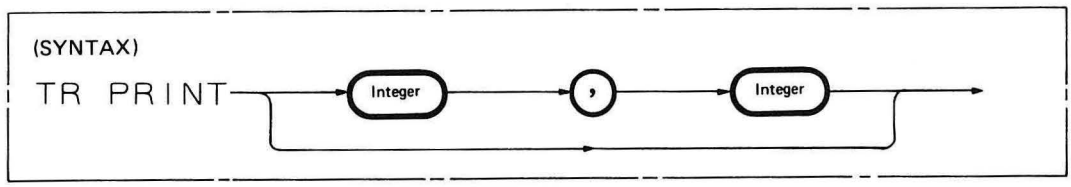
1. The trace function is displayed on the CRT for line number m to n ($m \leqq n$).
2. If a RUN command (see page 78) is input following this command, the program is executed while displaying the currently running program line contents on the CRT. When m and n are omitted, this trace function is set on for all line numbers.
3. To trace individual lines, set the trace function on by inputting TR ON [m, n] then press the DEB key (this key stores the RUN/ command) while the CTRL key is pressed. Subsequently, each time the DEB key (this key stores the CONT/ command) is pressed, the trace function is executed for individual lines.
4. Even if a line number not existing in the program is input, the trace function is set on. (For example, if TR ON 25, 75 is executed for a program which has lines 10 to 200 with an increment by 10, the trace function is executed for the lines which have line numbers 30 to 70.)
5. If the input line number m is greater than line number n, this command is invalidated.

● **Function 2:** Turns on the trace function for the printer.

Format: TR PRINT [m, n]

m : Start line number

n : End line number



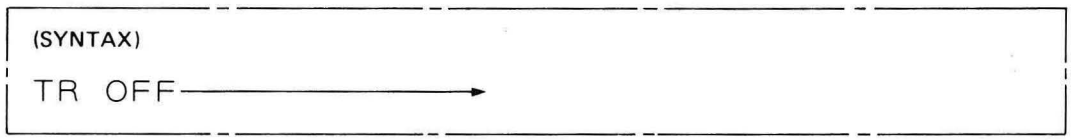
*) Trace: To observe the program flow and execution results.

Explanation:

Sets the printer trace function for line numbers m to n ($m \leqq n$).
See Function 1 for the details.

● **Function 3:** Turns off the trace function.

Format: TR OFF



Explanation:

1. Turns off the trace function.
2. If the trace function is set on, it remains active. To turn this function off, input this command before executing the next program.

Note

The trace function cannot be turned on simultaneously for both the CRT and printer. Only the last TR command input is valid.

TR STOP

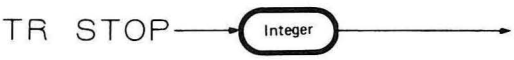
Stops program execution at the specified line number.

〈Purpose of use〉
① When checking the valiables state during calculation.
② When assigning a value to a variable during calculation.

Format: TR STOP n

n : Line number

(SYNTAX)



Explanation:

1. Stops execution of the program before executing the line which has line number n. To continue execution of the program, use a CONT command (see page 82).
2. To reset the stop function, use Function 3 of the TR command (see page 100).

Simple example:

```
    ⋮
250 A=A+1
    ⋮
```

Stop execution of the above program before line number 250, check the contents of variable A, then change the A value according to the result in the following procedure:
Specify the following before execution of the program:
TR STOP 250 [ENTER]
The execution of the program is stopped before line number 250.
If DISP A[ENTER] is input in this case, the A value is displayed on the CRT.
To change the A value to 100, input the following:
A=100 [ENTER]
To continue execution of the program from line number 250, press the CONT key or input the following:
CONT [ENTER]

Note

1. This command has no trace function.
   The trace and stop functions are activated by inputting both the TR and TR STOP commands.
   When the TR OFF command is executed, both the trace and stop functions are reset.
2. If a line number not existing in the program is input, this command is invalidated.
3. Unlike a STOP statement (see page 126), this command enables the program to be stopped at the specified program line manually.

3. STANDARD STATEMENTS

The standard statements are basic statements used in programming.

There are the following standard statements:

LET ※
DISP ※
PRINT ※
IMAGE
INPUT ※
DATA
READ
RESTORE
REM ※
STOP
END ※
DIM ※
GO TO
GO SUB
RETURN
ON……GO TO
ON……GO SUB
IF……THEN
IF…… : ELSE
FOR……TO……STEP／NEXT

The statements marked with ※ can also be used in manual operation.

LET

Assigns a certain value (numeric expression or character string.) to a variable.

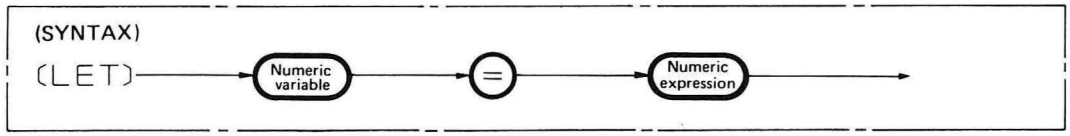
〈Purpose of use〉
① When strong numeric values or calculation results by assigning them to a variable.
② When strong a character by assigning it to a variable.

● **Function 1:** Assigns a numeric expression value.

Format: 〔LET〕 A=X

A : Numeric variable

X : Numeric expression indicating numeric data



Explanation:

1. Assigns the value of numeric expression X to numeric variable A.
   An equal sign (=) in this statement means that the value of the numeric expression on the right side is assigned to the numeric variable on the left side.
   Note that this sign does not mean that the values on the right and left sides are the same.
   The value is assigned from the right side to the left side. Therefore, write a variable on the left side and the numeric value or numeric expression on the right side.
2. In this statement, LET can be omitted. Normally, LET is omitted.
   For example, LET A = 10 is equivalent to A = 10.
3. This statement can also be used manually.

Simple example:

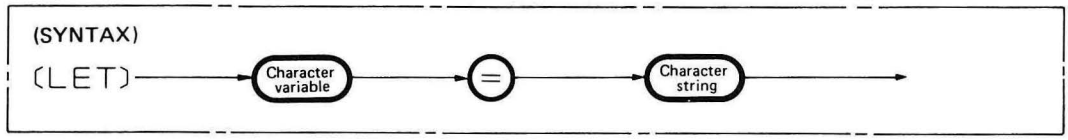
1. `10 LET A=3.14` ..... Assigns numeric constant 3.14 to numeric variable A.
2. `10 B=B+1` ..... Adds 1 to the value of numeric variable B, then assigns the sum to numeric variable B.
   (In this example, LET is omitted.)
3. `10 M=(X/Y-5)` ..... Assigns the value of the expression in the parentheses to numeric variable M.
   (In this example, LET is omitted.)
4. `N=SIN 30` [ENTER] ..... Assigns the value of sin 30 to numeric variable N.
   (In this example, LET is omitted in manual operation.)

● **Function 2:** Assigns a character string.

Format: 〔LET〕 **A$=C**

A$: Character variable

C : Character string indicating character data



Explanation:

Assigns character string C to character variable A$.

As in Function 1, the equals sign (=) means that the character string on the right side is assigned to the character variable on the left side. For details, see Function 1.

Simple example:

1. `10 LET A$="DISPLAY"` ..... Assigns character constant "DISPLAY" to character variable A$.
2. `10 NAME$="BASIC MANUAL"` ..... Assigns character constant "BASIC MANUAL" to character variable NAME $.
3. `10 M$=CHR$ 65` ..... Assigns character "A" of ASCII code (decimal) 65 to character variable M$.
4. `P$="BROWN"` [ENTER] ..... Assigns a character constant "BROWN" to character variable P$.
   (In this example, LET is omitted in manual operation.)

Note

The type of the variable must match that of the assigned data. That is, only a numeric value can be assigned to a numeric variable and only characters can be assigned to a character variable.

Application example:

[Program A] Performs addition and subtraction.

```
10 A=10 ..... Assigns 10 to numeric variable A.
20 B=20 ..... Assigns 20 to numeric variable B.
30 C=A+B ..... Assigns the value of A + B (30 in this case) to numeric variable C.
40 D=A-B ..... Assigns the value of A - B ( -10 in this case) to numeric variable D.
50 DISP C ..... Displays the value of numeric variable C.
60 DISP D ..... Displays the value of numeric variable D.
70 END ..... Ends the program.
```

[Program B] Adds (combines) character strings.

```
10 A$="I am"          ·····Assigns "I am" to character variable A$.
20 B$=" a boy."       ·····Assigns "a boy." to character variable B$.
30 C$=A$+B$           ·····Assigns A$ + B$ ("I am a boy.") to character variable C$.
40 DISP C$            ·····Displays the contents of character variable C$ on the CRT.
50 END                ·····Ends the program.
```

Character strings can be only added, they cannot be subtracted.

DISP

Outputs numeric values and characters to the CRT.

〈Purpose of use〉
To display the calculation results and numeric or character variable contents on the CRT.

● **Function 1:** Displays a numeric expression value or character string on the CRT or moves the cursor.

Format: **DISP** 〔{{**X** | **C** | **TAB Y**}} 〔; | , 〕〕

X : Numeric expression indicating the numeric value to be displayed

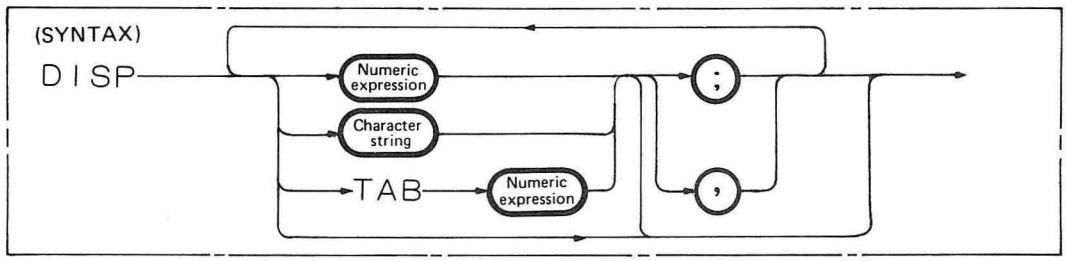
C : Character string to be displayed

Y : Cursor position (Numeric expression)

$0 \leqq Y \leqq 79$ in the 80-column mode

$0 \leqq Y \leqq 39$ in the 40-column mode

(For details, see DISP CLEAR statement (page 214).)



Explanation:

1. Displays numeric expression X or character string C; or moves the cursor to the right up to the position of the numeric expression given by Y (the CRT left margin is 0).
   If a semicolon (;) is used for separation, no spaces are output between operands.
   If a comma (,) is used for separation, the cursor automatically moves to the beginning of the next block (20-character unit) before displaying the next item. When displaying a long item covering the next block, the next item is displayed in the next block.
   When displaying a plus numeric value or 0, this value is preceded and followed by a space. When displaying a minus numeric value, this value is followed by a space.
   Characters are output as specified.
2. TAB is a command that moves the cursor from the CRT left margin to the right by the value of the numeric expression.
3. If this statement is not followed by parameters (the numeric expression value, character string, and cursor position, etc. are all omitted), this statement specifies a line feed.

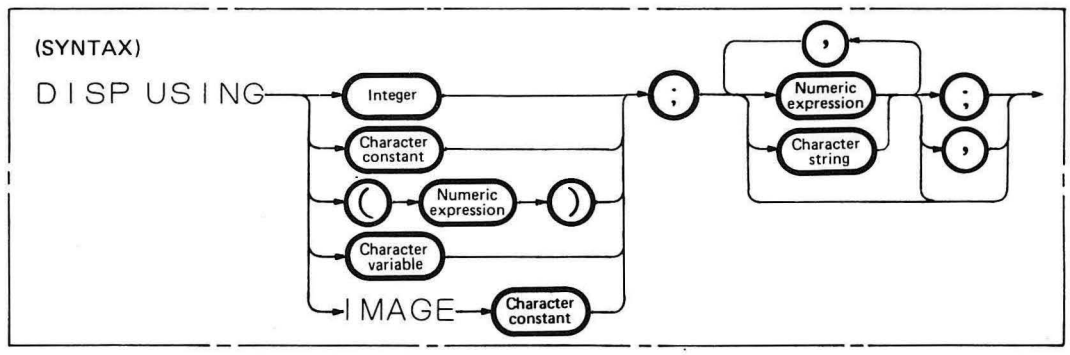
Simple example:

1. `10 DISP A,B` ..... Displays the contents of numeric variable A, then displays the contents of numeric variable B from the beginning of the next block.
2. `10 DISP "ANSWER"` ..... Since this statement ends with a comma (,), this statement displays ANSWER, then positions the cursor after moving one block forward.
3. `10 DISP TAB A*2;` ..... Since this statement ends with a semi colon (;), this statement moves the cursor from the CRT left margin to the position given by the value of A * 2, then remains there. (If A is 10, positions the cursor at the 21st character position from the CRT left margin.)
4. `10 DISP A,"+",B;"=",C` ..... Displays the contents of numeric variable A, displays + at the beginning of the next block, displays the contents of numeric variable B from the beginning of the next block, displays = following this value, then displays the contents of numeric variable C from the beginning of the next block.

● **Function 2:** Displays a numeric expression value or character string in the specified format.

Format: **DISP USING n | S | (Z) | A$ | IMAGE T ; [{X | C}] [; | ,]**

- n : Line number value of IMAGE statement
- S : Line label name of IMAGE statement
- Z : Line number (numeric expression) of IMAGE statement
- A$: Character variable for specified format (image symbol)
- T : Character constant indicating image symbol
- X : Numeric value (numeric expression) to be output
- C : Character string to be output



Explanation:

Displays the numeric expression value or character string in the format specified in an IMAGE statement (see page 113).

This statement outputs the numeric expression value or character string given after the semicolon (;) in the format specified in the IMAGE statement of the line indicated by line number n, line label name S, or the numeric expression value Z (fractions are truncated); the format specified in character variable A$ (image symbol); or the format specified by the IMAGE statement.

For details, see the IMAGE statement (page 113).

Note

If this statement when used in a program ends with a semicolon (;), the cursor remains at the current position until the next CRT display statement is executed or the program execution ends (the END statement is executed).

If this statement ends with a comma (,), the cursor remains at the position after advancing one block (20-character unit) forward.

Application example:

Be careful when using the semicolon (;), TAB, and comma (,) functions.

```
10 INPUT "A=";A,"B=";B ··· Reads data input from the keyboard into numeric variables A and B.
20 C=A*B ····· Assigns the value of A x B to numeric variable C.
30 DISP A;TAB 11;"*",B;"=";C ····· Displays the execution results.
40 END
```

(Execution results) When 3 is input to numeric expression A, and 4 is input to numeric expression B from the keyboard, the display is as follows:

```
>RUN
A=3
 3                *          B=4
READY                       4 = 12
>
```

11 columns

20 columns (1 block)

PRINT

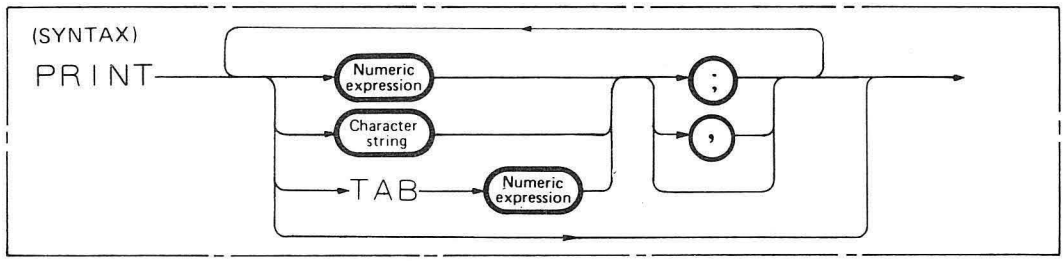
Outputs numeric values or characters to a printer.

〈Purpose of use〉
To output calculation results or contents of a numeric or character variable to a printer.

● **Function 1:** Outputs numeric expression values or character strings to a printer or moves the pointer*).

Format: PRINT [{{X | C | TAB Y}} [; | ,]]

X: Numeric expression value to be output
C: Character string to be output
Y: Pointer position (numeric expression)



Explanation:

1. Outputs numeric expression X or character string C, or moves the pointer to the right up to the position of numeric expression Y value (the printer left margin is 0).
2. For the output format used when a semicolon (;) or comma (,) is used for separation, see the DISP statement (page 107).
3. If this statement is not followed by parameters (the expression value, character string, and pointer position are all omitted), this statement specifies a line feed.

Simple example:

1. `10 PRINT A,B` ..... Outputs the contents of numeric variable A, then outputs the contents of numeric variable B from the beginning of the next block.
2. `10 PRINT "ANSWER",` ..... Since this statement ends with a comma, this statement outputs ANSWER, then positions the pointer after moving one block forward.
3. `10 PRINT TAB A*2;` ..... Since this statement ends with a semicolon (;), moves the pointer from the printer left margin to the position of the A * 2 value, and remains there. (If A is 5, the pointer is positions at the 11th character position from the printer left margin.)

*) Pointer: Print position (printer head position).

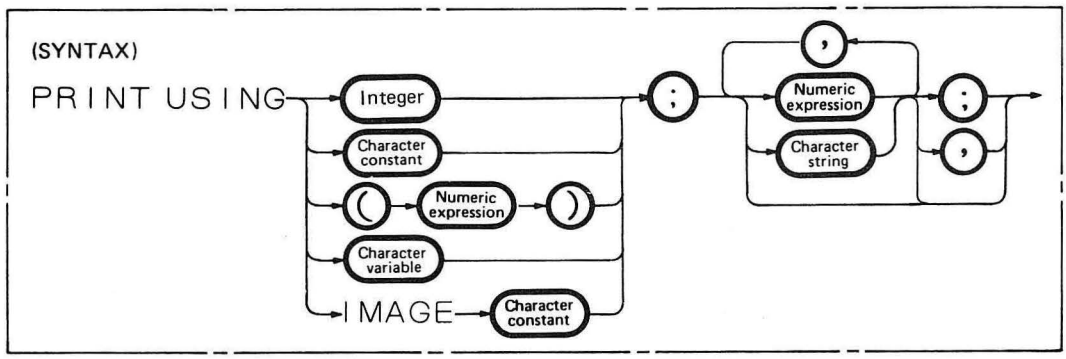
4. `10 PRINT A,TAB 30;B$,C`

..... Outputs the contents of numeric variable A, moves the pointer to the 31st character position from the printer left margin, outputs the contents of character variable B$, then outputs the contents of numeric variable C from the beginning of the next block.

● **Function 2:** Outputs the numeric expression value or character string to a printer in the specified format.

Format: **PRINT USING n | S | (Z) | A$ | IMAGE T ; [{X | C}] [; | ,]**

n : Line number of IMAGE statement
S : Line label name of IMAGE statement
Z : Line number (numeric expression) of IMAGE statement
A$: Character variable of specified format (image symbol)
T : Character constant indicating image symbol
X : Numeric expression to be output
C : Character string to be output



Explanation:

Outputs the expression or character string in the format specified in an IMAGE statement (see page 113). This statement outputs the numeric expression value or character string given after the semicolon (;) by the format specified in the IMAGE statement of the line indicated by line number n, line label name S, or the numeric expression value Z (fractions are truncated); the format specified in character variable A$ (image symbol), or the format specified by the IMAGE statement.

For details, see the IMAGE statement (page 113).

Note

1. If this statement when used in a program ends with a semicolon (;), the pointer remains at the current position until the next printer output statement is executed, or when the program execution ends (the END statement is executed). If this statement ends with a comma (,), the pointer is moved to the next block, and remains there (20-character unit).
2. If this statement is executed when a printer is not connected to this system, or when paper is not set in the printer, an error (ERROR 25) occurs.

Application example:

Denomination calculation program

```
 10 PRINT "***** DENOMINATION CALCULATION *****"
 20 DIM S(12)
 30 S(1)=100:S(2)=50:S(3)=20:S(4)=10
 40 S(5)=5:S(6)=2:S(7)=1:S(8)=0.5
 50 S(9)=0.25:S(10)=0.1:S(11)=0.05:S(12)=0.01
 60 INPUT "AMOUNT";A
 70 PRINT "***** "AMOUNT =";A
 80 FOR J=1 TO 12
 90 B=INT (A/S(J))
100 PRINT "$";S(J);TAB 13;B
110 A=A-S(J)*B
120 NEXT J
130 PRINT "****************************************"
140 END
```

(Execution results) When 1638.28 is input as AMOUNT in the above example, the CRT displays the following:

```
>RUN
AMOUNT =1638.28
READY
>
```

The print output is as follows:

```
***** DENOMINATION CALCULATION *****
***** AMOUNT =  1638.28
$ 100          16
$ 50           0
$ 20           1
$ 10           1
$ 5            1
$ 2            1
$ 1            1
$ 0.5          0
$ 0.25         1
$ 0.1          0
$ 0.05         0
$ 0.01         3
****************************************
```

IMAGE

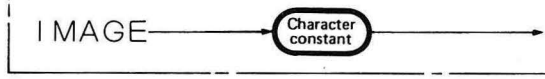
Specifies the output format with an image symbol.

〈 Purpose of use 〉
When outputting an expression value or character string in an edited format.

Format: I MAGE S

S: Image symbol specifying the output area or format

(SYNTAX)



Explanation:

Image symbol S is specified as a character constant. This image symbol specifies the output numeric value, character area, or format. See Function 2 of the DISP statement (page 107) and PRINT statement (page 110).

Image symbols are classified into those related to variables (edited according to the variable contents) and those not related to variables (used independently of variable contents).

(Image symbols related to variables)

Type	Image symbol	Repetition	Function
Character format	A	yes	Specifies the character position.
Numeric value format	#	no	Specifies the digits of a numeric value. The leading 0s are replaced by spaces with zero-suppression*).
	Z	no	Specifies the digits of a numeric value. The leading 0s are output without zero-suppression.
	*	no	Specifies the digits of a numeric value. The first 0s are output as *'s without zero-suppression.
	.	no	Specifies the decimal point position for separation between the integer and fraction parts.
	,	no	Specifies the 3-digit partitioning (for integer part only).
	E	no	Specifies the exponent format.
	+	no	Specifies the sign display. Outputs + for a plus value and outputs - for a minus value.
	–	no	Specifies the sign display. Outputs a space for a plus value and outputs - for a minus value.
	$	no	Specifies outputting a dollar sign ($) preceding a numeric value.
Others	%	no	Specifies outputting the character or symbol specified following the percent symbol (%), the same number of times as the number of related variable values.

*) Zero suppression: This is to delete meaningless 0s such as those on the left side of the most significant digit in the integer part, and those on the right side of the least significant digit in the fraction part.

(Image symbols not related to variables)

Type	Image symbol	Repe-tition	Function
Space	X	yes	Specifies a space.
Character constant	" ..... "	yes	Specifies a character string enclosed in quotation marks (" or ').
Carriage return	L	yes	Specifies carriage return and line feed.
For peripheral equipment	N	yes	NULL code (&00): Null
	BL	yes	BEL code (&07): Bell
	BS	yes	BS code (&08): Backspace
	T	yes	TAB code (&09): Tabulation
	FF	yes	FF code (&0C): Form feed
	C	yes	CR code (&0D): Carriage return
	SO	yes	SO code (&0E): Shift-out
	SI	yes	SI code (&0F): Shift-in
	CN	yes	CAN code (&18): Cancel
	ES	yes	ESC code (&1B): Escape

Note

For the functions of the image symbols for peripheral equipment, refer to the instruction manual for each item of peripheral equipment.

(Specification method)

i) Repetition

For an image symbol which has the "yes" mark in the repetition field in the table, the symbol can be repeated the number of times by given the value preceding this symbol. (For example, "3A" is equivalent to " AAA".)

ii) Specifying image symbols of two or more items in a character constant

When writing image symbols of two or more items in a character constant, one or more spaces are necessary between image symbols.

Simple example:

```
10 IMAGE "10A 2X $###,###"
```

..... Specifies outputting 10 digits for the first variable, outputting two spaces, then outputting the second variable value with a dollar sign at the beginning, suppressing the first 0s (0s are replaced by spaces), and positioning 3 digits.

Note

1. Unless image symbol + or - is specified in the numeric value format, a minus sign is not assigned even for a minus value output. If the variable is not of the exponent format, + or - can be specified before or after the numeric value.

```
10 A=-1234
20 PRINT USING IMAGE "##,###";A
30 PRINT USING IMAGE "-##,###";A
40 PRINT USING IMAGE "+##,###";A
50 PRINT USING IMAGE "ZZ,ZZZ+";A
60 PRINT USING IMAGE "ZZ,ZZZ-";A
70 END
```

See Function 2 of the PRINT statement (page 110).

(Print output)

```
 1,234
 -1,234
 -1,234
01,234-
01,234-
```

2. If the character constant in the IMAGE statement is enclosed in quotation marks (" "), the character string in this character constant must be enclosed in quotation marks (" "), or vice versa.
   IMAGE "20A 'ADDRESS'" or IMAGE '20A "ADDRESS"' must be specified.

Application example:

[Program A]

```
10 A$="BROWN":B=158200
20 PRINT USING 30;A$,B:END
30 IMAGE '"NAME" 3X 5A 4X $#,###,###'
40 END
```

(Execution results)

```
NAME  BROWN      $158,200
```

[Program B]

```
 10 FOR A=1 TO 5
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "10X -##Z.##"
110 DATA 123,-1.256,0,-987.1,.05
120 END
```

(Execution results)

```
>RUN
              123.00
               -1.25 ..... The third digit after the decimal point is truncated.
                0.00
             -987.10
                0.05
READY
>
```

[Program C]

```
 10 FOR A=1 TO 3
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "5X +#,***.**E02"
110 DATA 123456,2E4,-25
120 END
```

(Execution results)

```
>RUN
     +1,234.56E+02
       +200.00E+02
       -**0.25E+02
READY
>
```

[Program D]

```
 10 FOR A=1 TO 3
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "5X -#,###.## L"
110 DATA 123,6E3,-25.11
120 END
```

(Execution results)

```
>RUN
         123.00
                ..... There is a space of one line.
       6,000.00

         -25.11

READY
>
```

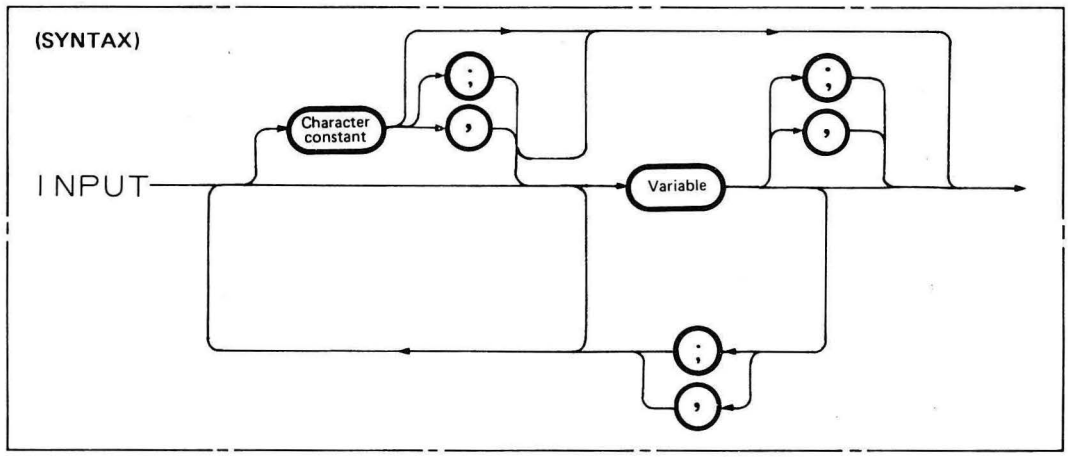
INPUT

Inputs numeric values or characters with the keyboard.

〈Purpose of use〉
To input data with the keyboard during program execution.

Format: INPUT [S ; | ,] M [; | , {{T | N}}] [; | ,]

S: Display character
M: Variable for assigning input data
T: Display character
N: Variable for assigning input data



Explanation:

1. Displays character variable S and waits for input with the keyboard. When data is input with the keyboard, this data is assigned to variable M.
   The cursor is at the position succeeding character constant S when a semicolon (;) is used for separation and at the 21st character position (at the beginning of the next block) from the beginning of character constant S when a comma (,) is used for separation.
2. If a semicolon (;) is used for separation when repeating a character constant or variable, data is displayed without spaces; if a comma (,) is used, data is displayed in block units.
3. If this statement is used in a program and ends with a semicolon (;) or comma (,), see notes for a DISP statement. (See page 107).

Simple example:

```
10 INPUT A ·····Awaits input with the keyboard and assigns the input data to numeric variable A.
```

2. 10 INPUT "WHAT IS YOUR WEIGHT?";B

..... Displays WHAT IS YOUR WEIGHT? on the CRT, awaits input with the keyboard, and assigns the input data to numeric variable B.

3. 10 INPUT "WHAT IS YOUR WEIGHT?";B;"kg"

..... Performs processing of Example 2 above, then displays kg following the data input. For example, when 50 is input with the keyboard, the following is displayed: WHAT IS YOUR WEIGHT? 50 kg

4. 10 INPUT "A=";A,"B=";B ..... Displays A= on the CRT, awaits input with the keyboard, assigns the input data to numeric variable A, displays B= from the next block, and awaits input; when data is input, assigns the input data to numeric variable B.

Note

1. When inputting data with the keyboard, it is not necessary to enclose the input characters in quotation marks (" or ').
2. If the [ENTER] key is pressed without inputting data with the keyboard and no data has been assigned to the variable, 0 is assigned to a numeric variable and a null string (" ") is assigned to a character variable. If a value has already been assigned to the variable, this value is input again.
3. If the type of the input data is not the same as that of the variable assigning this data, an error (ERROR 21) occurs. A key input error can be cleared with the [CL] or [↑] key.

Application example:

[Program A] Performs addition and subtraction on input data.

```
10 INPUT "A=";A ..... Displays A= on the CRT and assigns input data to numeric variable A.
20 INPUT "B=";B ..... Displays B= on the CRT and assigns input data to numeric variable B.
30 C=A+B ..... Assigns the value of A + B to numeric variable C.
40 D=A*B ..... Assigns the value of A x B to numeric variable D.
50 DISP "A+B=";C ... Displays the addition result on the CRT in the format A + B = C.
60 DISP "A*B=";D ... Displays the multiplication result on the CRT in the format A * B = D.
70 END
```

(Execution results) To assign 10 to A and 20 to B, specify 10 [ENTER] , then 20 [ENTER] .

```
>RUN
A=10
B=20
A+B= 30
A*B= 200
READY
>
```

[Program B] Performs the same calculation as that of program A, but the display is different.

```
10 INPUT "A=";A,"B=";B,
20 C=A+B:D=A*B
30 DISP "A+B=";C,"A*B=";D
40 END
```

When data is input to numeric variable B, the cursor moves to the beginning of the next block.

(Execution results) Assign 10 to A, and 20 to B.

```
>RUN
A=10          B=20          A+B= 30          A*B= 200
READY
>
```

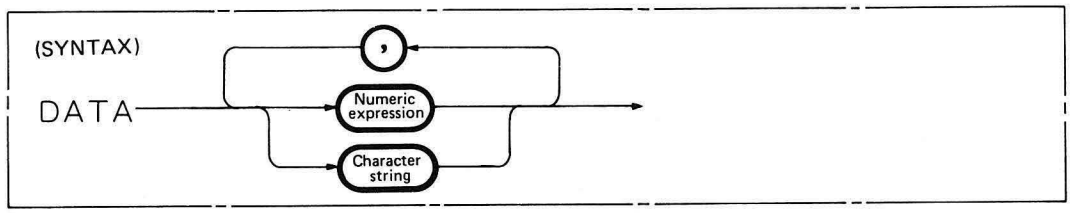
DATA

Assigns data to a variable in the READ statement.

〈Purpose of use〉
When using much fixed data in the program.

Format: DATA 〔X | C〕

X: Numeric data
C: Character data



Explanation:

1. This statement is related to the READ statement explained later.
   When the READ statement is executed, data in this statement is assigned to the specified variable.
2. More than one DATA statement can be used in a program. Even if data is specified separately in the program in such cases, data is assumed to be continuous.
3. This statement can be specified arbitrarily regardless of the READ statement position in the program.

Simple example:

1. `10 DATA 12,345,6.7` ..... Specifies numeric constants 12, 345 and 6.7 as data.
2. `10 DATA "ABC","XYZ"` ..... Specifies character constants "ABC" and "XYZ" as data.
3. `10 DATA 12,"SHARP",34.5` ..... Specifies numeric constant 12, character constant "SHARP", and numeric constant 34.5 as data.

Application example:

Displays the menu on the CRT.

```
 10 READ A$,B$
 20 DISP A$,B$
 30 END
100 DATA "COFFEE","$1.00"
```

In this example, COFFEE is assigned to character variable A$, and $1.00 is assigned to character variable B$; they are displayed on the CRT.
For details, see the READ statement (next page).

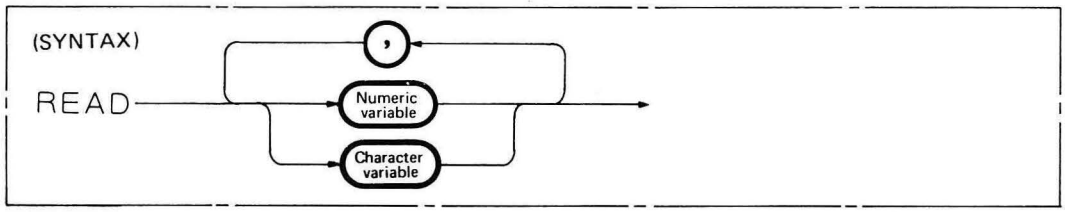
READ

Reads data from a DATA statement and assigns the value to the specified variable.

Format: READ {A | A$}

A : Numeric variable

A$: Character variable



Explanation:

1. This statement is used in combination with a data statement explained before.
   When the first READ statement is executed, reading starts with the first item of data in the DATA statement which has the lowest line number, and assigns the read data to the variables in sequence. When reading of data in the first DATA statement is completed, the second DATA statement is assigned to the variable sequentially.
   If a line number is specified in the RESTORE statement explained later, data can be read from the DATA statement at an arbitrary position.
2. When the reading of data is complet, the location of the last data already read from the DATA statement is in memory. When the next READ statement is executed, reading starts from the data item following the last data item read in the preceding read operation.

Simple example:

1. `10 READ A,B,C` ..... Assigns the data of the DATA statement sequentially to numeric variables A, B, and C.
2. `10 READ A$,B$,C` ..... Assigns the data of the DATA statement sequentially to character variable A$, character variable B$, and numeric variable C.

Note

1. The type of the variable in the READ statement must be the same as that of the data read from the DATA statement.
2. If there is no data to be read, an error (ERROR 2) occurs. If data remains in the DATA statement without being read, no errors occur.

Application example:

Display the menu on the CRT.

```
10 FOR I=1 TO 5 ··· In the FOR . . . NEXT loop, increments variable I by 1.
20 READ A$,B$ ····· Reads data from DATA, then assigns the data to variables A$ and B$.
30 DISP A$,B$ ····· Displays the contents of variables A$ and B$ on the CRT.
40 NEXT I
50 END
60 DATA "DRINK","PRICE","COFFEE","$2.50",
   "MILK","$2.00"
70 DATA "COLA","$1.50","JUICE","$3.00"
```

(Execution results)

```
 RUN
DRINK                PRICE
COFFEE               $2.50
MILK                 $2.00
COLA                 $1.50
JUICE                $3.00
READY
```

RESTORE

Specifies the position of the DATA statement ralated to a READ statement.

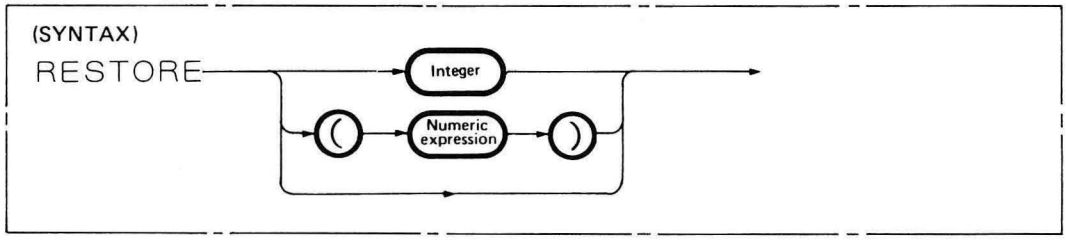
〈Purpose of use〉

When reading in a DATA statement (see page 120) with a READ statement (see page 121).

Format: RESTORE [n | (X)]

n: Line number

X: Line number (numeric expression)



Explanation:

1. The DATA statement in the line of line number n or numeric expression X value (the fractions are truncated) relates the DATA statement associated with the first READ statement after this statement.
   That is, READ statements after this statement start reading data from the DATA statement of the specified line.
2. When the line number is omitted, the DATA statement which has the lowest line number of all DATA statements is related to the first READ statement after this statement.

Simple example:

1. `10 RESTORE 100` ..... Specifies the DATA statement which has line number 100.
2. `10 RESTORE (500/3)` ..... Specifies the DATA statement which has line number 166.
3. `10 RESTORE` ..... Specifies the DATA statement which has the lowest line number.

Application example:

[Program A]

```
 10 READ A$,B,C$
 20 RESTORE 110
 30 READ D
 40 DISP A$,B,C$,D
100 DATA "NOTE"
110 DATA 200
120 DATA "BOOK"
130 END
```

Reads "NOTE", 200, and "BOOK".

Reads 200.

(Execution results)

```
>RUN
NOTE          200           BOOK          200
READY
>
```

[Program B]

```
 10 READ A,B,C
 20 DISP A,B,C
 30 READ X$,Y$
 40 DISP X$,Y$
 50 RESTORE 210
 60 READ V$
 70 DISP V$
 80 RESTORE
 90 READ D,E
100 DISP D,E
110 END
200 DATA 100,200,300
210 DATA "MAIN","SUB"
```

Reads 100, 200, and 300.

Reads "MAIN" and "SUB".

Reads "MAIN".

Reads 100 and 200.

(Execution results)

```
>RUN
 100                 200                 300
MAIN                SUB
MAIN
 100                 200
READY
>
```

REM

Writes a comment in a program.

〈Purpose of use〉

To write a comment in a program in order to make the program list easy to read and understand.

Format: REM

(SYNTAX)



Explanation:

This statement writes a comment in the program list.

An arbitrary number of comments can be written at arbitrary positions in the program. If a REM statement is executed in the program, no operation is performed for it, and the next line is executed. Therefore, a multistatement (two or more statements in a line) succeeding a REM statement is not executed. A mark ! can be used instead of REM. If ! is used, this must be written following immediate the line number. When writing a comment in a multistatement, REM must be used.

Simple example:

```
10 REM***PAYROLL PROGRAM*** ..... Writes a comment indicating that this is a PAY ROLL program.
(This statement is equivalent to 10!***PAYROLL PROGRAM***)
10 INPUT A:REM UNIT PRICE ..... Writes a comment indicating that A is a unit price.
```

Note

This statement cannot be used as follows:

```
10 REM "QUANTITY":INPUT A
```

In this case, INPUT A is assumed to be a comment and is therefore not executed.

```
10 INPUT A:! "QUANTITY"
```

! is valid only following immediate the line number. In this case, a syntax error occurs.

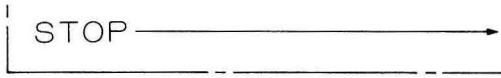
STOP

Stops execution of a program.

〈 Purpose of use 〉
To check the variable status and assign a value to a variable during execution of a program.

Format: STOP

(SYNTAX)



Explanation:

Temporarily stops execution of a program and places it in the input-awaiting state (the prompt symbol is displayed on the screen and the cursor blinks). The program stopping is released and execution restarts when a CONT command is executed (by pressing the CONT key).

See the CONT command (page 82) for details. Manual operations are possible in the stop status. If the program is modified, the program cannot be restarted with a CONT command. One or more STOP statements can be used in a program.

Application example:

```
10 INPUT A,B,C
20 X=5*A-B
30 Y=C/4-8
40 DISP Y
50 STOP ..... When Y is 0, and error occurs at the program line number 60.
60 Z=X/Y      Therefore, stop program execution temporarily at line number 50
70 DISP Z     and check the contents of Y.
80 END
```

END

Ends execution of a program.

〈Purpose of use〉
When ending execution of a program.

Format: END

(SYNTAX)



Explanation:

Ends execution of a program. This statement must be written at the end of program execution. (If an END statement is not written at the end of execution, an error (ERROR 54) occurs.)

Simple example:

```
100 END
```
..... Ends execution of the program at line 100.

Application example:

Write a program that calculates the sum of the integers 1 to A.

```
10 INPUT A
20 B=0
30 FOR I=1 TO A
40 B=B+I
50 NEXT I
60 DISP B
70 END
```

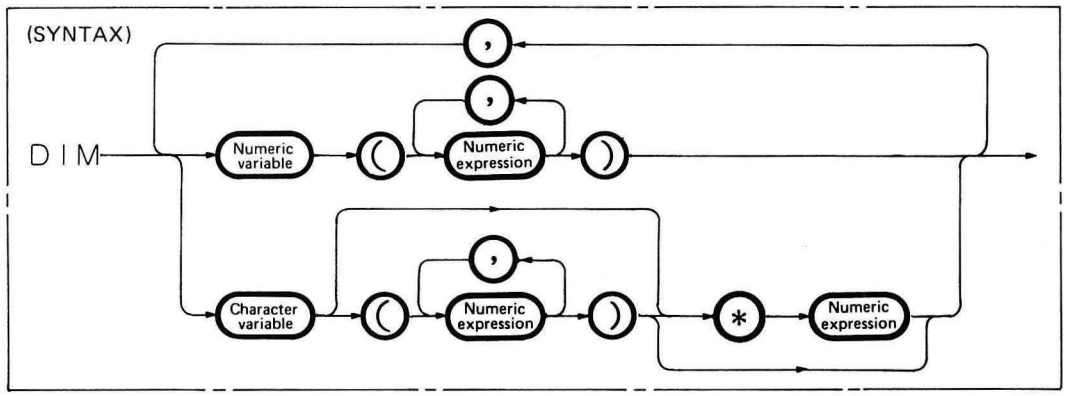
DIM

Declares an array variable and reserves the necessary memory area.

〈Purpose of use〉
When processing a set of data using a variable.

Format: DIM {A({X}) | A$({Y}) {*Z}}

A : Numeric array variable name
X : Number of array elements
A$: Character array variable name
Y : Number of array elements
Z : Array element size (byte count = number of characters that to be assigned)



Explanation:

1. When declaring a numeric array variable (select A ({X})), use A as the numeric array variable name and the value of numeric expression X (truncate the fractions) as the number of array elements.
2. When declaring a character array variable (select A$ ({Y}) [*Z]), use A$ as the character array variable name, the value of numeric expression Y (truncate the fractions) as the number of array elements, and Z bytes (truncate the fractions) as the size of an array element. When numeric expression Y is omitted, the array element size is set to 16 bytes.
   A character uses one byte. Therefore, when numeric expression Y is omitted, 16 characters can be assigned.
   For a @-type character variable including a simple variable, the number of assignable characters can be specified. (See Simple examples.)
3. If a numeric array variable is declared, all elements are initialized to 0s; if a character array variable is specified, all elements are initialized to NULL .

4. An array variable which has one numeric expression such as A (10) or A$ (10) is called a 1-dimensional array variable. If two or more numeric expressions are repeated by separating them with a comma (,), these array variables are called a multi-dimensional array variable. (For example, A (2, 3) is a 2-dimensional array variable and A (2, 3, 4) is a 3-dimensional array variable.)
   The number of elements in a multi-dimensional array variable is the product of the numeric expression values separated by commas (,). (For example, array variable A (2, 3) has 2 * 3 = 6 elements, and array variable A (2, 3, 4) has 2 * 3 * 4 = 24 elements.)
5. The elements are arranged sequentially from the element which has the lowest subscript value.
   For example, the array variables declared with DIM A(3) are arranged in the order A (1), A (2), and A (3), and those declared with DIM A (2, 3) are arranged in the order of A (1, 1), A(1, 2), A (1, 3) A (2, 1), A (2, 2), A(2, 3).
   An MAXS, MINS, or SEARCH statement retrieves array variables in this order.
6. See I. 1 (8) "Array variables". (page 19).

Simple example:

1. `10 DIM A(5)` ..... Declares a 1-dimensional numeric array variable using A as the numeric array variable name, and 5 as the number of array elements. When this is declared, five variables can be used:

A (1)	A (2)	A (3)	A (4)	A (5)

2. `10 DIM B$(6)*30` ..... Declares a 1-dimensional character array variable using B$ as the character array variable name, 6 as the number of array elements, and 30 bytes as the array element size (up to 30 characters can be assigned). When this is declared, six variables can be used:

B$ (1)	B$ (2)	B$ (3)	B$ (4)	B$ (5)	B$ (6)

3. `10 DIM C$(3,4)` ..... Declares a 2-dimensional array variable using C$ as the character array variable name, 12 (= 3 x 4) as the number of array elements, and 16 bytes as the array element size (up to 16 characters can be assigned). When this is declared, 12 variables can be used:

C$ (1, 1)	C$ (1, 2)	C$ (1, 3)	C$ (1, 4)
C$ (2, 1)	C$ (2, 2)	C$ (2, 3)	C$ (2, 4)
C$ (3, 1)	C$ (3, 2)	C$ (3, 3)	C$ (3, 4)

4. `10 DIM B$(6)*30,A(5)` ..... Declares the same array variables as those declared in Simple examples 1 and 2 at the same time.
5. `10 DIM A@*20` ..... Sets 20 as the number of characters that can be assigned to character variable A@.

Note

1. Numeric array variables are classified into real and integer types. The sizes of the array elements are 8 and 2 bytes respectively. For details, see the real and integer type numeric simple variables (page 12).
2. An array variable must not exceed 32K bytes. For example, if the array element size of a character array variable is set to 32K bytes, up to 1023 array elements can be specified. (When declaring an array variable, the memory size required is array element size x number of array elements + several bytes.)
3. The array element size can exceed 255 bytes if it is within the range explained in Note 2, but generally, the upper limit is set to 255 bytes taking into consideration the INPUT and character string buffers.
   If 256 or more bytes are to be set, use a MOVE statement (see page 187).
4. If the declared array variable name is the same as one already declared, an error (ERROR 20) occurs. For a program declaring many arrays with a DIM statement, it is better to declare all arrays at the same time at the beginning of the program, in order to avoid this error.
5. Use a generalized arry variable to specify all array elements of declared array variables at the same time.

Example: A$(*)

An entire array is specified by writing an asterisk (*) in parenthese.

Application example:

In many cases, an array variable, using a simple variable as a subscript, is used together with a FOR/NEXT statement (see page149).The examples are given below.

[Program A] Assigns the calculated value to an array variable.

```
10 DIM X(10) ..... Declares an array variable.
20 FOR K=1 TO 10 ..... Increases the value of variable K from 1 to 10 with
30 X(K)=K^2 ..... increments by 1 in the FOR . . . NEXT loop.
40 NEXT K ..... Assigns the value of K² to the array variable.
50 END
```

[Program B] Assigns the data input with the keyboard to an array variable.

```
10 INPUT "HOW MANY DATA ITEMS?";I ... Input the number of array elements.
20 DIM X(I) ..... Declares the array variable.
30 FOR K=1 TO I ..... Increments the value of variable K from 1 to
40 DISP ;K;"TH DATA"; I by 1.
50 INPUT X(K) ..... Inputs the numeric value to the array variable.
60 NEXT K
70 END D
```

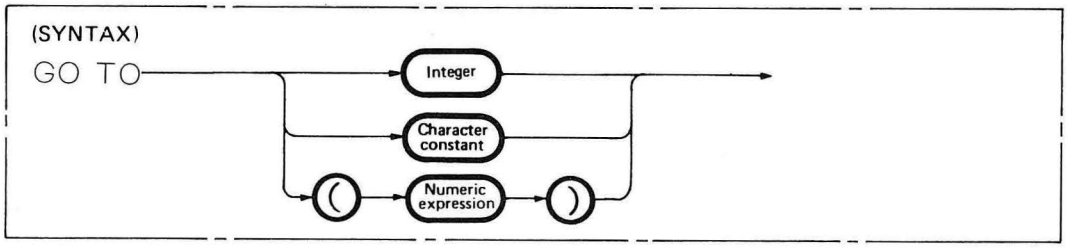
GO TO

Transfers the program execution to the specified program line.

〈Purpose of use〉
① When repeating the same processing during program execution.
② When changing the processing flow (conditionally).

Format: GO TO n | S | (X)

n : Line number of jump destination
S : Line label name of jump destination
X : Line number (numeric expression) of jump destination



Explanation:

1. Unconditionally transfers the program execution to the program line indicated by the line number n, line label name S, or the value of numeric expression X (the fractions are truncated).
   If the GO TO statement is used succeeding an IF statement (see page 141), this GO TO statement specifies a conditional jump. In addition to this GO TO statement, a conditional jump is specified by an ON GO TO, ON ERROR GO TO, ON KEY GO TO, or ON TIME GO TO statement. For details, see the explanations of these statements.
2. If STATUS 0 is used in the numeric expression, this statement can be used for a relative jump*).

Simple example:

```
100 GO TO 200
```
Jumps to the line which has line number 200. (Unconditional jump)

```
100 GO TO "ABC"
```
Jumps to the line having line label name "ABC". (Unconditional jump)

```
100 GO TO(A*5)
```
Jumps to the line indicated by the value of the numeric expression. (Unconditional jump)
For example, if the A value is 100, jumps to the line which has line number 500.

```
100 IF A>0 GO TO 50
```
If the condition of A > 0 is satisfied, jumps to the line which has line number 50. If A ≦ 0, the execution is transferred to the next line. (Conditional jump)

*) Relative jump: The jump destination is given by the relative line number counted from the current program line.

[Program A] Calculates the sum of integers 1 to 100.

```
10 S=0:N=1
20 S=S+N
30 IF N=100 THEN 60
40 N=N+1
50 GO TO 20
60 DISP "TOTAL=";S
70 END
```

- 10: Assigns 0 to numeric variable S and assigns 1 to numeric variable N.
- 20: Assigns the sum of variable S and variable N to variable S.
- 30: If the value of variable N is 100, jumps to the line which has line number 60.
- 40: Increments the value of variable N by 1.
- 50: Jumps to the line which has line number 20.
- 60: Displays the results on the screen.

Examples of a conditional jump are given below.

[Program B] Decides whether the input numeric value is plus, minus, or 0.
(IF . . . . . GO TO (see page 141).)

```
10 INPUT "ENTER A NUMBER ";N
20 IF N<0 GO TO 50
30 IF N=0 GO TO 60
40 DISP "PLUS":END
50 DISP "MINUS":END
60 DISP "ZERO":END
```

- 10: Awaits input of a numeric value to variable N.
- 20: If the value of N is minus, jumps to the line which has line number 50.
- 30: If the value of N is 0, jumps to the line which has line number 60.
- 40: Displays PLUS on the screen and ends program execution.
- 50: Displays MINUS on the screen and ends program execution.
- 60: Displays ZERO on the screen and ends program execution.

[Program C] Selects and executes four fundamental arithmetic operations.
(ON . . . . . GO TO (see page 138).)

```
10 DISP "+       PUSH 1"
20 DISP "-       PUSH 2"
30 DISP "*       PUSH 3"
40 DISP "/       PUSH 4"
50 INPUT N
60 ON N GO TO 70,90,110,130
70 INPUT A,B
80 DISP "ANSWER=";A+B:GO TO 150
90 INPUT A,B
100 DISP "ANSWER=";A-B:GO TO 150
110 INPUT A,B
120 DISP "ANSWER=";A*B:GO TO 150
130 INPUT A,B
140 DISP "ANSWER=";A/B
150 END
```

- 10 to 50: When the line which has line number 50 is executed, the cursor blinks and input is awaited. The execution jumps to the program line for calculating + (addition) if 1 is input, to the program line for calculating − (subtraction) if 2 is input, to the program line for calculating x (multiplication) if 3 is input, and to the program line for calculating ÷ (division) if 4 is input.

[Program D] Calculates a square root. (ON ERROR GO TO (see page 172).)

```
10 ON ERROR GO TO 100
20 INPUT "X=";X
30 IF X<>0:ELSE END
40 DISP SQR X
50 GO TO 20
100 DISP SQR (-X)
110 GO TO 20
```

- 10: If an error occurs, jumps to the line which has line number 100.
- 20: Displays X = and awaits input of a numeric value to variable X.
- 30: If the value of numeric expression X is 0, the condition is not satisfied and executes ELSE and subsequent statement.
- 40: Displays the square root of the value of numeric expression X. If the value of X is minus, causes an error and jumps to the line which has line number 100.
- 50: Jumps to the line which has line number 20.
- 100: Changes the value of numeric expression X to a plus value and displays the square root of this value.
- 110: Jumps to the line which has line number 20.

[Program E] Sequentially displays digits from 1 and stops displaying when the [A] key is pressed.
(ON KEY . . . . . GO TO (see page 176).)

```
10 ON KEY "A" GO TO 50
20 N=N+1
30 DISP N;
40 GO TO 20
50 END
```

- 10: If the [A] key is pressed, jumps to the line which has line number 50.
- 20: Increments variable N by 1.
- 30: Displays the value of variable N.
- 40: Jumps to the line which has line number 20.

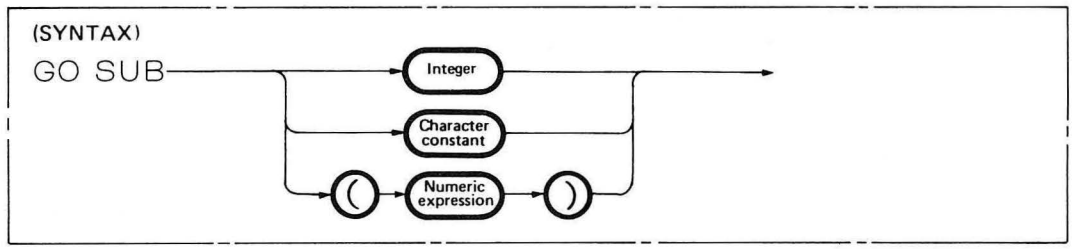
GO SUB

Transfers program execution to the subroutine*) beginning with the specified program line.

〈Purpose of use〉
To transfer the program execution to an arbitrary subroutine.

Format: GO SUB n | S | (X)

n : Subroutine line number
S : Subroutine line label name
X : Subroutine line number (numeric expression)



Explanation:

Transfers the program execution to the subroutine beginning with the program line indicated by the line number n, line label name S, or the value of numeric expression X (the fractions are truncated).

Like a GO TO statement, an unconditional jump is specified if a GO SUB statement is used only. If a GO SUB statement is used following an IF statement, a conditional jump is specified. In addition, a conditional jump is specified in an ON GO SUB, ON ERROR GO SUB, or ON KEY GO SUB statement. For details, see the explanations of these statements.

Use a GO SUB statement together with a RETURN statement explained later. See the explanation on the RETURN statement.

Simple example:

```
100 GO SUB 200
```
Jumps to the subroutine beginning with line number 200. (Unconditional jump)

```
100 GO SUB "SUB-1"
```
Jumps to the subroutine beginning with the line which has line label name SUB-1. (Unconditional Jump)

```
100 GO SUB(X*100)
```
Jumps to the subroutine beginning with the line which has a line number of a numeric expression. (Unconditional jump)

For example, if the value of X is 5, jumps to the subroutine beginning with line number 500.

*) Subroutine: A subroutine is a set of program lines executed frequently as a routine.

```
100 IF A=0 GO SUB 50
```
..... If the condition of A = 0 is satisfied, jumps to the subroutine beginning with line number 50.
If A ≠ 0, the execution is transferred to the next line. (Conditional jump)

Note

1. The processing speed of a program using a subroutine with a GO SUB statement is higher than that of a program using a subprogram with a CALL or SUB statement (see pages (157/160)) and explained later.
2. A subroutine can be called any number of times in the program.
   A subroutine can call another subroutine.
3. The jump destination routine specified in the GO SUB statement ends the subroutine processing with a RETURN statement and returns to the main routine. The return destination from an unconditional jump is the statement following the GO SUB statement, but that from a conditional jump varies according to the command.
   See the following table:

Statement	Return destination
IF . . . . . GO SUB	Statement succeeding IF . . . . . GO SUB
ON . . . . . GO SUB	Statement succeeding ON . . . . . GO SUB
ON ERROR GO SUB	Beginning of the line where the error has occured.
ON KEY . . . . . GO SUB	Line succeeding the line executed when the key was pressed
ON TIME. . .GO SUB	Line succeeding the line executed at the specified time

RETURN

Ends processing of a subroutine and returns control to the main process.

〈 Purpose of use 〉
To end the subroutine and return execution to the program from which the execution jumped to the subroutine.

● **Function 1:** Returns the processing flow to the statement succeeding the GO SUB statement.

Format: RETURN

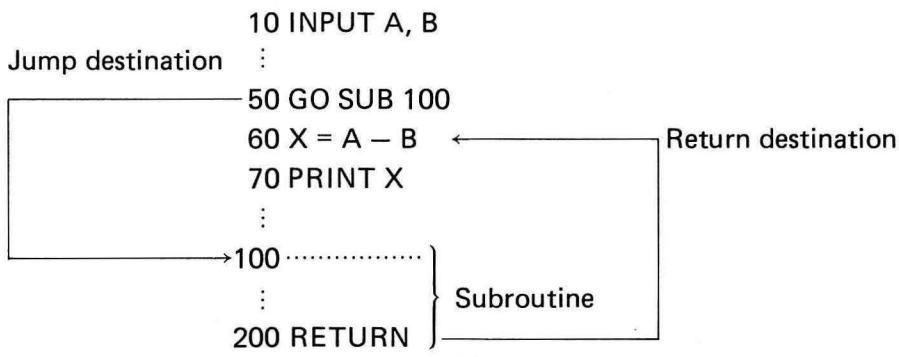
(SYNTAX)

RETURN ——→

Explanation:

Ends execution of the subroutine and returns the processing flow to the statement succeeding the GO SUB statement that has transferred execution to the subroutine. Since the statement to which the execution is returned is stored automatically when jumping to the subroutine, an error (ERROR 10) occurs if this statement is executed without executing the GO SUB statement. This error is caused because the return destination is undefined.

Simple example:



● **Function 2:** Returns the processing flow to the line succeeding the statement to which the processing flow is returned in Function 1.

Format: RETURN NEXT

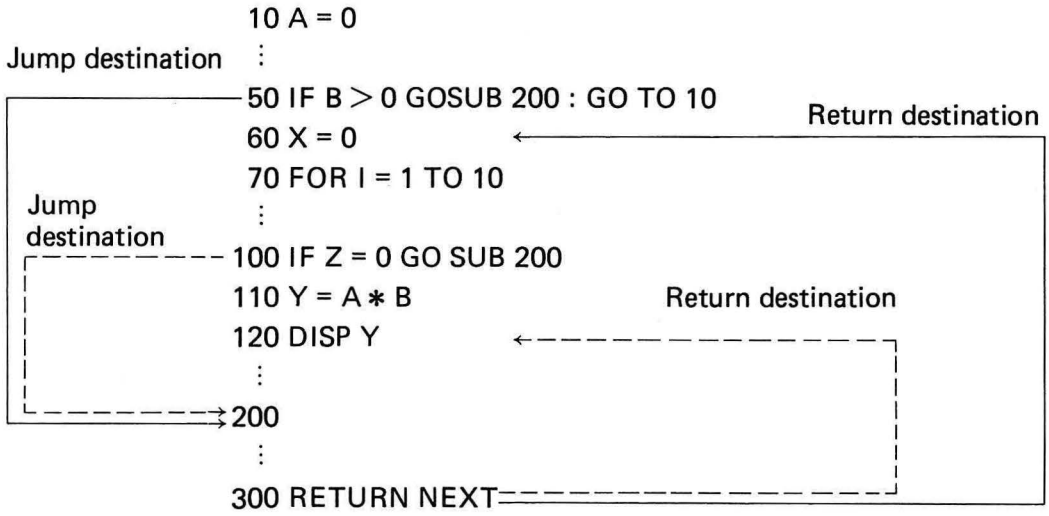
(SYNTAX)

RETURN NEXT ——→

Explanation:

Ends the subroutine and returns the processing flow to the beginning of the line succeeding the statement to which the processing flow is returned in Function 1.

Simple example:



After the execution of the first subroutine, the processing flow returns to the line (line number 60) following the line (line number 50) of the statement (GO TO 10) following the GO SUB statement from which the transfer was made.

After the execution of the next subroutine, the processing flow returns to the line (line number 120) succeeding the line (line number 110) of the statement (Y = A * B) following the GO SUB statement from which the transfer was made.

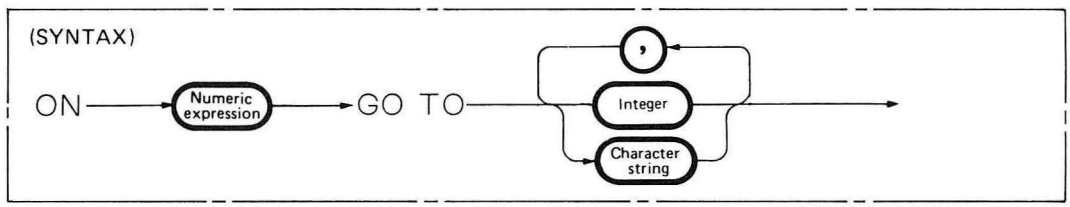
ON···GO TO

Transfers program execution to the program line specified according to the numeric expression value following ON.

〈Purpose of use〉
To select the program line to be executed according to the numeric expression value.

Format: ON X GO TO {n | S}

n : Jump destination line number
S : Jump destination line label name



Explanation:

If the value of numeric expression X is 1, jumps to the first line number after GO TO which has line number n or line label name S; if the value of numeric expression X is 2, jumps to the second line number after GO TO which has line number n or line label name S. That is, if the value of numeric expression X is m, jumps to the mth line number after GO TO indicated in the list of jump destinations. The fractions of the value of numeric expression X are truncated. If the value of numeric expression X is smaller than 1 or greater than the number of jump destinations in the list, execution is transferred to the line succeeding the ON GO TO statement.

Simple example:

```
20 ON A GO TO 50,100,150
```
·····If the value of A is 1, jumps to the line which has line number 50; if the value of A is 2, jumps to the line which has line number 100; if the value of A is 3, jumps to the line which has line number 150. If the value of A is not 1, 2, or 3, transfers to the next line.

For a Application example of a GO TO statement (see page 131).

ON···GO SUB

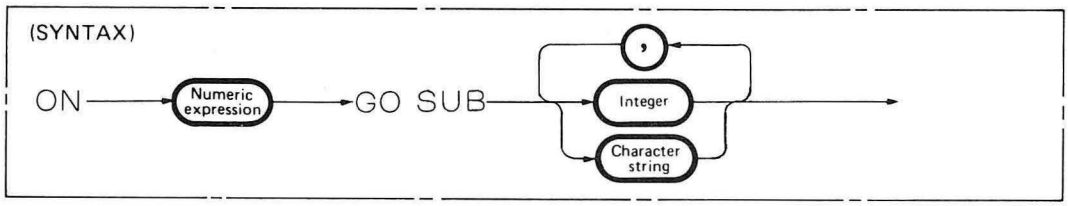
Transfers the program execution to the subroutine beginning with the line specified according to the numeric expression value following ON.

〈Purpose of use〉
To select the program line to be executed according to the numeric expression value.

Format: ON X GO SUB {n | S}

n : Subroutine line number

S : Subroutine line label name



Explanation:

If the value of numeric expression X is 1, jumps to the subroutine beginning with the first line number after GO SUB having line number n or line label name S; if the value of numeric expression X is 2, jumps to the subroutine beginning with the second line number from GO SUB having line number n or line label name S. That is, if the value of numeric expression X is m, jumps to the subroutine beginning with the mth line number after GO SUB indicated in the list of jump destinations. The fractions of the value of numeric expression X are truncated. If the value of expression X is smaller than 1 or greater than the number of the jump destinations in the list, transfers the execution to the next line.

Simple example:

```
20 ON A GO SUB 100,200,"END"
```

If A is 1, jumps to the subroutine beginning with the line which has line number 100; if A is 2, jumps to the subroutine beginning with the line which has line number 200; if A is 3, jumps to the subroutine beginning with the line which has line label name "END". If A is not 1, 2, or 3, transfers execution to the next line.

Application example:

Displays whether the input value is minus, 0, or plus.

```
 10 INPUT "A=";A
 20 ON SGN (A)+2 GO SUB 100,200,300
 30 END
100 DISP "A<0"
110 RETURN
200 DISP "A=0"
210 RETURN
300 DISP "A>0"
310 RETURN
```

If A is minus, SGN (A) is -1 and the numeric expression value becomes 1; therefore, the subroutine beginning with line number 100 is executed. If A is 0, SGN (A) is 0 and the numeric expression value becomes 2; therefore, the subroutine beginning with line number 200 is executed. If A is plus, SGN (A) is 1 and the numeric expression value becomes 3; therefore, the subroutine beginning with line number 300 is executed.

(See an SGN function (page 26).)

IF···THEN

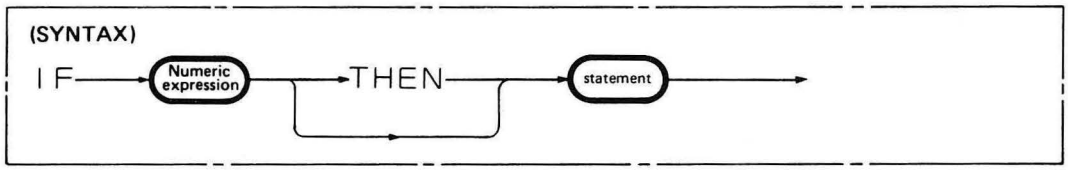
Executes the statement or jumps to the specified line if the condition after IF is satisfied.

〈Purpose of use〉
To specify the processing to be performed when the condition is satisfied.

● **Function 1:** Executes the statement.

Format: IF X [THEN] statement

X : Conditional (numeric expression)
statement : Statement



Explanation:

If the condition indicated by numeric expression X is satisfied, executes the statement following THEN; if not, executes the statement on the next line. If the statement is a LET statement, THEN or LET must be written. If a multistatement is written following THEN, this multistatement is executed only when the condition is satisfied. If the condition is not satisfied, this multistatement is not executed and the statement in the next line is executed.

Simple example:

1. `50 IF A>100 GO TO 100` ····· If A is greater than 100, executes the GO TO statement; if A is less than or equal to 100, executes the statement on the next line.
2. `50 IF A*B<0 DISP A,B:GO TO 100` ····· If A * B is less than 0, executes the DISP and GO TO statements; if A * B is greater than or equal to 0, executes the statement in the next line.

● **Function 2:** Jumps to the specified line

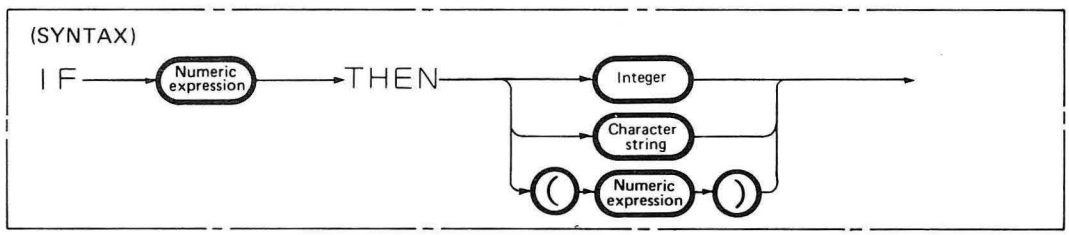
Format: IF X THEN n|S|(Y)

X : Conditional (numeric expression)

n : Jump destination line number

S : Jump destination line label name

Y : The value of numeric expression Y is the jump destination line number (numeric expression).



Explanation:

If the condition indicated by numeric expression X is satisfied, jumps to the line which has the line number n or line label name S, or the one which has the line number indicated by the numeric expression X value (the fractions are truncated); if the condition is not satisfied, executes the statement in the next line.

Simple example:

```
10 IF A>0 THEN 100
```
..... If A is greater than 0, jumps to line number 100.

```
10 IF A*B=0 THEN "ERROR"
```
..... If A * B is 0, jumps to the line which has the label name "ERROR".

Application example:

Write a division processing program that displays an error when the denominator is 0.

```
10 INPUT A,B
20 IF B<>0 THEN C=A/B:GO TO 40
30 DISP "ERROR":GO TO 50
40 DISP C
50 END
```

In the above example, if B is not 0, evoluates C = A/B and jumps to line 40; if B is 0, executes line 30.

Note

If the numeric expression is not a conditional numeric expression, the condition is assumed to be satisfied when the value of numeric expression X is other than 0, and the condition is assumed to be not satisfied when the value of numeric expression X is 0.

< Detailed information >

Conditional numeric expressions can be relational and logical expressions etc. In any of these cases, the condition is assumed to be satisfied when the value is other than 0, and the condition is assumed to be not satisfied when the value is 0.

Example:

When the condition is satisfied	When the condition is not satisfied
10 > 8	10 < 8
5 = 5	6 = 5
13 AND 11	13 AND 0
6 OR 13	0 OR 0
8 – 2	0

IF···:ELSE

If the condition following IF is not satisfied, executes the statement following ELSE, or jumps to the specified line.

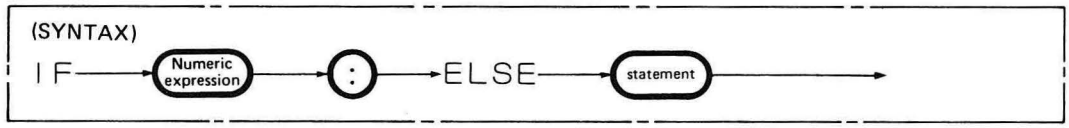
〈Purpose of use〉
To decide the condition and specify the processing to be performed when the condition is not satisfied.

● **Function 1:** Executes the statement

Format: IF X : ELSE statement

X : Conditional expression (numeric expression)

statement : Statement



Explanation:

If the condition indicated by numeric expression X is not satisfied, executes the statement following ELSE; if the condition is satisfied, executes the statement in the next line.

If a multistatement is written following ELSE, this multistatement is executed only when the condition is not satisfied. When the condition is satisfied the following multistatement will not be executed and the statement on the following line will be executed.

Simple example:

```
1. 100 IF A>100:ELSE DISP A
```
.....If A is less than or equal to 100, executes the DISP statement; if A is greater than 100, executes the statement in the succeeding line.

```
2. 100 IF A$="FD1":ELSE DISP A$:GO TO 200
```
.....If the A$ contents are other than FD1, executes the DISP and GO TO statements; if the A$ contents are FD1, executes the statement in the following line.

● **Function 2:** Jumps to the specified line.

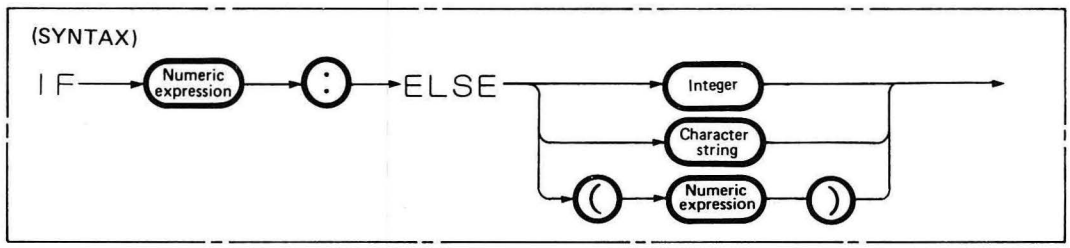
Format: IF X:ELSE n|S|(Y)

X : Conditional expression (numeric expression)

n : Jump destination line number

S : Jump destination line label name

Y : Jump destination line number (numeric expression)



Explanation:

If the condition indicated by numeric expression X is not satisfied, jumps to the line which has line number n or line label name S, or to the line which has the line number of the value of numeric expression Y (the fractions are truncated). If the condition is satisfied, executes the statement in the following line.

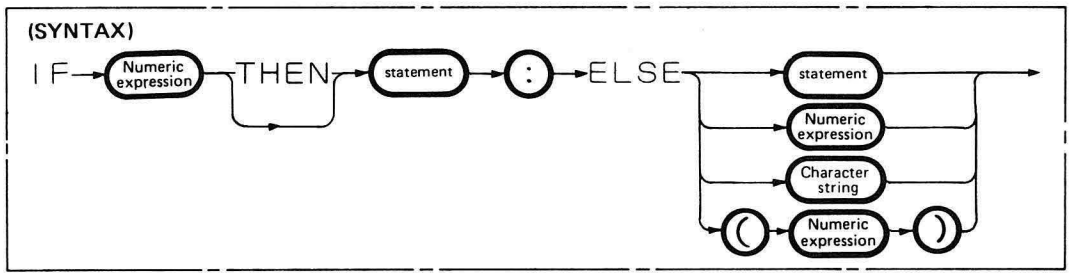
Simple example:

1. `100 IF B<>100:ELSE 200` ..... If B is 100, jumps to the line which has line number 200.
2. `100 IF A$="END" :ELSE "STEP1"` .... If A$ is other than "END", jumps to the line which has the label name "STEP1".
3. `100 IF A-B=0:ELSE (Y*10)` ..... If A − B is other than 0, jumps to the line of which the line number is the value of Y * 10.

● **Function 3:** Specifies the processing to be performed when the condition indicated by numeric expression X is satisfied and the processing or the jump destination to be used when the condition is not satisfied.

Format: IF X (THEN) statement : ELSE statement | n | T | (Z)

X	: Conditional expression (numeric expression)
statement following THEN	: Statement to be executed when the condition is satisfied
statement following ELSE	: Statement to be executed when the condition is not satisfied
n	: Jump destination line number
T	: Jump destination line label name
Z	: Jump destination line number (numeric expression)



Explanation:

If the condition indicated by numeric expression X is satisfied, executes the statement following THEN; if the condition is not satisfied, executes the statement following ELSE or jumps to the line which has line number n or line label name S, or to the line of which the line number is the value of numeric expression Z (the fractions are truncated).

Simple example:

1. `50 IF A$="END" END:ELSE 20` ..... If character variable A$ is END, ends the execution of the program; if character variable A$ is other than END, jumps to line number 20 (when THEN is omitted).

2. `50 IF A=999 LET Y$="END":ELSE "PROCESS"` ..... If numeric variable A is 999, assigns "END" to character variable Y$; if numeric variable A is other than 999, jumps to the line which has the line label name " PROCESS".

Note

If THEN is followed by a LET statement (see page 104), THEN or LET must be written as shown in Simple example 2.

● **Function 4:** Specifies the jump destination to be used when the condition indicated by numeric expression X is satisfied and the processing or jump destination to be used when the condition is not satisfied.

Format: IF X THEN m|S|(Y):ELSE statement|n|T|(Z)

X : Condition (numeric expression)

m : Jump destination line number to be used when the condition is satisfied

S : Jump destination line label name to be used when the condition is satisfied

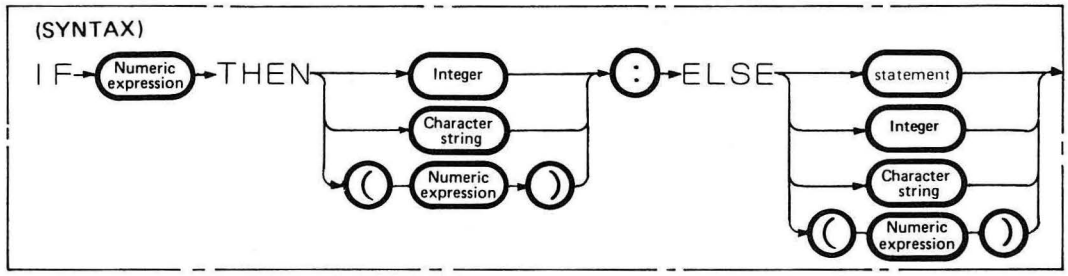
Y : Jump destination line number (numeric expression) to be used when the condition is satisfied

statement : Statement to be executed when the condition is not satisfied

n : Jump destination line number to be used when the condition is not satisfied

T : Jump destination line label name to be used when the condition is not satisfied

Z : Jump destination line number (numeric expression) to be used when the condition is not satisfied



Explanation:

If the condition indicated by numeric expression X is satisfied, jumps to the line which has line number m, or line label name S, or to the line of which the line number is the value of expression Y (the fractions are truncated); if the condition is not satisfied, executes the statement following ELSE, or jumps to the line which has line number n or line label name T, or to the line of which the line number is the value of numeric expression Z (the fractions are truncated).

Simple example:

```
50 IF A<>10 THEN 100:ELSE DISP "ERROR"
```

..... If numeric variable A is other than 10, jumps to line number 100. If numeric variable A is 10, executes the DISP statement.

Application example:

```
    :
    :
50 IF Y<>0 THEN Y=0:ELSE 70
60 PRINT A$
70 INPUT M,N
    :
    :
```

In the above example, assigns 0 to numeric variable Y if the value of Y is other than 0 and jumps to line number 70 if the value of numeric variable Y is 0.

Note

If the numeric expression is not a conditional expression, the condition is assumed to be satisfied when the value of numeric expression X is other than 0 and the condition is assumed not to be satisfied when the value of numeric expression X is 0.

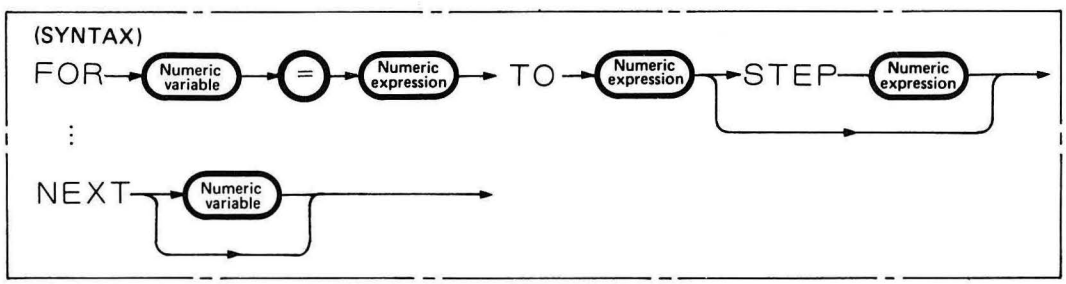
FOR···TO···STEP/NEXT

Executes the statement enclosed between FOR and NEXT repeatedly (loop) under the specified condition.

〈Purpose of use〉
① To execute processing repeatedly.
② To assign a value to individual elements of an array variable, or to output them.

Format: FOR A=X TO Y 〔STEP Z〕
⋮
NEXT 〔A〕

A : Numeric variable
X : Initial value
Y : Last value
Z : Increment



Explanation:

1. First, assigns initial value X to numeric variable A, then executes the first loop. In this case, it executes the lines (statements) enclosed between FOR and NEXT.
   After the above loop, automatically increments the value of numeric variable A by increment Z. If the value of numeric variable does not exceed the last value Y in this case, executes the second loop.
   Subsequently, performs the same processing. If the value of numeric variable A exceeds the last value, terminates the loop, and starts execution of the statement following the NEXT statement.
2. Increment Z may have a decimal part or be a minus value. If the increment Z is plus, stops the loop when A > Y; if the increment is minus, stops the loop when A < Y, then starts execution of the statement following the NEXT statement.
3. When STEP Z is omitted, the increment is assumed to be 1.
4. The numeric variable following FOR and that following NEXT must be the same.
5. In a loop of FOR and NEXT statements, another loop of FOR and NEXT statements can be specified. (Loop nesting*)

*) Loop nesting: To generate a loop in another loop. (Nesting depth or levels)

6. Numeric variable A following the NEXT statement can be omitted.
   If NEXT statements are specified continuously for loop nesting, they can be written by separating them with a comma (,); in such cases, numeric variable A following the NEXT statement must not be omitted.

Simple example:

1.
```
10 FOR J=1 TO 10 STEP 2
    ⋮
50 NEXT J
```
.... Repeats the loop in lines 10 to 50 while incrementing numeric variable J by 2 from 1, until numeric variable J exceeds 10. (See Application example 1.)

2.
```
10 FOR K=5 TO A*B STEP-1
    ⋮
50 NEXT
```
.... Repeats the loop in lines 10 to 50 while incrementing numeric variable K by -1 from 5, until numeric variable K exceeds the value of numeric expression A * B. (The first value of A * B is valid.)

3.
```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT K
80 NEXT J
90 NEXT I
```
Loop 1, Loop 2, Loop 3

.....Loop nesting example (Since STEP is omitted, the increment is 1.)
.....K can be omitted.
.....J can be omitted.
..... I can be omitted.

4. Simple example 3 above can also be specified as follows:

```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT K,J,I
```
Loop 1, Loop 2, Loop 3

..... K, J, and I cannot be omitted. (see explanation 6.)

```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT:NEXT:NEXT
```
Loop 1, Loop 2, Loop 3

Application example:

[Program A] Program for printing the values of numeric variable J of the FOR/NEXT statements.

```
10 FOR J=1 TO 10 STEP 2
20 PRINT J
30 NEXT J
40 PRINT J
50 END
```

In the loop between the FOR and NEXT statements, increases the value of variable J from 1 to 10 by increments of 2.

Execution results:

```
 1
 3
 5
 7
 9
11
```

First value of variable J
Second value of variable J
Third value of variable J
Fourth value of variable J
Fifth value of variable J
Value of variable J after loop

[Program B] Program for inputting five data values and calculating the sum, average value, and maximum value.
(A MAX statement (see page 190) can be used to obtain the maximum value.)

```
 10 DIM A(5)
 20 B=0:C=0
 30 FOR J=1 TO 5
 40 PRINT "DATA NO";J;" =";
 50 INPUT A(J)
 60 PRINT A(J)
 70 IF B<A(J) THEN B=A(J)
 80 C=C+A(J)
 90 NEXT J
100 D=C/5
110 PRINT "***************"
120 PRINT "TOTAL   =";C
130 PRINT "MAXIMUM =";B
140 PRINT "AVERAGE =";D
150 END
```

Declares the array variable.
Loop
Assigns the maximum value at the current time to variable B.
Assigns the sum of data values at the current time to variable C.
Assigns the average value of data to variable D.

(Execution results)

```
DATA NO 1   = 60
DATA NO 2   = 75
DATA NO 3   = 46
DATA NO 4   = 67
DATA NO 5   = 32
**************
TOTAL    = 280
MAXIMUM = 75
AVERAGE = 56
```

[Program C]

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

Loop 1

Loop 2

(Execution results)

```
I= 1          J= 1 }
I= 1          J= 2 } Loop 1
I= 1          J= 3 }
I= 2          J= 1 }
I= 2          J= 2 } Loop 1
I= 2          J= 3 }
I= 3          J= 1 }
I= 3          J= 2 } Loop 1
I= 3          J= 3 }
I= 4          J= 1 }
I= 4          J= 2 } Loop 1
I= 4          J= 3 }
I= 5          J= 1 }
I= 5          J= 2 } Loop 1
I= 5          J= 3 }
I= 6          J= 4 ..... I and J values after the loop
```

Loop 2

Both numeric variables I and J begin at 1. At line 40, the value of J is set to 2 having been increased by 1, and the execution returns to line 20. The value of I is not increased at this time.

Loop 1 is repeated in the same way. When the value of J is 4, loop 1 ends. Then, the value of I is set to 2, increased by 1, then the execution returns to line 20. The value of J is returned to 1 by the FOR/NEXT statements in line 20, then loop 1 is executed repeatedly.

Execution of loops 1 and 2 is completed when the values of I and J are 6 and 4 respectively.

[Program D] Program for printing a multiplication table.

```
 10 DIM A(9,9) ..... Declares the array variable.
 20 PRINT "MULTIPLICATION TABLE"
 30 FOR L=1 TO 9
 40 PRINT TAB (4*(L-1)+4);L;")";   } Loop 1
 50 NEXT L
 60 PRINT ..... Line feed
 70 FOR M=1 TO 9
 80 PRINT M;")";
 90 FOR N=1 TO 9
100 A(M,N)=M*N                     } Loop 2
110 PRINT TAB (4*(N-1)+4);A(M,N);  } Loop 3
120 NEXT N
130 PRINT ..... Line feed
140 NEXT M
150 END
```

In the above four examples, the execution results are output to the printer. When the PRINT statements in these programs are substituted with a DISP statement, the execution results are output to the CRT.

(Execution results)

```
MULTIPLICATION TABLE
     1  2  3  4  5  6  7  8  9
 1   1  2  3  4  5  6  7  8  9
 2   2  4  6  8  10 12 14 16 18
 3   3  6  9  12 15 18 21 24 27
 4   4  8  12 16 20 24 28 32 36
 5   5  10 15 20 25 30 35 40 45
 6   6  12 18 24 30 36 42 48 54
 7   7  14 21 28 35 42 49 56 63
 8   8  16 24 32 40 48 56 64 72
 9   9  18 27 36 45 54 63 72 81
```

Note

1. The execution can leave the loop while repeating the loop enclosed in the FOR and NEXT statements, but the execution cannot jump into the loop from outside without passing through the FOR statement. If the execution jumps out from the loop before executing the NEXT statement, the execution can jump into the loop without passing through the FOR statement.
2. The nesting (depth) of the FOR/NEXT loop is related to the subroutine started with a GO SUB statement (see page 134). If there is no GO SUB statement in the FOR/NEXT loop and there is no FOR/NEXT loop in the subroutine, the nesting is available up to 14 levels when numeric variable A is a real-type variable. When numeric variable A is an integer-type variable, the levels are increased to approximately double that of a real-type variable. If the upper limit of these nesting levels is exceeded, an error (ERROR 14) occurs.

(Incorrect example of FOR/NEXT)

```
 10 FOR I=1 TO 10
  :                          }
  :                          }
 50 FOR J=2 TO 5             }  Loop 1
  :                          }
  :                          }
100 NEXT I                   }
110 PRINT A                  }  Loop 2
120 NEXT J                   }
```

A loop can be specified in another loop, but they cannot be overlapped.

```
 10 FOR I=5 TO 10
 20 A=A+I
 30 PRINT A
 40 NEXT
  :
  :
100 GO TO 20
```

Loop

The execution cannot jump into a loop without passing through the FOR statement. (See Note 1.) The following nesting is allowed:

```
 10 DIM A(5)
 20 Y=0
 30 FOR J=1 TO 5
 40 INPUT A(J)
 50 IF A(J)>100 GO TO 90
 60 Y=Y+A(J)
 70 NEXT J
 80 DISP Y:END
 90 DISP "ERROR";A(J)
100 GO TO 40
```

Loop

In this example, the execution leave the loop before execution of the NEXT statement. Only in this case can, the execution jump back into the loop. The value of variable J set when the execution jumps into the loop is the same as that set when the execution leave the loop. (See Note 1.)

4. SPECIAL STATEMENTS

Here are explained the special statements to make more effective use of this system.

CALL
SUB
SUBEND
KEY
KEY I N
POP
ON ERROR
OFF ERROR
ON KEY
OFF KEY
ON T I ME
OFF T I ME
RESUME
MOVE ※
MAXS ※
M I NS ※
SEARCH ※
DEFFN
MUS I C ※
WA I T ※
DEF I NT ※
DEFREAL ※
DEFSTR ※
DEG ※
RAD ※
RANDOM ※
ERROR MUS I C ※

The statements marked with ※ can also be used in manual operation.

CALL

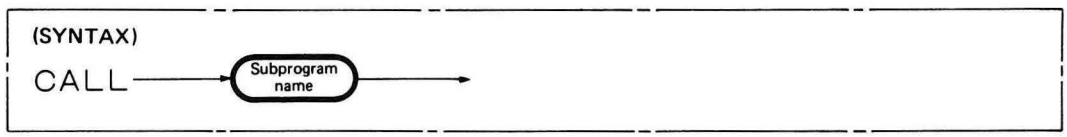
Calls a subprogram*).

〈Purpose of use〉
To call a part of a program as an independent program.

● **Function 1:** Without argument

Format: CALL SF

SF: Subprogram name



Explanation:

Calls subprogram SF.

Simple example:

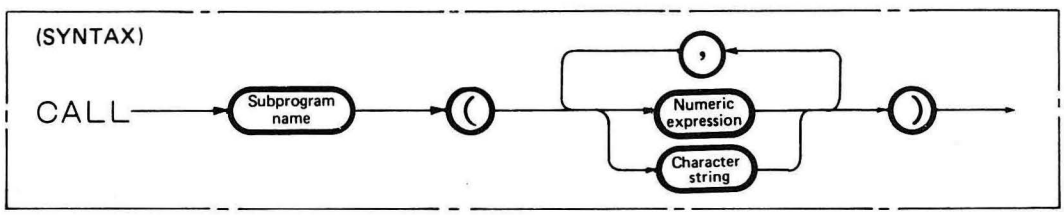
```
10 CALL PROGRAM
```
..... Calls a subprogram named PROGRAM.

● **Function 2:** With argument

Format: CALL SF (|X|C|)

SF : Subprogram name

X or C: Argument



Explanation:

1. Calls subprogram SF with argument X or C written in parentheses.
2. When argument X or C is a variable whose value should not be changed after returning from the subprogram, enclose it in parentheses like (X) or (C), then the value is not changed.
3. When argument X or C is an array variable, write X(*) or C(*) (generalized array variable) to make the entire array variable the argument.

Simple example:

```
10 CALL CALCU(A,B,C$)
```
..... Calls the subprogram named CALCU with argument A, B, and C$.

*) Subprogram: A program which is in a program as an independent entity. A program which calls a subprogram is called a main program.

Application example:

[Program A]

```
10 A=15
20 B=32
30 C=99
40 D=99
50 CALL CALCU(A*2,B,C,D)
60 DISP "TOTAL C=";C
70 DISP "TOTAL D=";D
80 END
90 REM SUBPROGRAM
100 SUB CALCU(X,Y,Z1,Z2)
110 Z1=X+Y
120 Z2=X+Y
130 SUBEND
```

Values of variables before calling the subprogram

Main program (The program which calls the subprogram)

Arguments are passed as follows.

```
50 CALL CALCU(A*2,B,C,D)
100 SUB CALCU(X,Y,Z1,Z2)
```

Subprogram

[What Is An Argument]

Let us consider lines 50 and 100 from [Program A].

```
50 CALL CALCU(A*2,B,C,D)
100 SUB CALCU(X,Y,Z1,Z2)
```

When program execution reaches line 50, the CALL statement is executed to call a subprogram (a program from line number 100 to 130) named CALCU.

The values of A * 2, B, C, and D are passed to variables X, Y, Z1, and Z2 used in the subprogram, then subprogram CALCU is executed, after which control returns to the next line (line 60) of line number 50. The values of X, Y, Z1, and Z2 obtained by the subprogram execution are passed to A * 2, B, C, and D. A * 2, B, C, and D passed from the main program to the subprogram by the CALL statement and X, Y, Z1 and Z2 returned from the subprogram to the main program are called arguments. Arguments are used to exchange information between the main program and its subprogram by a CALL statement.

(Execution result)

```
>RUN
TOTAL C=62
TOTAL D=62
READY
```

The values of variables C and D are changed after the subprogram is called.

In order not to change the value of variable D, change line 50 to

```
50 CALL CALCU(A*2,B,C,(D))
```

(Execution result)

```
>RUN
TOTAL C=62
TOTAL D=99
READY
```

The value of variable D is the same as that before the subprogram is called.

[Program B]

```
 30 C=99
 40 D=99
 50 CALL CALCU(15*2,32,C,D)
 60 DISP "TOTAL C=";C
 70 DISP "TOTAL D=";D
 80 END
 90 REM SUBPROGRAM
100 SUB CALCU(X,Y,Z1,Z2)
110 Z1=X+Y
120 Z2=X+Y
130 SUBEND
```

When numeric constants are used as arguments.

This program uses numeric constants as arguments. The execution result is the same as that of [Program A].

SUB

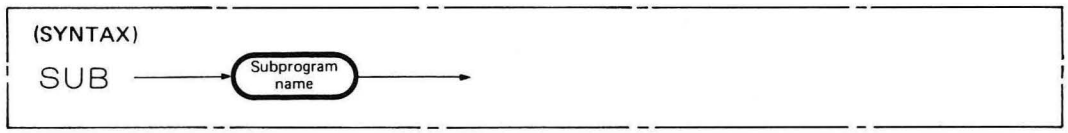
Indicates a subprogram.

〈Purpose of use〉
① When many similar procedures are included in a program.
② To divide a program into several independent blocks because it is so large that it is difficult to grasp the overall flow.

● **Function 1:** Without argument

Format: SUB SF

SF: Subprogram name



Explanation:

Indicates that the program is a subprogram named SF.

Simple example:

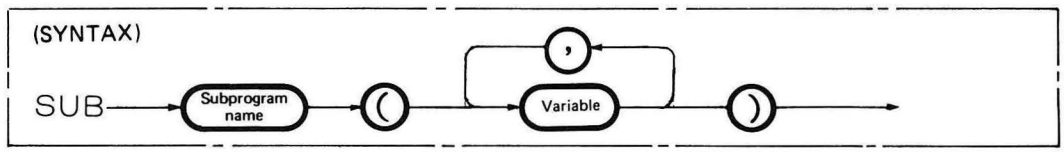
```
100 SUB Total
```
..... Indicates that the program is a subprogram named Total.

● **Function 2:** With argument

Format: SUB SF (|N|)

SF: Subprogram name

N: Argument



Explanation:

Indicates that the program is a subprogram named SF with argument N written in parentheses.

Simple example:

```
100 SUB CALCU(A,B(*),C$)
```
..... Indicates that the program is a subprogram named CALCU with arguments, i.e. numeric variable A, numeric array variable B(*), and character variable C$.

Application example:

See the application example (page 157) of the CALL statement.

Note

1. Processing cannot be stopped in a subprogram, nor can the VRUN (page 80) and the ERASE commands (page 95) be used in it. The SUBEND statement (explained next) must be executed before such processing.
2. Start a subprogram with a line number greater than the largest line number in the main program.

< Detailed information >

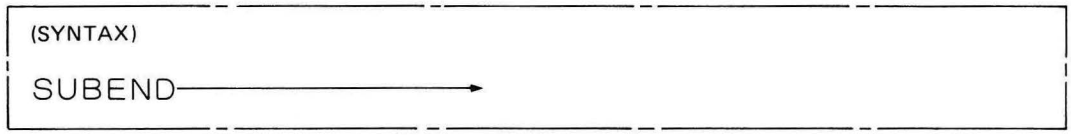
① The variables used in a subprogram specified by a CALL or SUB statement are independent of the variables used in the main program (and other programs). Subprograms can be used without concern for the variables used in the main program. Also, the variable declaration (page 13) is dependent. The variable declarations in the main program are invalid in its subprograms.

② FNs (constant functions defined by the DEFFN statement) in a subprogram are independent of FNs in the main program; the FNs defined by the DEFFN statement in the main program are invalid in its subprograms.

SUBEND

Indicates the end of a subprogram.

Format: SUBEND



Explanation:

Ends a subprogram, and returns control from the subprogram to the main program. Control returns to the statement after the CALL statement (page 157) which called the subprogram.

Simple example:

```
 :
50 CALL CALCU(A,B,C)
60 DISP "TOTAL IS";C
 :
100 SUB CALCU(X,Y,Z)
110 Z=X+Y
120 SUBEND
```

Main program

Subprogram

In this example, the CALL statement at line number 50 calls subprogram CALCU lines · 100 to 120 are executed, then control returns to the main program where the DISP statement at line number 60 is executed.

KEY

Enters one character with the keyboard.

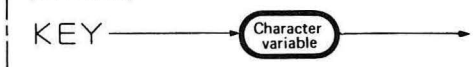
〈Purpose of use〉
To enter one character with the keyboard during program execution.

● **Function 1:** Assigns one character with the keyboard to the specified character variable

Format: KEY A$

A$: Character variable

(SYNTAX)



Explanation:

1. Assigns one character with the keyboard to character variable A$.
2. Unlike the INPUT statement (page 117), execution advances to the next line if nothing is entered, also the entered character is not displayed on the CRT.

Simple example:

```
10 KEY C$
```
..... Assigns the character entered from the keyboard to character variable C$.

● **Function 2:** Invalidates the previously entered key.

Format: KEY

(SYNTAX)



Explanation:

1. Clears the key input buffer; the previously entered key is invalidated.
2. Used to invalidate the key entered before the required key is entered.

Application example:

Displays the key pushing at the top left corner of the CRT. When the key is E, however, END is displayed and execution ends.

```
 10 DISP CLEAR ..... Clears the screen.
 20 KEY A1$:IF A1$="E" THEN 90:ELSE 40 ..... Ends execution if the key is E.
 30 CURSOR 0,0:DISP " ":GO TO 20 ..... Jumps to line 20 if no key is entered.
 40 KEY ..... Clears the input key.
 50 KEY A2$:IF A2$="E" THEN 90:ELSE 70 ..... Ends execution if the key is E.
 60 GO TO 20 ..... Jumps to line 20 if no key is entered.
 70 IF A1$=A2$:ELSE 20
 80 CURSOR 0,0:DISP A1$:GO TO 40 ..... Displays the key which is held down.
 90 CURSOR 0,0:DISP "END"
100 END ..... Displays END.
```

KEYIN

Enters numeric values and characters with the keyboard.

〈Purpose of use〉
To display the entered numeric values right-justified, or the entered characters left-justified. This statement prevents entry of characters in excess of the specified number, and is useful when formatting tables.

● **Function 1:** Enters numeric values.

Format: KEY I N A, X, Y, C

A: Numeric variable
X: Total number of digits
Y: Number of decimal digits
C: Display format

"N" ..... The leading zeros are not suppressed. The sign of numeric value is not displayed.

"L" ..... The sign (minus) of numeric value is displayed preceding the display digits. The plus sign is replaced by a space.

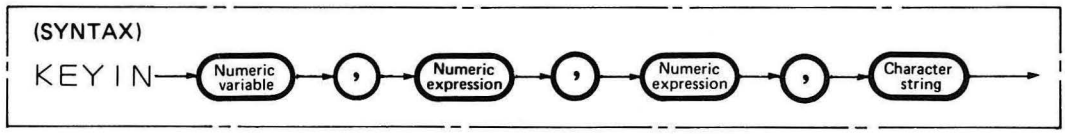
"R" ..... The sign (minus) of numeric value is displayed following the display digits. The positive sign is replaced by a space.

"F" ..... The sign (minus) of numeric value is displayed preceding the numeric value. (floating display) For a positive value, the sign display digit for the negative value is used for number display.

"L,"..... Separates each 3 digits by a comma (,) for partitioning. The sign display is the same as "L".

"R,"..... Separates each 3 digits by a comma (,) for partitioning. The sign display is the same as "R".

"F,"..... Separates each 3 digits by a comma (,) for partitioning. The sign display is the same as "F".



Explanation:

1. Displays the numeric value to be assigned to numeric variable A, in the format indicated by character string C within the total number of X digits leftward from the current cursor position. To handle only integers, set numeric expression Y to 0.

2. The formats indicated by character string C are listed below.

Character string C	Zero-suppression	3-digit positioning	Sign
"N"	X	X	None
"L"	O	X	Left end
"R"	O	X	Right end
"F"	O	X	Floating (minus only)
"L,"	O	O	Left end
"R,"	O	O	Right end
"F,"	O	O	Floating (minus only)

A period (.) can be used instead of a comma (,).

3. Exponent-format and hexadecimal numbers cannot be entered.
4. When something is already displayed within the specified number of digits, this area is cleared by entering the first digit, and the entered digits are displayed.
5. The total number of digits X includes the positioning comma (,), sign (minus or the space for sign display when "L", "R", "L,", or "R,", is specified), and the decimal point.
6. When the [ENTER] key is pressed with all the specified number of digits padded with spaces, 0 is assigned to the numeric variable if nothing has been assigned. If a value is already assigned, the value is reinput.

Simple example:

1. `10 KEYIN A,7,2,"N"` ... Assigns the keyed in numeric data to numeric variable A, and displays it with a total of 7 digits (2 decimal digits) in format N.

(Input data)		(Display)
123	[ENTER]	0123.00
-12.34	[ENTER]	0012.34
1234.56	[ENTER]	1234.56
1234567	[ENTER]	Error
1234567	[—ENTER]	Error
1.234	[ENTER]	Error

2. `10 KEYIN A,7,2,"L"` .... Assigns the keyed in numeric data to numeric variable A, and displays it with a total of 7 digits (2 decimal digits) in format L.

(Input data)		(Display)
123	[ENTER]	123.00
-12.34	[ENTER]	- 12.34
-123.45	[ENTER]	-123.45
1234.56	[ENTER]	Error
1234567	[—ENTER]	Error
1.234	[ENTER]	Error

3. `10 KEYIN A,7,2,"R"` ... Assigns the keyed in numeric data to numeric variable A, and displays it with a total of 7 digits (2 decimal digits) in format R.

(Input data)		(Display)
123	ENTER	123.00
-12.34	ENTER	12.34-
-123.45	ENTER	123.45-
1234.56	ENTER	} Error
1234567	(-) ENTER	
1.234	ENTER	

4. `10 KEYIN A,7,2,"F"` .... Assigns the keyed in numeric data to numeric variable A, and displays it with a total of 7 digits (2 decimal digits) in format F.

(Input data)		(Display)
123	ENTER	123.00
-12.34	ENTER	-12.34
1234.56	ENTER	1234.56
-1234.56	ENTER	} Error
1234567	(-) ENTER	
1.234	ENTER	

5. `10 KEYIN A,9,2,"R,"` ... Assigns the keyed in numeric data to numeric variable A, and displays it with a total of 7 digits (2 decimal digits) in format R,.

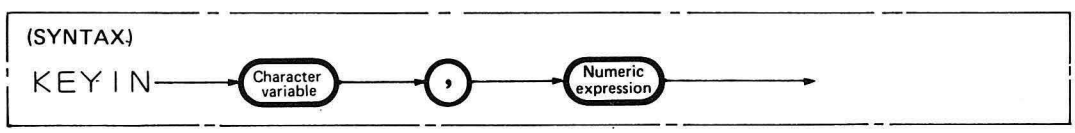
(Input data)		(Display)
1234	ENTER	1,234.00
-123.45	ENTER	123.45-
1234.56	ENTER	1,234.56
-12345.6	ENTER	} Error
1234567	(-) ENTER	
123.456	ENTER	

● **Function 2:** Enters characters.

Format: KEY I N A$, X

A$: Character variable

X: Total number of characters



Explanation:

1. Displays the characters to be assigned to character variable A$ left-justified within the total number of X characters rightward from the current cursor position. The trailing spaces are ignored.
2. If something is already displayed within the specified number of characters, these characters are replaced by each character entered with the keyboard.
3. When the [ENTER] key is pressed with all the specified characters padded with spaces, a null string (" ") is assigned to the character variable.

Simple example:

```
KEYIN A$,5.....
```
Assigns the keyed-in character data to A$, and displays it with a total number of 5 characters.

(Input data)		(Display)
ABC	[ENTER]	ABC
SHARP	[ENTER]	SHARP
COMPUT	Error	

Note

1. This statement enters information within the specified number of characters (digits) displayed on CRT, not directly from the keyboard. Therefore, if the keyed-in characters (digits) are less than the specified number of characters (digits), the entered information may include other displayed characters (digits).
2. When more than the specified number of characters (digits) are keyed in, an error (ERROR 31) occurs.

< Detailed information >

① An error caused by key-in errors can be released by the [CL] key or [↑] key.

② Take the following steps to correct the entered value.

(1) For numeric entry

Press the [CL] key to clear the display in the range of the specified number of digits. For numeric value correction, [→] and [DEL] keys may be used instead of the [CL] key. The [→] key and the [DEL] keys have the same function. When either of these keys is pressed once, the least significant digit is deleted and the display in the range of the specified number of digits is shifted one digit to the right.

(2) For character entry

Press the [CL] key to clear the display in the range of the specified number of characters. For character correction, the [←], [→], [INS], and [DEL] keys may be used instead of the [CL] key. Move the cursor by using the [←] and [→] keys. By pressing the [←] or [→] key once, the cursor moves one character position in the direction indicated by the arrow.

Use the [INS] key to insert a character in the current cursor position.

Use the [DEL] key to delete the character in the current cursor position.

Note

1. If the number of characters exceeds the specified number of characters after the [INS] key operation, the characters outside the range are cleared. (This does not result in an error.)
2. An error (ERROR 31) occurs if a key other than the [ENTER] or the [DEL] key is pressed when the cursor is positioned immediately after the range of the specified number of characters.

③ An error which occurred during program execution can be released by the [CL] key or the [↑] key. However, this simply releases the error; the program execution is not completed.

Application example:

Table-making

```
 10 DIM A$(4),A(4,4)
 20 DISP CLEAR :CHANGE DISP "80"
 30 TABLE ,2,7,12,11,9,15:5,3,3,3,3,3
 40 CURSOR 3,6:DISP "CODE"
 50 CURSOR 13,6:DISP "NAME"
 60 CURSOR 23,5:DISP "UNIT"
 70 CURSOR 23,6:DISP "PRICE"
 80 CURSOR 32,6:DISP "QUANTITY"
 90 CURSOR 42,6:DISP "TOTAL PRICE"
100 FOR I=1 TO 4
110 CURSOR 6,3*I+6:KEYIN A(1,I),4,0,"N"
120 CURSOR 11,3*I+6:KEYIN A$(I),9
130 CURSOR 29,3*I+6:KEYIN A(2,I),7,2,"F,"
140 CURSOR 37,3*I+6:KEYIN A(3,I),5,0,"F,"
150 A(4,I)=A(2,I)*A(3,I)
160 CURSOR 42,3*I+6:DISP USING 190:A(4,I)
170 NEXT I
180 CURSOR 0,21:END
190 IMAGE "$###,###.#Z  "
```

(Keyed-in data)

1. APPLE, .25, 150
2. ORANGE, .23, 250
3. PINEAPPLE, 4.25, 180
4. PEACH, 1.23, 2500

(Execution result)

```
>RUN
```

CODE	NAME	UNIT PRICE	QUANTITY	TOTAL PRICE
0001	APPLE	0.25	150	$37.50
0002	ORANGE	0.23	250	$57.50
0003	PINEAPPLE	4.25	180	$765.00
0004	PEACH	1.23	2,500	$3,075.00

```
READY
>
```

POP

Pops up*) the subroutine stack.*)

〈Purpose of use〉
To return control from a subroutine to the process, bypassing the previous subroutine from which the current subroutine is called.

Format: POP

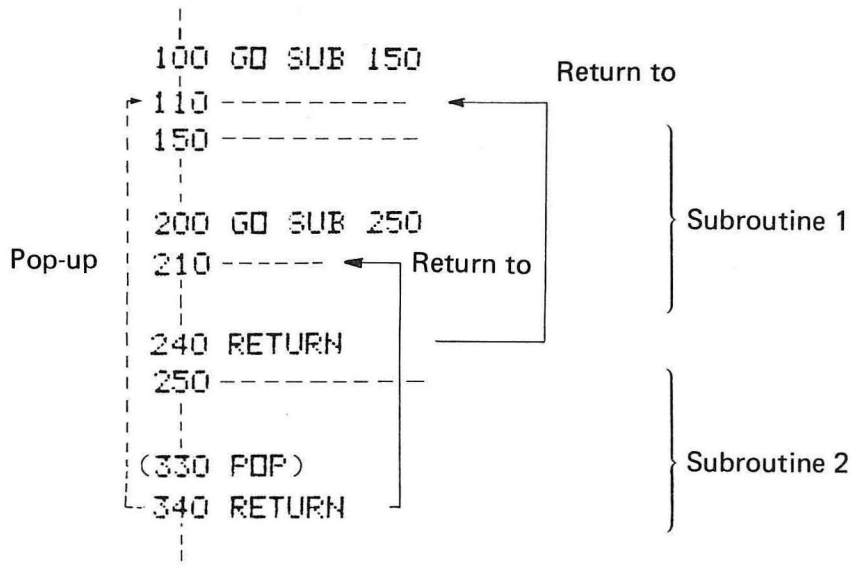
(SYNTAX)

POP→

Explanation:

1. Clears the previous subroutine return address from the subroutine stack. (This operation is called pop-up.)
2. The subroutine stack areas can be continuously popped up by specifying the POP statement repeatedly (in multi-statement mode, etc.).

Simple example:



Without the POP statement on line 330, execution returns to the line following line 200. When a POP statement is used as in the above example, execution returns to the line following line 100.

Note

When a POP statement is used in a FOR/NEXT statement (page 149) in a subroutine, the FOR/NEXT loop is also popped up.

*) Pop-up: To clear the return addresses one by one.

*) Subroutine stack: An area used for storing subroutine return addresses.
When control returns from a subroutine, the last return address area is cleared.

ON ERROR

Recovers the program from the error and continues execution when an error occurs during program execution until an OFF ERROR statement is encountered.

〈Purpose of use〉

To recover the program from an error and continue execution without correcting the error, when it occurs during program execution.

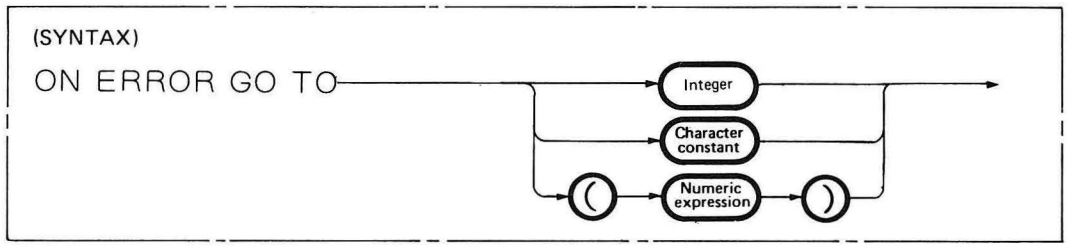
● **Function 1:** Jumps using GO TO.

Format: ON ERROR GO TO n | S | (X)

n: Line number of the line to be jumped to

S: Line label of the line to be jumped to

X: Line number (expression) of the line to be jumped to



Explanation:

Jumps to the line with line number n, line label S, or the line number indicated by the value of the numeric expression X (with decimal part truncated) when an error occurs during program execution.

Simple example:

1. `10 ON ERROR GO TO 100` ..... Jumps to line 100 when an error occurs.
2. `10 ON ERROR GO TO "ABC"` ..... Jumps to the line with line label ABC when an error occurs.
3. `10 ON ERROR GO TO (A+10)` ..... Jumps to the line with the line number indicated by the value of numeric expression (A + 10) when an error occurs.
   For example, if A = 20, jumps to line 30 when an error occurs.

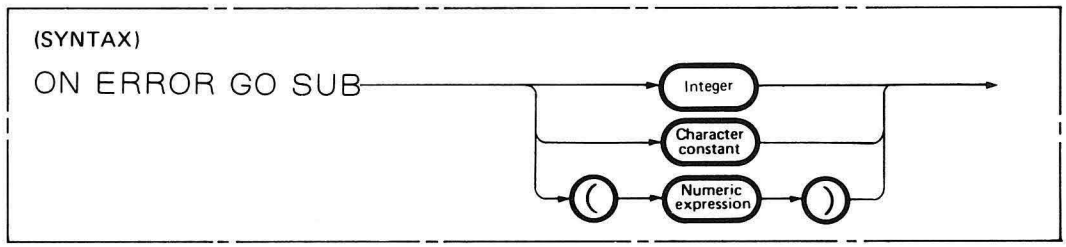
● **Function 2:** Jumps to a subroutine.

Format: ON ERROR GO SUB n | S | (X)

n: Line number of the subroutine to be jumped to

S: Line label of the subroutine to be jumped to

X: Line number (expression) of the subroutine to be jumped to



Explanation:

1. Jumps to the subroutine starting at line number n, line label S, or the line number indicated by the value of numeric expression X (with decimal part truncated) when an error occurs during program execution.
2. Control returns from the subroutine to the beginning of the line where the error occurred. To return from the subroutine, execute a RESUME statement (page 185).

Simple example:

1. `10 ON ERROR GO SUB 200` ..... Jumps to the subroutine starting at line 200, when an error occurs.
2. `10 ON ERROR GO SUB "SUB1"` ..... Jumps to the subroutine starting at line labelled SUB1 when an error occurs.
3. `10 ON ERROR GO SUB (1000-A)` ..... Jumps to the subroutine starting at the line indicated by the value of numeric expression (1000-A) when an error occurs. For example, if A = 100, jumps to the subroutine starting at line 900 when an error occurs.

Note

1. The errors discussed here do not include syntax errors, etc.
   This statement is effective only for the errors with odd-numbered code numbers listed in IV. "Error Code List" (page 51) of the Appendix.
2. When this statement is redeclared, the previous declaration becomes invalid.

Application example:

[Program A] Jumps to line 100 when an error occurs, and ends execution if input data is 0.

```
 10 ON ERROR GO TO 100 ..... Jumps to line 100 when an error occurs.
 20 INPUT "X=";X,
 30 IF X=0 THEN GO TO 200
 40 DISP "SQR X=";SQR (X) ... When X is negative, an error occurs and execution
 50 GO TO 20                     jumps to line 100.
100 DISP SQR (-X)
110 GO TO 20
200 END
```

(Input data) 256, −81, 324, −169, 0

(Execution result)

```
>RUN
X=256                    SQR X= 16
X=-81                    SQR X= 9
X=324                    SQR X= 18
X=-169                   SQR X= 13
X=0
READY
>
```

[Program B] Jumps to the subroutine starting at line 100 when an error occurs.

```
10 INPUT "X=";X,"Y=";Y
20 ON ERROR GO SUB 100
30 Z=X^Y ..... When X is negative, an error occurs and execution jumps to the subroutine starting at line 100.

40 OFF ERROR ..... Releases ON ERROR.
50 DISP "X^Y=";Z
60 GO TO 10 ..... Returns to line 10, and ON ERROR previously released by line 40 is redeclared.

70 END
100 REM SUB ROUTINE
110 IF Y<>INT Y THEN DISP "ERROR":END
120 Y1=Y/2
130 IF Y1=INT Y1 THEN X=-X:RESUME
140 Z=-(-X)^Y
150 RESUME NEXT ..... Execution returns from the subroutine to line 40, which is the next line after that in which the error occurred. (line 30)
```

Subroutine

(Input data) 3, 2, −5, 3, −4, 1.2

(Execution result)

```
>RUN
X=3                      Y=2
X^Y= 9
X=-5                     Y=3
X^Y=-125
X=-4                     Y=1.2
ERROR
READY
>
```

OFF ERROR

Releases the ON ERROR statement.

Format: OFF ERROR

(SYNTAX)

ON KEY

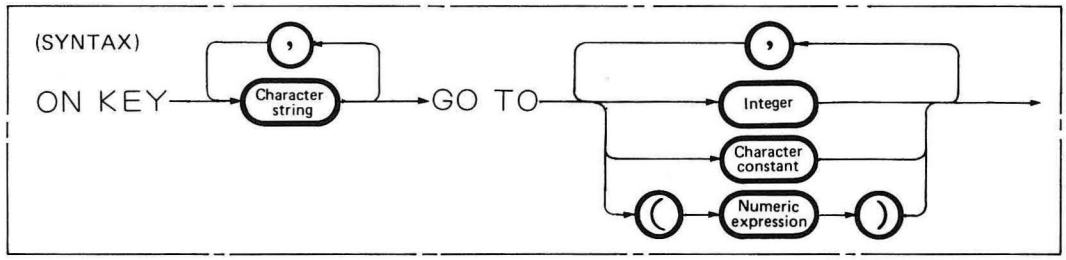
Jumps to the specified line when the specified key is entered until an OFF KEY statement is executed.

〈Purpose of use〉
To change the flow of the program being executed by entering a specific key.

● **Function 1:** Jumps using GO TO.

Format: ON KEY {C} GO TO {n | S | (X)}

C: Specified key
n: Line number of the line to be jumped to
S: Line label of the line to be jumped to
X: Line number (expression) of the line to be jumped to



Explanation:

1. Jumps to the line with line number n, line label S, or the line number indicated by the value of numeric expression X (with decimal part truncated) when character string C is keyed in.
2. A jump is made after executing the line being processed when a key entry is given to character string C.

Simple example:

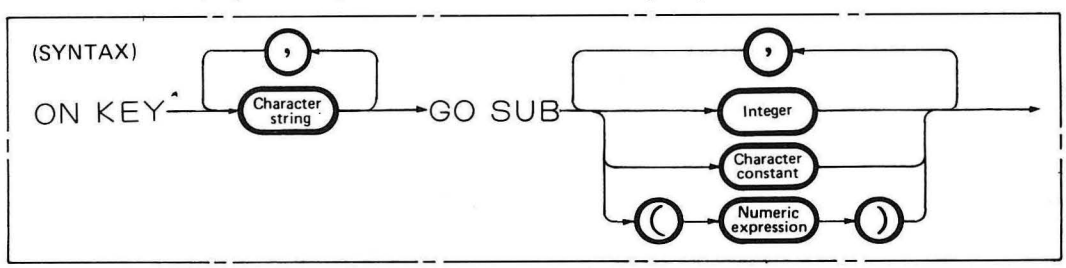
```
10 ON KEY "A","B","C" GO TO 100,"ABC",(150+50)
```

Jumps to line 100 when A is keyed in, to line labeled ABC when B is keyed in, and to line 200 when C is keyed in.

● **Function 2:** Jumps to a subroutine.

Format: ON KEY {C} GO SUB {n | S | (X)}

C: Specified key
n: Line number of the subroutine to be jumped to
S: Line label of the subroutine to be jumped to
X: Line number (expression) of the subroutine to be jumped to



Explanation:

1. Jumps to the subroutine starting at the line with line number n, line label Ś, or the line number indicated by the value of numeric expression X (with decimal part truncated) when character string C is keyed in.
2. A jump is made after executing the line being processed when a key entry is given to character string C. Control returns to the next line after the line where the jump was made to the subroutine. To return from the subroutine, execute a RESUME statement (page 185).

Simple example:

```
10 ON KEY "A","B","C" GO SUB (X+50),"ABC",100
```

Jumps to the subroutine starting at the line with the line number indicated by the value of numeric expression (X + 50) when A is keyed in, to the subroutine starting at the line labeled ABC when B is keyed in, and to the subroutine starting at line 100 when C is keyed in.

Note

1. Up to 28 character strings Cs can be specified following ON KEY, i.e., up to 28 keys can be specified.
2. Only the first character of each character string C is valid as a key in this statement.
3. To specify a definable key or a special key as a character string C, use the character string which corresponds to the key.

Key	Character string
DEF 1	CHR$ &01 (or CHR$ 1)
DEF 2	CHR$ &02 (or CHR$ 2)
|	| |
DEF 9	CHR$ &09 (or CHR$ 9)
DEF10	CHR$ &0A (or CHR$ 10)
DEF11	CHR$ &11 (or CHR$ 17)
DEF12	CHR$ &12 (or CHR$ 18)
|	| |
DEF19	CHR$ &19 (or CHR$ 25)
DEF20	CHR$ &1A (or CHR$ 26)
INS	CHR$ &0B (or CHR$ 11)
DEL	CHR$ &7F (or CHR$ 127)
ENTER	CHR$ &0D (or CHR$ 13)
ENTER	CHR$ &0E (or CHR$ 14)
CL	CHR$ &0F (or CHR$ 15)
HOME	CHR$ &1B (or CHR$ 27)
↑	CHR$ &1C (or CHR$ 28)
↓	CHR$ &1D (or CHR$ 29)
→	CHR$ &1E (or CHR$ 30)
←	CHR$ &1F (or CHR$ 31)

Example:

```
10 ON KEY CHR$&01,CHR$&0D GO TO 200,300
 :
90 OFF KEY
```

In the above example, execution jumps to line 200 when definable key DEF1 is pressed, and to line 300 when the [ENTER] key is pressed.

4. When this statement is redeclared, the previous declaration becomes invalid.
5. This statement transfers execution to the specified line regardless of the location of the required key in the key input buffer. However, in the INPUT statement (page 117) or the KEYIN statement (page 165) wait state, execution is not transferred until after the wait state is ended.

Example:

When the INPUT statement is not in the wait state

```
 10 ON KEY "A" GO TO 1000
  :
100 INPUT X
  :
1000·····
```

If 123.45 [ENTER] A is keyed in during execution of line numbers 10 ~ 100, 123.45 is cleared and execution jumps to line 1000.

Example:

When the INPUT statement is in the wait state

```
 10···········
  :
100 ON KEY "A" GO TO 1000:INPUT X:OFF KEY
110 GO TO 100
  :
1000·········
```

If 123.45 [ENTER] A is keyed in during execution of line numbers 10 ~ 100, 123.45 is assigned to numeric variable X and execution jumps to line 1000.

6. When execution is transferred this by statement when the required key is entered, or by an ON ERROR statement (page 172), an OFF KEY statement (page 180) is automatically executed.
   However, when execution is returned from the subroutine to which execution was transferred by an ON ERROR statement, by using a RESUME statement (page 185), the ON KEY statement remains valid. (When a RETURN statement (page 136) is used for returning, the ON key statement becomes invalid.)

Application example:

[Program A] Roll a dice and guess the number. To guess 3 or less, press key A and, to guess a number over 3, press key B. To end execution, press key C.

```
10 ON KEY "A","B","C" GO TO 40,50,100
20 X=INT (RND 6)+1
30 GO TO 20
40 IF X<4 THEN 60:ELSE 80
50 IF X>3 THEN 60:ELSE 80
60 DISP "HIT"
70 GO TO 10
80 DISP "MISS"
90 GO TO 10
100 DISP "END" :END
```

- Jumps to line 40 when A is keyed in, to line 50 when B is keyed in, and to line 100 when C is keyed in.
- Selects the face of the dice by using random numbers.
- 3 or less is a win when A is keyed in.
- 4 or more is a win when B is keyed in.
- Redeclares ON KEY.
- Redeclares ON KEY.
- Execution ends when C is keyed in.

[Program B] Reprogram [program A] using subroutines.

```
10 ON KEY "A","B","C" GO SUB 40,70,100
20 X=INT (RND 6)+1
30 GO TO 20
40 REM SUB 1
50 IF X<4 THEN DISP "HIT":ELSE DISP "MISS"
60 RESUME
70 REM SUB 2
80 IF X>3 THEN DISP "HIT" :ELSE DISP "MISS"
90 RESUME
100 REM SUB 3
110 DISP "END" :END
```

- Jumps to the subroutine starting with line 40 when A is keyed in, to the subroutine starting with line 70 when B is keyed in, and to the subroutine starting with line 100 when C is keyed in.
- Even after returning from the subroutine, the ON KEY declaration remains valid.
- Even after returning from the subroutine, the ON KEY declaration remains valid.

OFF KEY

Releases the ON KEY statement.

Format: OFF KEY

(SYNTAX)



ON TIME

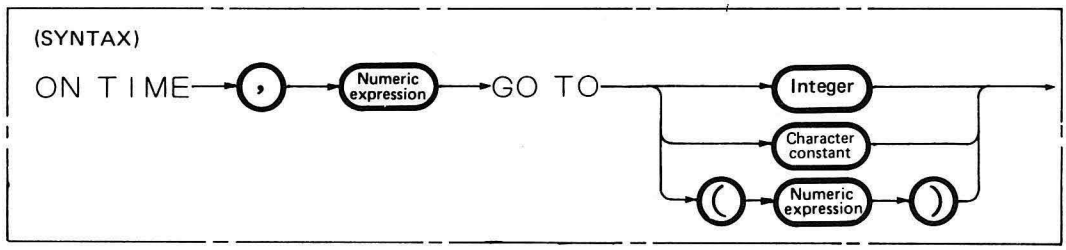
Jumps to the specified line at the specified time until an OFF TIME statement is executed.

〈Purpose of use〉
To stop execution or to execute other processing at the specified time.

● **Function 1:** Jumps using GO TO.

Format: ON TIME, X GO TO n | S | (Y)

X : Specified time (hour. minute second)
n : Line number of the line to be jumped to
S : Line label of the line to be jumped to
Y : Line number (numeric expression) of the line to be jumped to



Explanation:

1. Jumps to the line with line number n, line label S, or to the line number indicated by the value of numeric expression Y (with decimal part truncated) at the time indicated by the value of numeric expression X. The integer part indicates the hour (24-hour system) and the decimal part indicates the minute and second.
2. Execution jumps to the line described in 1. above at the end of the line being executed at the specified time.

Simple example:

1. `10 ON TIME,15.10 GO TO 100` ..... Jumps to line 100 at 15 : 10.
2. `20 ON TIME,A+0.3 GO TO "ABC"` ..... Jumps to line labeled ABC at the time indicated by numeric expression (A + 0.3). For example, if A = 12, jumps to the line labeled ABC at 12 : 30

● **Function 2:** Jumps to a subroutine

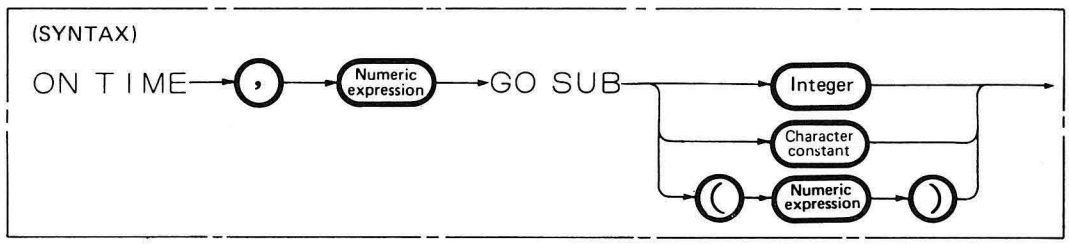
Format: ON TIME, X GO SUB n | S | (Y)

X : Specified time (hour. minute second)

n : Line number of the subroutine to be jumped to

S : Line label of the subroutine to be jumped to

Y : Line number (numeric expression) of the subroutine to be jumped to



Explanation:

1. Jumps to the subroutine starting at the line with line number n, line label S, or to the line number indicated by the value of expression Y (with decimal part truncated) at the time indicated by numeric expression X.
2. Jumps to the subroutine described in 1. Above at the end of the line being executed at the specified time. Execution returns from the subroutine to the line next to the line from which a jump was made.
   To return from the subroutine, execute a RESUME statement (page 185).

Simple example:

1. `10 ON TIME,11.5 GO SUB 100` ..... Jumps to the subroutine starting at line 100 at 11 : 50.
2. `20 ON TIME,A*2 GO SUB "ABC"` ..... Jumps to the subroutine starting with the line labeled ABC at the time indicated by the value of numeric expression (A * 2). For example, if A = 7, execution jumps to the subroutine starting at line labeled ABC, at 14 : 00 (2.p.m.).

Note

When this statement is redeclared, the previous declaration becomes invalid; the newest time declared is valid.

Application example:

Check if numbers 1 to 10000 can be displayed within the specified time.

```
10 INPUT "TIME=";X ..... Enter the time.
20 TIME=0
30 ON TIME,TIME+X GO TO 80 ..... Jumps to line 80 when the entered time is elapsed.
40 FOR I=1 TO 10000
50 DISP I; ..... Displays the number.
60 NEXT I
70 GO TO 90
80 DISP :DISP "TIME OVER" ..... Feeds new line and displays TIME OVER.
90 END
```

(Input data) 0.01 . . . . . 1 minute

```
>RUN
TIME=0.01
 1  2  3  4  5 .....
```

If 1 to 10000 could not be displayed within 1 minute, the following message is displayed.

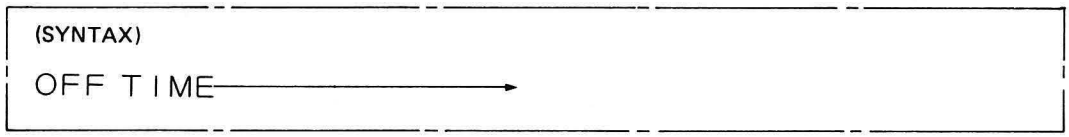
```
TIME OVER
READY
>
```

OFF TIME

Releases the ON TIME statement.

Format: OFF TIME

(SYNTAX)



RESUME

Ends the subroutine to which a jump was made by a statement starting with ON*) and returns the processing flow to normal.

〈Purpose of use〉
To end the subroutine to which a jump was made by a statement starting with ON and returns execution to the original program.

● **Function 1:** To return to the line being executed before jumping to a subroutine

Format: RESUME

(SYNTAX)

RESUME ——→

Explanation:

Ends the subroutine, and returns program execution to the beginning of the next line of the line being executed before jumping to the subroutine.

Simple example:

```
10 ON KEY "A" GO SUB 100
:
30 A=B+C  <--..Assume that key A was pressed during execution of this line.
40 DISP A                                  Return to
:                                          |
100 ..........  }                          |
:               } Subroutine               |
200 RESUME      }                          |
```

The specified key A was pressed during execution of line 30; execution jumped to the subroutine starting at line 100. After the subroutine was executed, it was ended by the RESUME statement on line 200 and execution returned to line 30.

● **Function 2:** Returns to the next line of the line to which execution would be returned by function 1.

Format: RESUME NEXT

(SYNTAX)

RESUME NEXT ——→

Explanation:

Ends the subroutine, and returns program execution to the beginning of the line next to the line to which execution is returned by function 1.

*) Statement starting with ON (subroutine jump): ON . . . GO SUB, ON ERROR, ON KEY and ON TIME statements

Simple example:

```
10 ON ERROR GO SUB 100
 :
30 A=SQR(X) ..... Assume that an error occurred during execution of this
40 B=SQR(-X) <--- line, and execution was transferred to the subroutine by
50 DISP A,B       an ON ERROR statement.
 :                                    Returned to
100 ..............
 :                  } Subroutine
200 RESUME NEXT
```

An error occurred during execution of line 30; execution jumped to the subroutine starting at line 100. After the subroutine was executed, it was ended by the RESUME statement on line 200 and execution returned to line 40.

Note

In a subroutine to which a jump was made by a statement starting with ON, the RETURN statement (page 136) may be used to end the subroutine and return to the original program instead of this statement. However, in this system, the use of the RESUME statement is recommended. The difference between the use of the RETURN statement and the RESUME statement is discussed in the "Helpful Hint" below.

< Detailed information >

In a program handling an ON KEY statement, the declaration of the ON KEY statement remains valid when a RESUME statement is used to return from the subroutine to which a jump was made by the ON KEY statement or by the ON ERROR statement. When a RETURN statement is used in such a case, the declaration of the ON KEY statement becomes invalid. (An OFF KEY statement (page 180) is automatically executed when jumping to the subroutine.)

MOVE

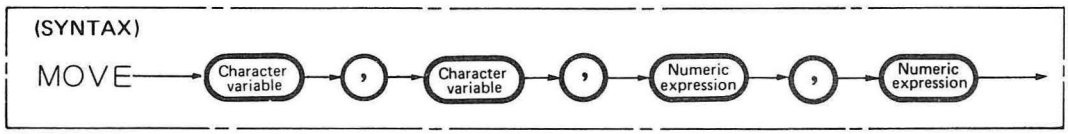
Assigns or concatenates a part of a character variable to another character variable, or copies the line at the specified line number.

〈Purpose of use〉
① To assign or concatenate one character or a series of characters in a character variable to another character variable.
② To generate a copy of a program by assigning different line numbers or to copy to different line numbers.

● **Function 1:** Assigns a part of a character variable to another character variable.

Format: MOVE A$, B$, X, Y

A$: Character variable to which characters are assigned
B$: Character variable containing characters to be moved
X : Position of the characters to be moved (first character position = 1)
Y : Number of characters to be moved



Explanation:

1. Assigns Y characters starting with the Xth character in character variable B$ to character variable A$.
2. This statement can process character variables each of which contains up to 256 characters. The character variables with 17 or more characters must be declared as a character array variable by a DIM statement (page 128).

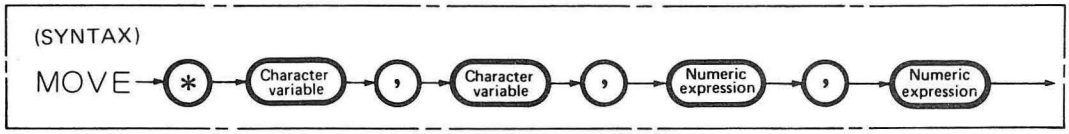
Simple example:

```
10 MOVE A$,B$,2,10
```
..... Assigns 10 characters starting with the second character in character variable B$ to character variable A$.

● **Function 2:** Concatenates a part of a character variable to another character variable.

Format: MOVE *A$, B$, X, Y

A$: Character variable to be concatenated
B$: Character variable containing characters to be moved
X : Position of the characters to be moved (first character position = 1)
Y : Number of characters to be moved



Explanation:

1. Concatenates Y characters starting with the Xth character in character variable B$ to the end of character variable A$.
2. This statement can process character variables each of which contains up to 256 characters.

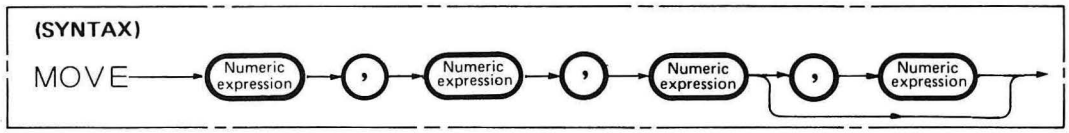
Simple example:

```
10 MOVE *A$,B$,2,10
```
..... Concatenates 10 characters starting with the 2nd character in character variable B$ to character variable A$.

● **Function 3:** Copies the line at the specified line number.

Format: MOVE U, V, X [, Y]

U : Copy start line number
V : Copy end line number
X : Start line number of the lines created by copying (X > V)
Y : Line numbers increment



Explanation:

1. Copies lines at line number U through line number V to the lines beginning at line number X with an increment of Y.
   The contents of the lines from line number U through line number V remain unchanged.
2. When the increment by Y of line number is omitted, the increment is automatically set by 10.
3. This statement can be used only in manual operation when the mode change switch is set to PRO mode. (This statement cannot be used in a program.)
   Execution of this statement in the OP mode results in an error (ERROR 34).

Simple example:

```
MOVE 30,100,150,10 [ENTER]
```
..... Copies the lines beginning from line number 30 up to and including line number 100 to the lines beginning at line number 150 with increment by 10.

Note

When lines are copied by function 3, the line numbers to be jumped to, etc., are not changed.

Application example:

[Program A] Extracts "SHARP" and "MODEL-3500" from the character string assigned to A$(1) and displays these characters.

```
10 DIM A$(1)*100
20 A$(1)="SHARP PERSONAL COMPUTER <MODEL-3500 SERIES>"
30 MOVE B$,A$(1),1,6
40 MOVE C$,A$(1),26,11
50 MOVE *B$,C$,1,10
60 DISP B$
70 END
```

Assigns 6 characters starting with the first character of A$ (1) to B$.

Assigns 11 characters starting with 26rd characters of A$ (1) to C$.

Concatenates the contents of C$ to the end of the contents of B$. (Assigns the concatenated result to B$.)

(Execution result)

```
>RUN
SHARP  MODEL-3500
READY
>
```

[Program B]

```
 10 READ A
 20 X=SIN A
 30 Y=COS A
 40 Z=TAN A
 50 DISP "sin=";X,"cos=";Y,"tan=";Z
 60 END
100 DATA 15,30,45,60,75
110 REM ************************
```

For the above program, in manual operation key MOVE 10,60,200,100 ENTER in the PRO mode.

(Execution result)

```
 10 READ A
 20 X=SIN A
 30 Y=COS A
 40 Z=TAN A
 50 DISP "sin=";X,"cos=";Y,"tan=";Z
 60 END
100 DATA 15,30,45,60,75
110 REM ************************
200 READ A
300 X=SIN A
400 Y=COS A
500 Z=TAN A
600 DISP "sin=";X,"cos=";Y,"tan=";Z
700 END
```

Line numbers 10 to 60 are copied.

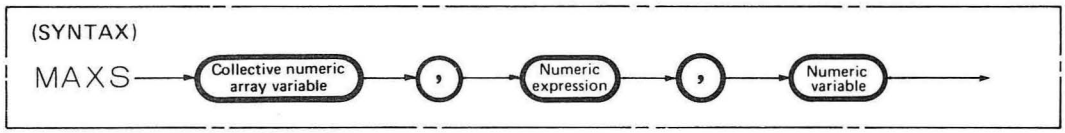
MAXS

Locates the element which has the maximum value in a numeric array variable.

〈Purpose of use〉
To see which element has the maximum value in a numeric array variable.

Format: MAXS A (*), X, B

A (*) : Generalized numeric array variable
X : Position of the element at which search is started
B : Variable for assigning the position of the maximum value



Explanation:

1. Starts searching for the maximum value from the element position indicated by the value of numeric expression X in generalized numeric array variable A (*) (indicates all the elements of numeric array variable A), and assigns the position of the element which has the maximum value to numeric variable B.
2. The element position means the ordinal number from the first element (= 1). Elements are numbered in ascending order of the subscript. (See the DIM statement (page 128).)
3. An error occurs when the value of numeric expression X is equal to or less than 0, or greater than the number of array elements of numeric array variable A.
4. If more than one element has the maximum value in array variable to be searched, the first such element position is assigned to numeric variable B.

Simple example:

```
50 MAXS A(*),3,B
```

Assigns the position of the element which has the maximum value begining the third element in numeric variable A to numeric variable B. For example, if the value of A(10) is the maximum value beginning the third element, 10 is assigned to B.

Application example:

Program to obtain the maximum value of the input numeric values and print it out

```
10 DIM N(5)
20 FOR J=1 TO 5
30 INPUT N(J)
40 PRINT "NO";J;"=";N(J)
50 NEXT J
60 MAXS N(*),1,A
70 PRINT "MAXIMUM VALUE NO";A,"MAXIMUM VALUE=";N(A)
80 END
```

When 12, -56, 38, 9, and 1 are entered,
the following print-out is obtained.

```
NO 1= 12
NO 2=-56
NO 3= 38
NO 4= 9
NO 5= 1
MAXIMUM VALUE NO 3   MAXIMUM VALUE= 38
```

When all the PRINT statements in this program are changed to DISP statements, the result is displayed on the CRT.

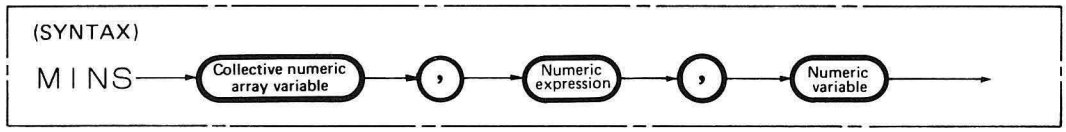
MINS

Locates the element which has the minimum value in a numeric array variable.

〈Purpose of use〉
To see which element has the minimum value in a numeric array variable.

Format: M I NS A (*), X, B

A (*) : Generalized numeric array variable
X : Position of the element at which search is started
B : Variable for assigning the position of the minimum value



Explanation:

1. Starts searching for the minimum value from the element position indicated by the value of numeric expression X in generalized numeric array variable A (*) (indicates all the elements of numeric array variable A), and assigns the position of the element which has the minimum value to numeric variable B.
2. For other details, see the MAXS statement (page 190).

Simple example:

```
50 MINS D(*),2,E
```

Assigns the position of the element which has the minimum value beginning the second element in numeric variable D to numeric variable E.
For example, if the value of D (5) is the minimum value beginning the second element, 5 is assigned to E.

Application example:

Program to read data given by the DATA statement (page 120) and display the minimum value on CRT

```
 10 DIM M(7)
 20 FOR I=1 TO 7
 30 READ M(I)
 40 NEXT
 50 MINS M(*),2,N
 60 DISP M(N)
 70 END
100 DATA -8,-2,1,3,-7,6,10
```

(Execution result)

```
>RUN
-7
READY
>
```

In this program, the fifth element has the minimum value beginning the second element in numeric array variable M so that 5 is assigned to variable N.

SEARCH

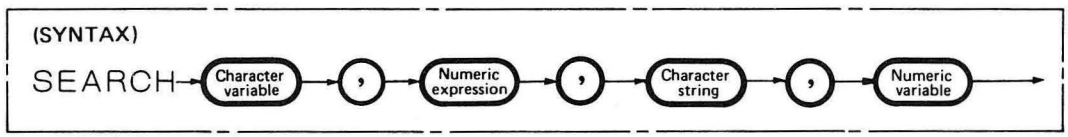
Searches for the specified character string in a character variable (including a character array variable), and indicates its position.

〈Purpose of use〉
① To see the position of a specific character string in a variable which stores very long character data.
② To see the position of the element which stores a specific name when each element of an array variable stores a name.

● **Function 1:** Searches for the specified character string.

Format: SEARCH A$, X, C, A

A$: Character variable (Character array variable) to be searched for
X : Position of the character where search is started (the first character position = 1), or the position of the array element when A$ is in A$ (*) format
C : Character string to be searched for
A : Numeric variable to which the position of character string C is to be assigned



Explanation:

1. Searches for the first character string C beginning the position indicated by the value of numeric expression X in character variable A$ and, if it is found, assigns its position (the beginning of the character string) to numeric variable A, when character variable A$ is a simple character variable or a single-element character array variable (example: A$ (5, 6)). (The search is done character by character.)
   If character string C is not found, the character constant assigned to character variable A$ is shorter than character string C, or a null string (" ") alone is assigned to character variable A$, 0 is assigned to numeric variable A. When character string C is a null string (" "), an error (ERROR 13) occurs.
2. Searches for the first character string C beginning element X in A$ (*) and, if it is found, assigns the position of the element which has character string C to numeric variable A when character variable A$ is in the A$ (*) format (generalized character array variable).
   The search is made in the order of the elements in the array. (See the DIM statement (page 128).)
   If character string C is not found, 0 is assigned to numeric variable A.

* For actual examples, see application example (program B).

Simple example:

1. `SEARCH A$,2,"COMPUTER",A` ..... Searches for character string COMPUTER beginning the second character in character variable A$, and assigns the position where the character string was found to numeric variable A.

For example, if A$ = "SHARP COMPUTER", A = 7 and, if
A$ = "COMPUTER", A = 0.

2. `SEARCH A$(*),1,"GEORGE",A` ..... Searches for character string GEORGE beginning the first element in generalized character array variable A$ (*), and assigns its position to numeric variable A.

For example, if A$ (1) = "JOHN", A$ (2) = "ANDREWS", A$ (3) = "GILBERT", A$ (4) = "WILLIAM", and A$ (5) = "GEORGE", A= 5.

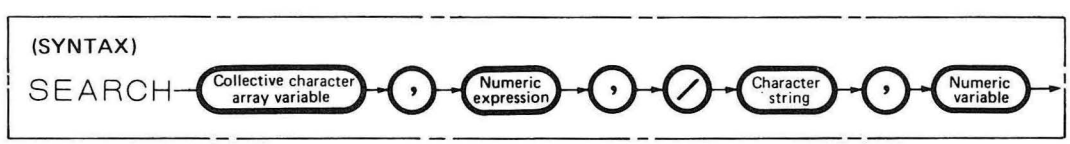
● **Function 2:** Searches for the character string containing the specified character at the specified position.

Format: SEARCH A$ (*), X, /C, A

A$ (*) : Generalized character array variable to be searched
X : Position in variable array from which the search is to be made. (first element = 1)
C : Character string which specifies the character and its relative position. Let the character to be searched for be A, then character string C is as follows:
"A –" : Character string starting with A
"–A " : Character string ending with A
"–A–" : Character string including A
"*A*" : Character string of three characters whose second character is A
"*A–" : Character string whose second character is A
"–A*" : Character string whose second to last character is A

Where – indicates all character string including a null string and * indicates a character. Thus the specification can be expanded by writing any number of * s consecutively, as follows.
"***A**" (Character string of six characters whose fourth character is A)
"***A–" (Character string whose fourth character is A)
" –A**" (Character string whose third to last character is A)

A : Numeric variable to which the position of character C is assigned



Explanation:

Starts searching from element X in generalized character array variable A$ (*), and assigns the position of the first element where the character which satisfies character string C was found. For array element search sequence, see Function 1.

If such a character string was not found, 0 is assigned to numeric variable A.

Simple example:

1. `20 SEARCH A$(*),1,/"W-",A` ..... Searches for a character string which starts with W in generalized character array variable A$ (*), and assigns the position of the element where the character string was first found to numeric variable A.

   For example, if A$ (1) = "JOHN", A$ (2) = "ANDREWS", A$ (3) = "GILBERT", A$ (4) = "WILLIAM", A$ (5) = "GEORGE", A = 4.

2. `20 SEARCH A$(*),1,/"-S",A` ..... Searches for a character string which ends with S in generalized character array variable A$ (*), and assigns the position of the element where the character string was found to numeric variable A.

   When the A$ (*) data of simple example 1 is used, A = 2.

3. `20 SEARCH A$(*),1,/"*N-",A` ..... Searches for a character string which contains N as the second character in generalized character array variable A$ (*), and assigns the position where the character string was found to numeric variable A.

   When the A$ (*) data of simple example 1 is use, A = 2.

4. `20 SEARCH A$(*),1,/"**H*",A` ..... Searches for a character string of four characters whose third character is H, and assigns the position where the character string was found to numeric variable A.

   When the A$ (*) data of simple example 1 is used, A = 1.

Application example:

[Program A] Converts a hexadecimal notation to decimal.

```
10 DIM A$(1)*16
20 A$(1)="123456789ABCDEF0"
30 INPUT X$
40 X=LEN X$
50 IF X>4 THEN GO TO 30 ···· The input character data is limited to four characters.
60 DISP X$;
70 N=0
80 FOR I=1 TO X
90 MOVE C$,X$,I,1 ···· Assigns one character of the input character data to C$.
100 SEARCH A$(1),1,C$,A ···· Assigns the position where C$ was found in A$ (1) to A.
110 IF A=0 THEN DISP "Error":END ···· If C$ was not found in A$ (1), execution ends.
120 IF A=16 THEN A=0 ···· When C$ is 0
130 N=N+A*16^(X-I) ···· Hexadecimal notation-to-decimal conversion equation
140 NEXT I
150 DISP N
160 END
```

(Input data) 1A4E (hexadecimal notation)

(Execution result)

```
>RUN
1A4E
1A4E 6734 ···· The decimal conversion of hexadecimal notation 1A4E i.e., 6734, is displayed.
READY
>
```

[Program B] Indicates the position of the element which contains data E in an array variable

```
10 DIM X$(2,3)
20 X$(1,1)="A":X$(1,2)="B":X$(1,3)="C"
30 X$(2,1)="D":X$(2,2)="E":X$(2,3)="F"
40 SEARCH X$(*),1,"E",X ···· Assigns the position of the element containing E in X$ (*) to X.
50 DISP "X=";X
60 END
```

(Execution result)

```
>RUN
X= 5
READY
>
```

DEFFN

Defines function FN.

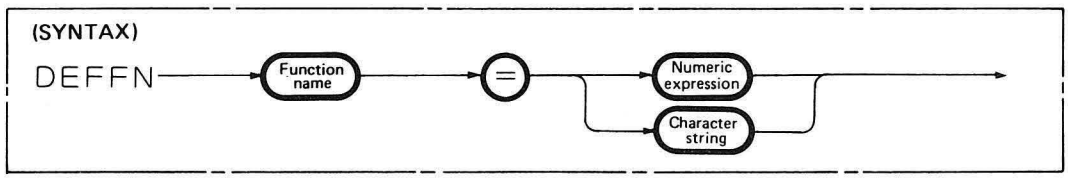
〈Purpose of use〉
To define a combination of several numeric expressions or character strings as a function.

● **Function 1:** Without argument

Format: DEFFN SF = X | C

SF : Function name

X or C : Function definition



Explanation:

Defines the contents of numeric expression X or character C as a function with function name SF. By defining functions at the beginning of a program, they can be used in the same way as character processing functions (page 23) and numeric processing functions (page 26). Function SF is written as FN SF in a program.

Simple example:

1. 10 DEFFN A=8^10·····Defines numeric expression (8∧10) as FN A.
2. 10 DEFFN B$="SHARP"·····Defines character string "SHARP" as FN B$.

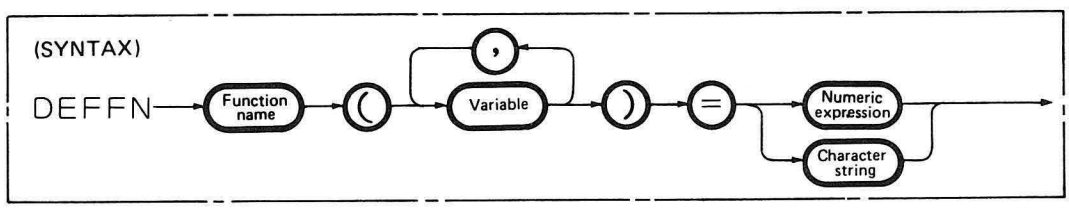
● **Function 2:** With arguments

Format: DEFFN SF ((N)) = X | C

SF : Function name

N : Argument

X or C : Function definition



Explanation:

Defines numeric expression X or character string C with arguments (N) as function SF when an arguments are contained in numeric expression X or character string C.

Simple example:

1. DEFFN A(X,Y)=X^Y ..... Defines numeric expression (X∧Y) with arguments X and Y as function A.
2. DEFFN B$(C$)="Good-"+C$ ..... Defines character string "Good-" +C$ with argument C$ as function B$.

Note

1. This statement cannot be labeled.
2. The format of the argument in this statement and that in FN must be the same.
3. Up to total of 42 DEFFN statements and subprograms SUBs (page 160) can be used in one program.
4. The variables in FN are independent of the variables in the program.

Application example:

[Program A] Defines a numeric expression with arguments

```
10 DEFFN A(X,Y)=X^Y ..... Defines function A (X, Y).
20 INPUT X,Y
30 M=FN A(X,Y) ..... Assigns the calculation result of (X∧Y) to numeric variable M.
40 DISP "X=";X;"Y=";Y,"X^Y=";M
50 END
```

(Input data) 5, 3

(Execution result)

```
>RUN
5                    3
X= 5 Y= 3            X^Y= 125
READY
>
```

The same execution result will be obtained by changing

20 INPUT X,Y ------→ 20 X=5:Y=3

30 M=FN A(X,Y) -----→ 30 M=FN A(5,3)

in program A (input data is not needed then).

[Program B] Defines a function by a character string with arguments

```
 10 DIM A$(3)
 20 A$(1)="Morning"
 30 A$(2)="Afternoon"
 40 A$(3)="Night"
 50 DEFFN B$(X$)="Good-"+X$ ..... Defines function B$.
 60 INPUT N
 70 IF N<1 THEN GO TO 200
 80 IF N>3 THEN GO TO 200
 90 X$=A$(N) ..... Assigns the result of "Good-"+A$ (N) to character variable C$.
100 C$=FN B$(X$)
110 DISP C$
120 GO TO 60
200 END
```

(Input data) 1, 2, 3

(Execution result)

```
>RUN
1
Good-Morning
2
Good-Afternoon
3
Good-Night
0
READY
>
```

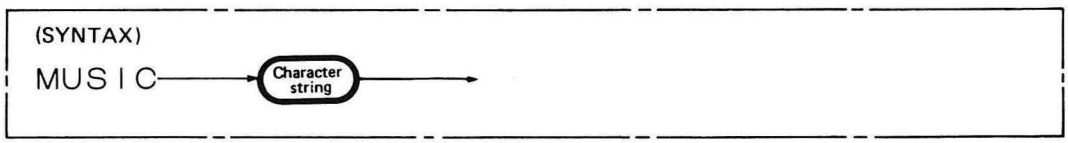
MUSIC

Plays the melody or sound effect specified by a character string through a speaker.

〈Purpose of use〉
① To use sound effects in a program.
② To play music.

Format: MUS I C C

C : Character string



Explanation:

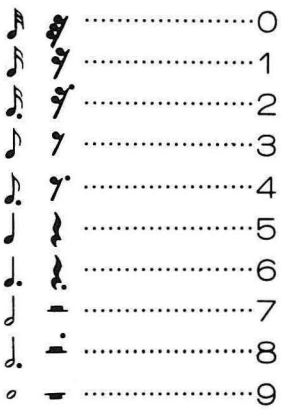
1. Plays the sound at the tempo and with the melody specified by character string C through a speaker.
   First specify the tempo, then the melody.
   Specify each note and rest in the order of length and pitch.
   Specify each element as follows.

i) Tempo(Specify this at the beginning of the character string.)

♩= 200·····T1
♩= 133·····T2
♩= 100·····T3
♩= 80·····T4
♩= 67·····T5
♩= 57·····T6
♩= 50·····T7

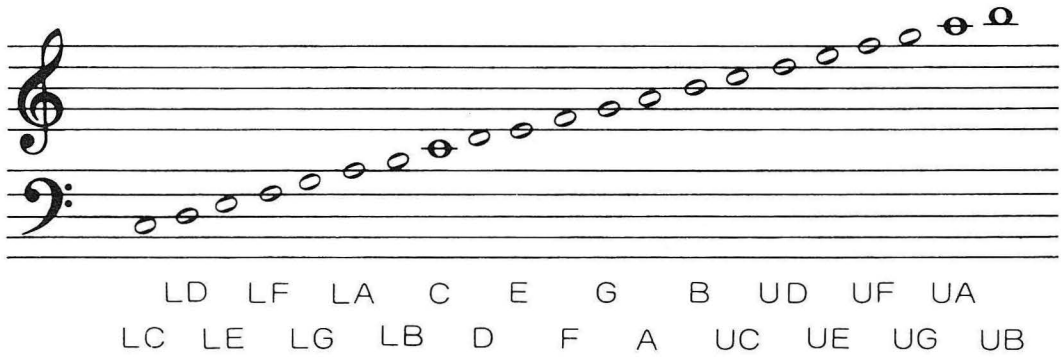
When omitted, T4 (♩ = 80) is used by default.

ii) Length of note and rest (Specify this before the pitch of note or rest.)



When notes (rests) of the same length follow, the specification of the length may be omitted for the second note (or rest) and the subsequent ones.

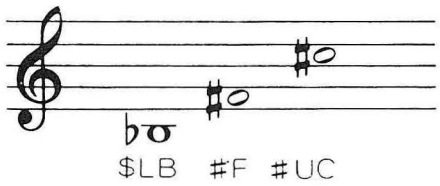
iii) Pitch of tone (Specify this following the length of note)



iv) Half step

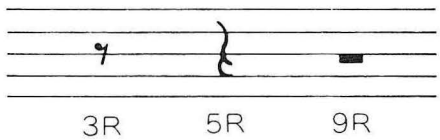
(Sharp) . . . .

b (Flat) . . . . . $



v) Rests

Specify rests by suffixing the number which indicates the length of a rest with R.



2. Up to 255 characters can be specified in a character string. To compose a piece of music longer than this, divide it into two or more statements.
3. The volume of the sound emitted from the speaker can be controlled by the VOLUME knob at the bottom left of the front panel.

Note

Up to 250 characters can be written on one line; music exceeding this limit cannot be composed in one line.

WAIT

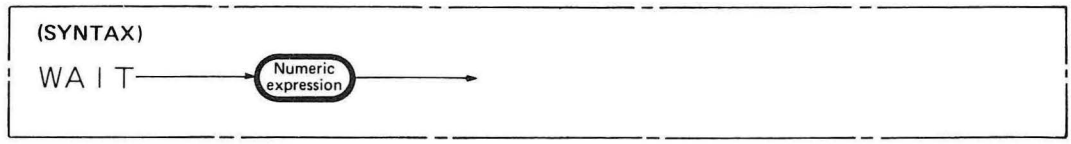
Suspends (waits) program execution for the specified time interval until the specified time.

〈Purpose of use〉
① To output the intermediate result of calculation to the CRT temporarily.
② To execute a program from the specified time.

● **Function 1:** Suspends execution for the specified time interval.

Format: WAIT X

X: Wait time (unit: 0.1 second)



Explanation:

Suspends program execution for the number of seconds indicated by 1/10 of the value of numeric expression X (X x 0.1 seconds).

Simple example:

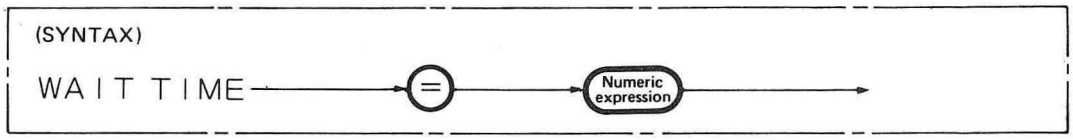
```
10 WAIT 30
```
..... Suspends program execution for 3 seconds.

WAIT 1 means to wait for 0.1 second.

● **Function 2:** Suspends execution until the specified time.

Format: WAIT TIME = X

X: Time (hour.minute second) when wait state is released



Explanation:

Suspends program execution until the time indicated by the value of numeric expression X. Specify the time with a real number whose integer part indicates the hour (24-hour system) and whose decimal part indicates the minute and second.

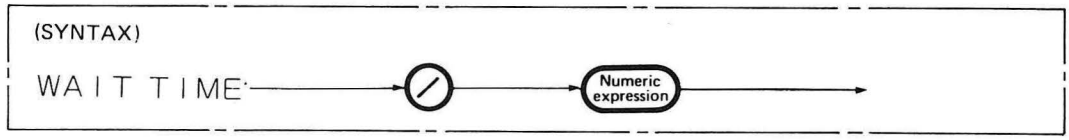
Simple example:

```
10 WAIT TIME=17.30
```
..... Suspends program execution until 17:30.

● **Function 3:** Suspends execution until the specified time. (Outputs the current time.)

Format: WAIT TIME / X

X: Time (hour, minute, second) when wait state is released



Explanation:

Outputs the current time (hour:minute:second) from the current cursor position while suspending program execution until the time indicated by the value of numeric expression X. The current time is displayed in the 12-hour system with AM or PM indication.

Simple example:

```
10 WAIT TIME/10.0935 ..... Outputs the current time while suspending
                           program execution until 10:09:35.
```

Note

This statement is invalid when a time earlier than the current time is specified. Set the correct time before using this statement.

<Helpful hint>

To cancel the execution of a wait effected by this statement, press the HALT button and halt program execution.

Application example:

Displays number 1 to 10 at 1 second intervals.

```
 10 DISP CLEAR ..... Clears the screen.
 20 N=0
 30 N=N+1
 40 IF N>10 GO TO 100
 50 CURSOR 20,20 ..... Positions the cursor at coordinates (20, 20)
 60 DISP N ..... Displays the number.
 70 WAIT 10 ..... Suspends execution for 1 second.
 80 GO TO 30
100 END
```

DEFINT

Declares integer-type numeric variables.

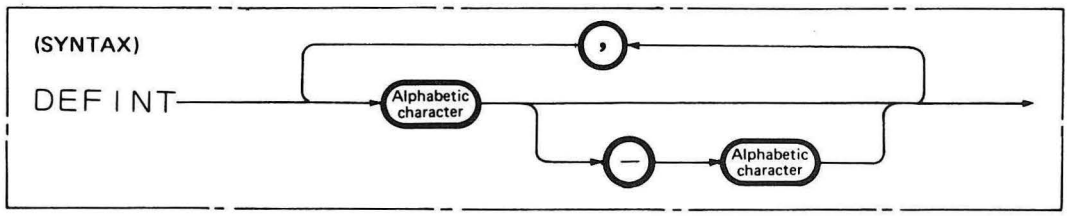
〈Purpose of use〉

① To define variables as integer-type numeric variables without suffixing the variable names with %.

② To define all the variables starting with the specified character as integer-type numeric variables.

Format: DEFINT [CH [-DH]]

CH (and DH): Alphabetic character which indicates the beginning of the variable name



Explanation:

1. Declares all the variables which begin with character CH or a character in the range of CH to DH as integer-type numeric variables (simple variables and array variables) when a capital letter is specified as CH (and DH).
2. When a small letter is specified as CH, declares all the variables which begin with a small letter as integer-type numbeic variables. In this case, DH need not be used.

Simple example:

1. `10 DEFINT A` ..... Declares all variables which begin with A as integer-type numeric variables.
2. `10 DEFINT I-N` ..... Declares all variables which begin with I, J, K, L, M, or N as integer-type numeric variables.
3. `10 DEFINT a` ..... Declares all variables which begin with a small letter as integer-type numeric variables.
4. `10 DEFINT I,L-N,a` ..... Declares variable which begin with I, L, M, N, or a small letter as integer-type numeric variables.

Note

1. Sepecify any small letter to declare variables which begin with any small letter.
2. All variables declared by this statement are handled as integer-type numeric variables without suffixing them with %.
3. The type declaration using #, $, and @ has priority over the variable declaration defined by this statement.
   For example, I# is handled as a real-type numeric variable and J$ and M@ are handled as character variables although they are declared as integer-type numeric variables by DEFINT I – N.

DEFREAL

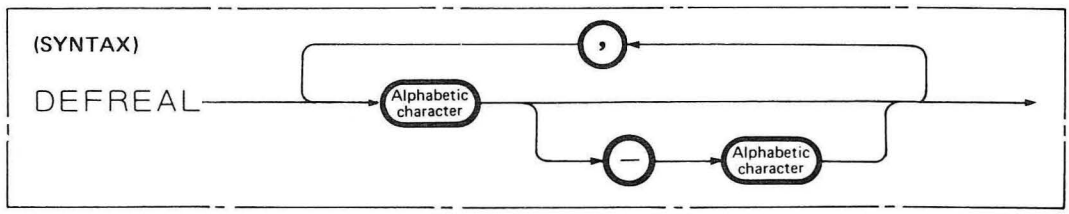
Declares real-type numeric variables.

〈Purpose of use〉

To define all the variables beginning with the specified character as real-type numeric variables.

Format: DEFREAL [CH [-DH]]

CH [and DH] : Alphabetic character which indicates the beginning of the variable name



Explanation:

Declares all variables which begin with the specified character as real-type numeric variables (simple variables and array variables).

For details, see the DEFINT statement (page 205).

Simple example:

1. `10 DEFREAL A-H` ..... Declares all variables which begin with A, B, C, D, E, F, G, and H as real-type numeric variables.
2. `10 DEFREAL a` ..... Declares all variables which begin with a small letter as real-type numeric variables.
3. `10 DEFREAL A-C,Q,X-Z` ..... Declares all variables which begin with A, B, C, Q, X, Y, and Z as real-type numeric variables.

Note

See the notes on the DEFINT statement.

DEFSTR

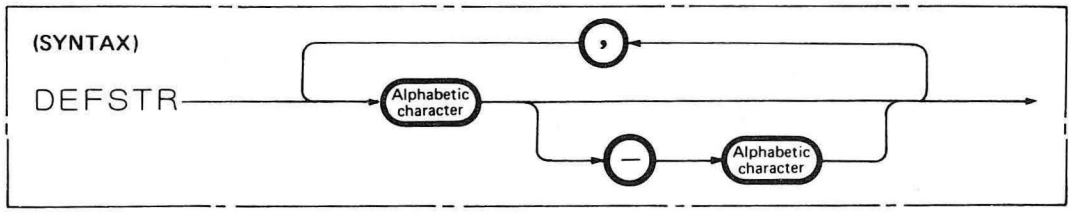
Declares character variables.

〈Purpose of use〉
1. To define variables as character variables without suffixing the variable names with $ or @.
2. To define all the variables starting with the specified character as character variables.

● **Function 1:** Declares $-type character variables.

Format: DEFSTR [CH [-DH]]

CH (and DH): Alphabetic character which indicates the beginning of the variable name



Explanation:

Declares all variables which begin with the specified character as ($-type) character variables (simple variables and array variables).

For details, see the DEFINT statement (page 205).

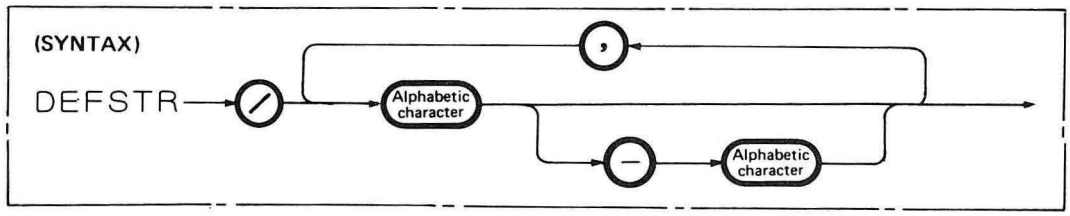
Simple example:

1. `10 DEFSTR C` ..... Declares all variables which begin with C as ($-type) character variables.
2. `10 DEFSTR A-Z` ..... Declares variables which begin with a capital letter A to Z as ($-type) character variables.

● **Function 2:** Declares @-type character variables.

Format: DEFSTR / [CH [-DH]]

CH (and DH): Alphabetic character which indicates the beginning of the variable name



Explanation:

Declares all variables which begin with the specified character as @-type character variables (simple variables or array variables).

For details, see the DEFINT statement.

Simple example:

1. `10 DEFSTR/h` ..... Declares all variables which begin will small letter as @-type character variables.
2. `10 DEFSTR/A-C,H,e` ..... Declares all variables which begin with A, B, C, H, or small letter as @-type character variables.

Note

See Notes on the DEFINT statement (page 205).

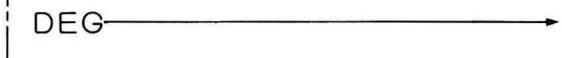
DEG

Specifies the unit of values used in trigonometric functions in degree (°).

〈Purpose of use〉
To specify the angles in trigonometric functions not in radians but in degrees.

Format: DEG

(SYNTAX)



Explanation:

1. Specifies the unit of values used in trigonometric functions to degree (°).
   This statement is effective when the unit was set to radians by the RAD statement (page 210).
2. In the normal state (after power ON), the unit is automatically set to degrees (°).

Application example:

Obtains the sine value of angle 30°.

```
10 DEG ..... Sets the unit of angles to degrees (°).
20 A=SIN 30 ..... Calculates sin 30°.
30 DISP "sin 30° =";A
40 END
```

(Execution result)

```
>RUN
sin 30° = 0.5
READY
>
```

RAD

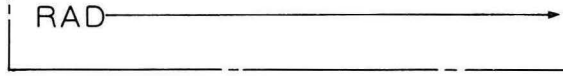
Specifies the unit of values used in trigonometric functions in radians.

〈Purpose of use〉

Specifies the angles in trigonometric functions not in degrees (°) but in radians.

Format: RAD

(SYNTAX)



Explanation:

Specifies the unit of values used in trigonometric functions in radians. When the unit is set to degrees (°) after power ON or after execution of a DEG statement (page 209), the unit is changed to radians by executing this statement.

Application example:

Obtains the sine value of angle $\pi/6$ in radians.

```
10 RAD ..... Specifies the unit of angles as radians.
20 A=SIN (PI/6) ..... Calculates sin (π/6) (radian) (PI = π)
30 DISP "sin(π/6)=";A
40 END
```

(Execution result)

```
>RUN
sin(π/6)= 0.5
READY
>
```

RANDOM

Generates a random number based on the initial random value generated after power ON.

〈Purpose of use〉
To use numeric processing function RND (page 29) for random number generation.

Format: RANDOM

(SYNTAX)



Explanation:

1. Generates a random number using the initial random value generated after power ON. This statement sets a random number as the initial value of random number generation, the use of numeric processing function RND is recommended in this state.
2. This statement can be used both in manual operation and in a program.

Application example: [Program A]
Displays one of the numbers from 0 to 9.

```
10 RANDOM ..... Sets the initial value for random number genration.
20 FOR I=1 TO 10 ..... Generates a random number less than 10, and extracts the
30 A=INT (RND 10)      integer part of the generated random number.
40 DISP A ..... Indicates the integer part which was extracted.
50 NEXT I
60 END
```

(Execution result) Displays 10 random numbers 0 to 9. If line 10 is deleted and the RANDOM statement is omitted, the results of random number generation will always be the same after power ON.

[Program B] Reverse-displays the entire screen, and paints up the center part.

```
10 DISP CLEAR
20 RV 0,0,2000
30 X=INT (RND 80)+40
40 Y=INT (RND 25)+12
50 SET X,Y
60 GO TO 30
```

ERROR MUSIC

Specifies whether or not to emit an error sound when an error occurs.

〈Purpose of use〉
When an error occurs, a warning tone is emitted and an error message is displayed on the CRT.
This statement is used to turn off the warning tone.

● **Function 1:** Turns off the error sound emitted when an error occurs.
Format: ERROR MUSIC OFF

(SYNTAX)
ERROR MUSIC OFF——→

Explanation:

When an error occurs in the initial state after power ON, a warning tone is emitted. After this statement is executed, no warning tone is emitted when an error occurs.

● **Function 2:** Resets the error sound indication.
Format: ERROR MUSIC ON

(SYNTAX)
ERROR MUSIC ON——→

Explanation:

Error sound is turned on again if this statement is executed when the error sound has been turned off. If the error sound was not turned off, this statement is invalid. (It does not result in an error.)

5. CRT DISPLAY (PRINTER OUTPUT) CONTROL STATEMENTS

DISP CLEAR
SET
RESET
POS
ENTER
CURSOR
MVCUR
LINE
BL
RV
RE
TABLE
SCROL
CHANGE DISP
CHANGE PRINT
COLOR

This section describes the above statements.
All these statements can be used in manual operation.
Except for the DISP CLEAR, CURSOR function 1, and MVCUR statement, the cursor position is not changed before and after execution of the CRT display statements.

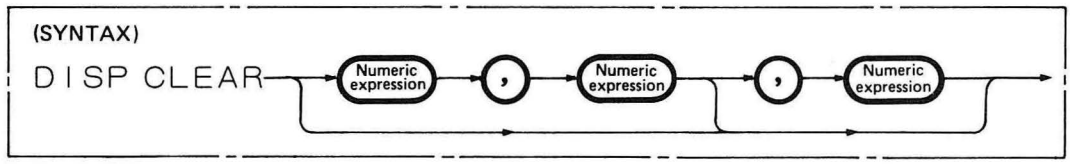
DISP CLEAR

Clears the character displayed on the CRT.

〈Purpose of use〉
To make effective use of the CRT, and to give legibility.

Format: DISP CLEAR [X, Y [, Z]]

X: X-coordinate of the first character to be cleared
Y: Y-coordinate of the first character to be cleared
Z: Number of characters to be cleared

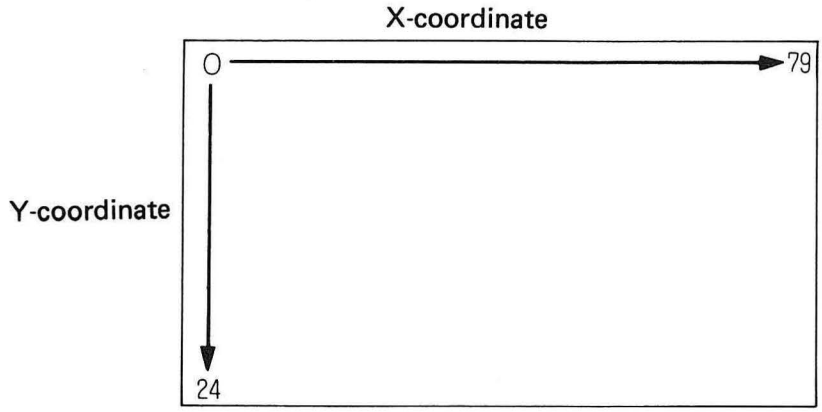


Explanation:

1. Clears the number of characters specified by Z from text coordinate position (X, Y). After the characters are cleared, the cursor is set to the position of the first character cleared. The ranges of numeric expressions X and Y are shown below.
2. Clears all the characters specified from text coordinate position (X, Y) when Z is omitted. After the characters are cleared, the cursor is positioned at the beginning character cleared.
3. When coordinates (X, Y) and number of characters Z are omitted, all characters on the CRT are cleared. After the screen is cleared, the cursor returns to the home position (at text coordinate position (0, 0)).

- 80 x 25 mode
  $0 \leqq X \leqq 79$ (X: 0 to 79)
  $0 \leqq Y \leqq 24$ (Y: 0 to 24)
- 80 x 20 mode
  $0 \leqq X \leqq 79$ (X: 0 to 79)
  $0 \leqq Y \leqq 19$ (Y: 0 to 19)
- 40 x 25 mode
  $0 \leqq X \leqq 39$ (X: 0 to 39)
  $0 \leqq Y \leqq 24$ (Y: 0 to 24)
- 40 x 20 mode
  $0 \leqq X \leqq 39$ (X: 0 to 39)
  $0 \leqq Y \leqq 19$ (Y: 0 to 19)

The (X, Y) coordinate setting in the 80 x 25 mode is shown below.



Simple example:

1. `10 DISP CLEAR 0,4,80` ..... Clears 80 characters from text coordinates (0, 4).
2. `10 DISP CLEAR 0,10` ..... Clears all the characters from text coordinates (0, 10).
3. `10 DISP CLEAR` ..... Clears the entire screen.

Note

This statement cannot clear graphics (see page 31), ruler lines (see page 225), blinking displays (see page 227), reverse displays (see page 228), and color displays (see page 237). Hold CTRL key and press HOME key to clear ruler line, blinking, reverse, and color displays after such a display is cleared, the cursor returns to the home position.

For clearing graphics, see the CLS statement (page 244).

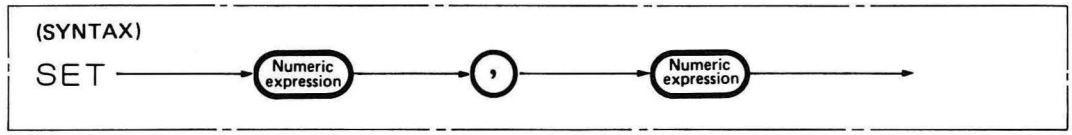
SET

Turns on (sets) the point at the specified coordinates on the CRT.

〈Purpose of use〉
To set points or lines on the text screen without using the graphic screen.

Format: SET X, Y

X: X-coordinate
Y: Y-coordinate

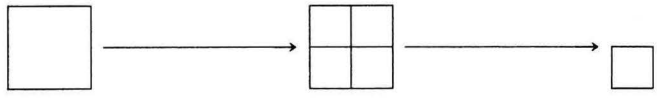


Explanation:

Turns on (sets) the point at coordinates (X, Y). When the values of numeric expressions X and Y are not integers, the decimal part is truncated. The ranges of expressions X and Y are as shown below.

- 80 x 25 mode
  $0 \leqq X \leqq 159$ (X: 0 – 159)
  $0 \leqq Y \leqq 49$ (Y: 0 – 49)
- 80 x 20 mode
  $0 \leqq X \leqq 159$ (X: 0 – 159)
  $0 \leqq Y \leqq 39$ (Y: 0 – 39)
- 40 x 25 mode
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 49$ (Y: 0 – 49)
- 40 x 20 mode
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 39$ (Y: 0 – 39)

Unlike the normal text coordinates, the unit of coordinates in the SET statement is 1/4 size of a character; the range of X-and Y-coordinate values is double the text coordinate range.



Text coordinate unit (one character)

Coordinate unit in SET statement

Simple example:

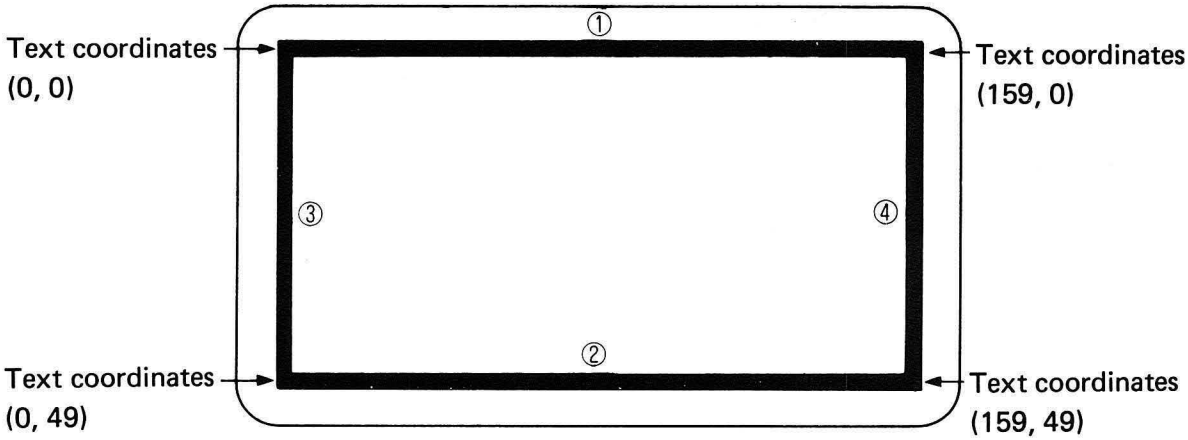
```
10 SET 4,6
```
·····Turns on the point at coordinates (4, 6).

Application example:

Draws a rectangle along the edges of the screen.

```
10 FOR X=0 TO 159
20 SET X,0 ..... Draws the upper side of the rectangle with a FOR/NEXT loop.
30 SET X,49 ..... Draws the lower side of the rectangle with a FOR/NEXT loop.
40 NEXT X
50 FOR Y=0 TO 49
60 SET 0,Y ..... Draws the left side of the rectangle with a FOR/NEXT loop.
70 SET 159,Y ..... Draws the right side of the rectangle with a FOR/NEXT loop.
80 NEXT Y
90 END
```

(Execution result)



Lines are drawn in the order ①, ②, ③, and ④.

RESET

Turns off (resets) the point at the specified coordinates on the CRT.

〈Purpose of use〉

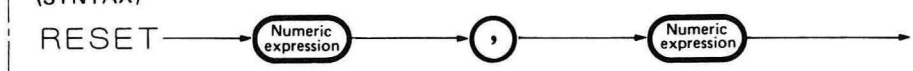
To erase the points or lines drawn by the SET statement.

Format: RESET X, Y

X: X-coordinate

Y: Y-coordinate

(SYNTAX)



Explanation:

Turns off the point (resets the point) at coordinates (X, Y) which was turned on by the SET statement. For coordinate specifications and other details, see the SET statement. This statement does not affect the coordinates which are not set.

POS

Assigns the coordinates of the current cursor position to the specified variables.

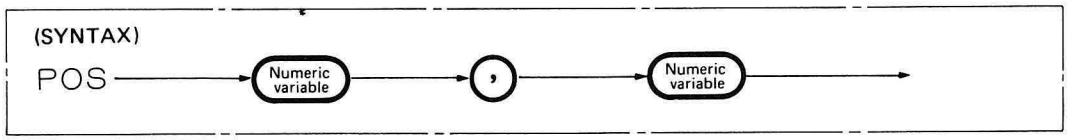
〈Purpose of use〉

To format the screen display based on the previous screen output in conjunction with the CURSOR statement.

Format: POS A, B

A: Numeric variable to which X-coordinate value is assigned

B: Numeric variable to which Y-coordinate value is assigned



Explanation:

Assigns the X-coordinate (text coordinate) of the current cursor position to numeric variable A, and the Y-coordinate (text coordinate) to numeric variable B.

Simple example:

```
10 POS M,N.....
```

Assigns the X-coordinate value of the current cursor position to variable M, and Y-coordinate value to variable N.

Application example:

Underlines the character string SHARP CORPORATION.

```
10 POS X,Y:DISP "SHARP CORPORATION"
20 CURSOR X,Y+1:DISP "──────────────────"
30 END
```

ENTER

Assigns the characters displayed on the CRT to the specified character variable.

〈Purpose of use〉

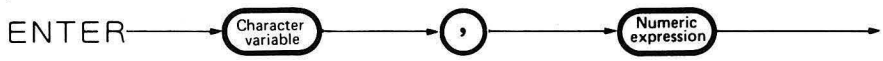
To use the characters displayed on the CRT in the program or to output them to the printer.

Format: ENTER A$, X

A$: Character variable

X: Number of characters to be assigned

(SYNTAX)



Explanation:

1. Assigns the number of characters specified by numeric expression X from the character starting with the current cursor position.
2. The size of the character variable A$ is determined by counting one ASCII character as 1 byte.

Simple example:

```
10 ENTER A$,B ..... Assigns the number of characters specified by the value
                    of the numeric variable B from the character starting with
                    the current cursor position.
```

For example, let ABCDEFG be displayed starting with the current cursor position, then four characters ABCD are assigned to character variable A$ when numeric variable B = 4 is specified.

Application example:

Outputs all the characters displayed on the CRT to printer.

```
   10 DIM A$(1)*2000 ..... Declares the array variable to input the CRT display
                           contents.
    :
    :
 1000 CURSOR 0,0 ..... Returns the cursor to the home position.
 1010 ENTER A$(1),2000 .... Assigns all characters on the CRT to variable A$ (1).
 1020 PRINT A$(1) ..... Outputs the contents of variable A$ (1) to the printer.
 1030 END
```

Note

1. Display characters include the ruler lines drawn by pseudo graphic symbols, but they do not include the ruler lines.
2. This statement cannot read the graphic information on the CRT into the specified character variable.

   To read the graphic information, use the GENTER statement (page 264) explained later.

CURSOR

Determines the cursor position, or turns the cursor display ON/OFF during program execution.

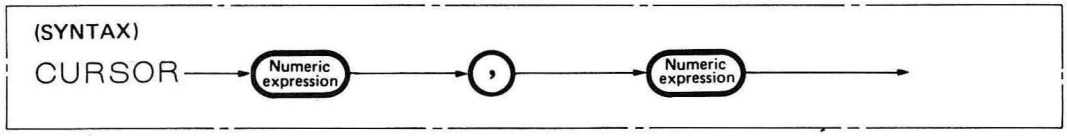
〈Purpose of use〉
To output characters at any position on the CRT.

● **Function 1:** Moves the cursor to the specified coordinate position.

Format: CURSOR X, Y

X: X-coordinate (text coordinate)

Y: Y-coordinate (text coordinate)



Explanation:

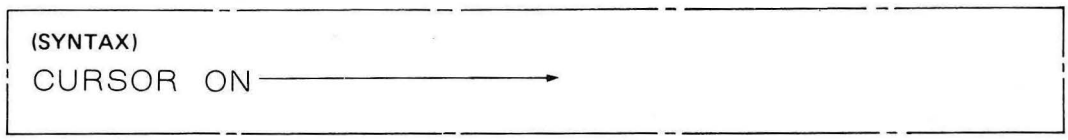
Moves the cursor to the position at text coordinates (X, Y). For the value ranges of numeric expressions X and Y, see the DISP CLEAR statement (page 214).

Application example:

Displays "SHARP COMPUTER MZ-3500 BASIC LANGUAGE MANUAL" in the center of the CRT.

```
10 DISP CLEAR ..... Clears the screen.
20 X=26 ..... Assigns 26 to numeric variable X.
30 CURSOR X,9:DISP "SHARP COMPUTER" .... Displays SHARP COMPUTER starting with the position of text coordinates (26, 9).
40 CURSOR X,11:DISP "MZ-3500" ..... Displays MZ-3500 starting with the position of text coordinates (26, 11).
50 CURSOR X,13:DISP "BASIC LANGUAGE" Displays BASIC LANGUAGE starting with the position of text coordinates (26, 13).
60 CURSOR X,15:DISP "MANUAL" ..... Displays MANUAL starting with the position of text coordinates (26, 15).
70 END ..... Ends program execution.
```

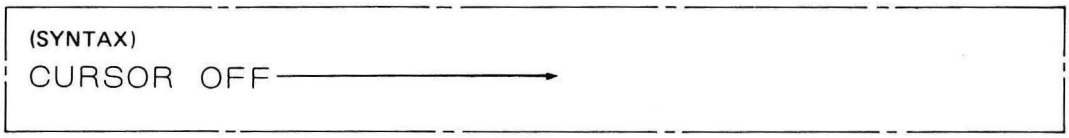
● **Function 2:** Turns ON (displays) the cursor display during program execution.
Format: CURSOR ON



Explanation:

Turns ON (displays) the cursor on the CRT during program execution.

● **Function 3:** Turns OFF (erases) the cursor display during program execution.
Format: CURSOR OFF



Explanation:

Suppresses cursor display during program execution.
CURSOR OFF is automatically set at power ON time.

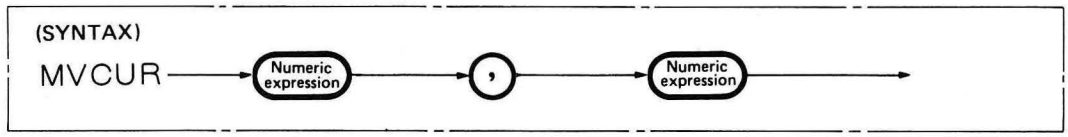
MVCUR

Moves the cursor relative to the current cursor position.

Format: MVCUR X, Y

X: Number of characters cursor is moved in X-direction

Y: Number of characters cursor is moved in Y-direction



Explanation:

Moves the cursor X characters to the right and Y characters down from the current cursor position. The cursor is moved | X | characters to the left when value X is negative, and the cursor is moved | Y | characters up when value Y is negative.

Simple example:

```
10 MVCUR 15,-7
```

Moves the cursor 15 characters to the right and 7 characters up from the current cursor position. (When the current cursor position is (0, 20), the cursor moves to (15, 13).

Application example:

Outputs an arbitrary character block to an arbitrary position on the CRT.

```
  10 DISP CLEAR ..... Clears the screen.
  20 X=30:Y=10:GO SUB 1000 .. Assigns 30 to variable X and 10 to variable Y, and
 999 END                      jumps to line number 1000.
1000 CURSOR X,Y:DISP "♥♥♥♥♥♥♥♥♥♥♥♥♥♥";
                    ..... Outputs ♥♥♥♥♥♥♥♥♥♥♥♥♥♥ from text coordinates (30, 10).
1010 MVCUR -14,1:DISP "SHARP COMPUTER";
                    ..... Outputs SHARP COMPUTER under the character-string.
1020 MVCUR -14,1:DISP "  MODEL-3500";
                    ..... Outputs MODEL-3500 under the character string.
1030 MVCUR -15,1:DISP "♥♥♥♥♥♥♥♥♥♥♥♥♥♥";
                    ..... Outputs ♥♥♥♥♥♥♥♥♥♥♥♥♥♥ under the character string.
1040 RETURN
```

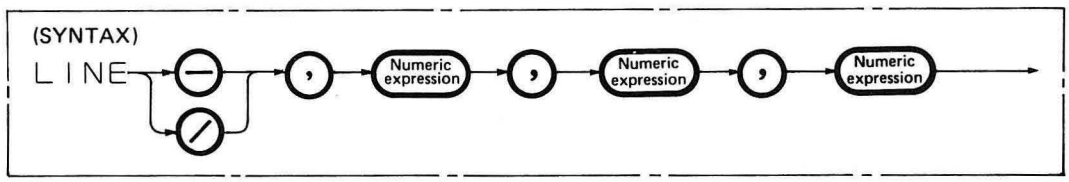
LINE

Draws a ruler line as specified.

〈Purpose of use〉
For effective use of CRT display.

Format: L I NE −/ , X, Y, Z

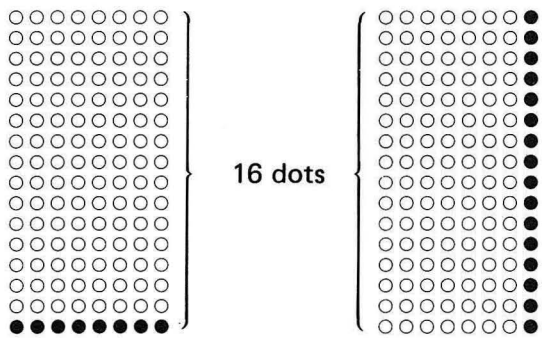
−: Draws line from left to right horizontally
/ : Draws line from up to down. (vertically)
X: X-coordinate (text coordinate) of the starting point
Y: Y-coordinate (text coordinate) of the starting point
Z: Number of characters used in drawing the ruler line.



Explanation:

Draws a ruler line from text coordinates (X, Y) for a length of Z characters in the direction specified by − or / .

For ruler line display in the X-direction, the bottom row of the character display dot matrix is used, and for the ruler line display in the Y-direction, the rightmost column of the character display dot matrix is used.



(X-direction) (Y-direction)

Simple example:

```
10 LINE -,0,10,40
```
..... Draws a ruler line starting with the position of text coordinates (0, 10) from left to right for a length of 40 characters.

Note

1. To erase the ruler line drawn by this statement, use the RE statement (page 229). For details, see function 1 of the RE statement. The ruler line can be erased at the same time as clearing characters by pressing CTRL and HOME keys.
2. If numeric expression Z is specified overrunning the bottom line of the text screen when drawing a vertical ruler line, an error (ERROR 1) occurs.

3. When drawing a horizontal ruler line, the ruler line is continued to the next line if numeric expression Z exceeds the rightmost column of the text screen. If Z exceeds the rightmost column of the bottom line, an error (ERROR 1) occurs.
4. The ruler line here differs from the pseudo ruler line drawn by a combination of pseudo graphic symbols.

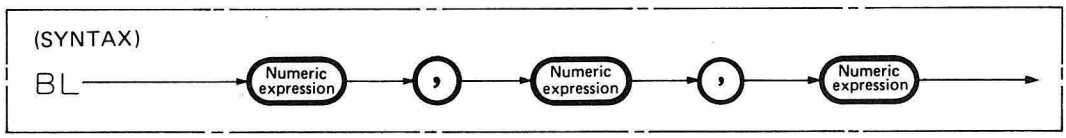
BL

Blinks the characters displayed on the CRT.

〈Purpose of use〉
To highlight characters by making them blink.

Format: BL X, Y, Z

X: X-coordinate (text coordinate) of the first character to be blinked.
Y: Y-coordinate (text coordinate) of the first character to be blinked.
Z: Number of characters to be blinked.



Explanation:

Blinks Z characters starting with the position of text coordinates (X, Y).
Reverse and color characters can also be blinked.

Simple example:

```
10 BL 0,2,80
```
..... Blinks the 80 characters starting with the position of text coordinates (0, 2).

Note

1. To release the blinking state, use the RE statement (page 229). For details, see function 2 of the RE statement. The blinking can be stopped at the same time as clearing characters by pressing CTRL and HOME keys.
2. This statement cannot blink graphic patterns. (Pseudo graphic symbols are a kind of character.)

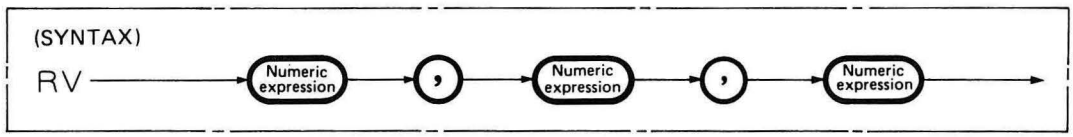
RV

Displays the characters displayed on the CRT in reverse.

〈Purpose of use〉
To highlight characters by displaying them in reverse.

Format: RV X, Y, Z

X: X-coordinate (text coordinate) of the first character to be displayed in reverse.
Y: Y-coordinate (text coordinate) of the first character to be displayed in reverse.
Z: Number of characters to be displayed in reverse.



Explanation:

Displays in reverse the Z characters starting with the position specified by text coordinates (X, Y). Blinking characters can also be displayed in reverse. Color characters cannot be displayed in reverse.

Simple example:

```
10 RV 5,5,20
```
..... Displays in reverse 20 characters from text coordinates (5, 5).

Note

1. To release the reverse-display state, use the RE statement (page 229). For details, see function 3 of the RE statement. Display in reverse can be stopped at the same time as clearing characters by pressing CTRL and HOME keys.
2. This statement cannot display in reverse graphic patterns. (Pseudo graphic symbols are a kind of character.

RE

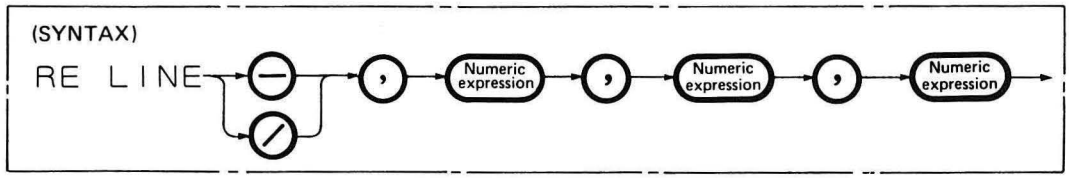
Erases ruler lines or releases blinking or reverse states.

〈Purpose of use〉
To clear ruler lines which cannot be cleared by the DISP CLEAR statement or to release the blinking or reverse states.

● **Function 1:** Erases ruler lines.

Format: RE LINE −|/, X, Y, Z

X: X-coordinate (text coordinate) of the starting point
Y: Y-coordinate (text coordinate) of the starting point
Z: Number of characters in the ruler line to be erased



Explanation:

Erases the ruler line for Z characters starting with the position of text coordinates (X, Y) in the direction specified by − or /. − is from left to right, and / is from top to bottom.

When erasing a vertical ruler line, an error occurs if the value of numeric expression Z overruns the bottom line of the text screen.

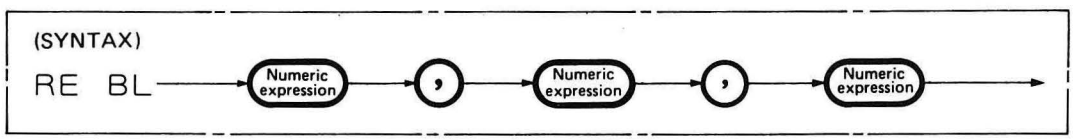
Simple example:

```
100 RE LINE /,0,0,25 ·····Erases the ruler line 25 characters from text
                          coordinates (0, 0) downward.
```

● **Function 2:** Releases the blinking state.

Format: RE BL X, Y, Z

X: X-coordinate (text coordinate) of the starting point
Y: Y-coordinate (text coordinate) of the starting point
Z: Number of characters for which the blinking state is released



Explanation:

Releases the blinking state of Z characters starting with the position of text coordinates (X, Y).

Simple example:

```
100 RE BL 0,2,80
```
..... Releases the blinking state of 80 characters starting with the position of text coordinates (0, 2).

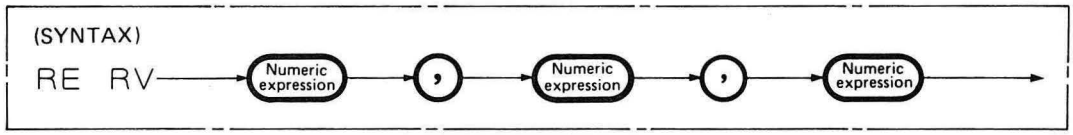
● **Function 3:** Releases the reverse state.

Format: RE RV X, Y, Z

X: X-coordinate (text coordinate) of the starting point

Y: Y-coordinate (text coordinate) of the starting point

Z: Number of characters for which the reverse state is released



Explanation:

Releases the reverse state of Z characters starting with the position of text coordinates (X, Y).

Simple example:

```
100 RE RV 5,5,20
```
..... Releases the reverse state of 20 characters starting with the position of text coordinates (5, 5).

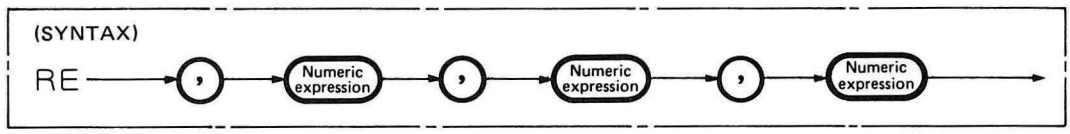
● **Function 4:** Performs functions 1 to 3 at the same time.

Format: RE, X, Y, Z

X: X-coordinate (text coordinate) of the starting point

Y: Y-coordinate (text coordinate) of the starting point

Z: Number of characters to be erased, or for which blinking display and/or reverse state is to be released



Explanation:

Clears a ruler line, blinking and/or reverse state starting with the position of text coordinates (X, Y).

For details, see function 1 to 3.

TABLE

Forms grids for making tables.

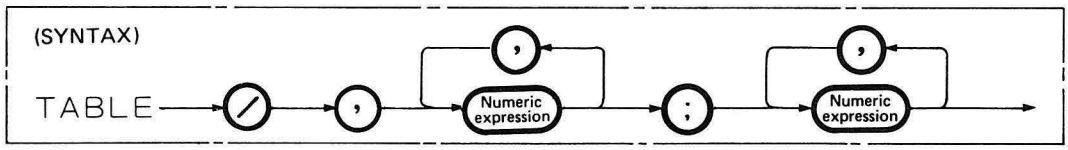
〈Purpose of use〉
To generate frames for table creation.

● **Function 1:** Forms grids using pseudo ruler lines.

Format: TABLE ／, {X} ; {Y}

X: X-coordinate of the grid point (X ≧ 1)

Y: Y-coordinate of the grid point (Y ≧ 1)



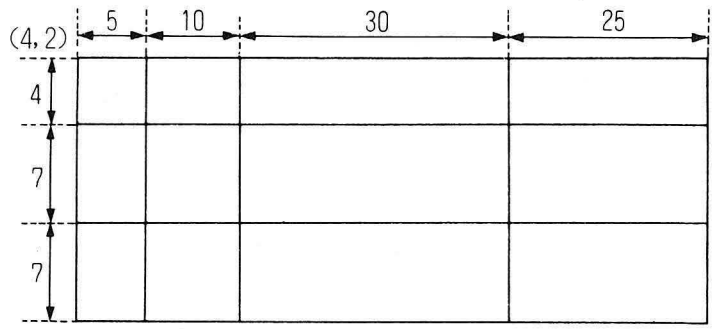
Explanation:

Forms a grid by combining pseudo graphic symbols ├ ┴ ┤ ┬ ┐ ┌ └ ┘ │ ─ ┼ . The top left grid point is at text coordinates (X−1, Y−1) given by the first numeric expression X and Y minus 1 each.

The intervals (in characters) of each grid in X and Y directions are the values of the second and the subsequent expressions X and Y less 1, respectively. This is because, unlike ruler lines, pseudo ruler lines are a kind of character.

Simple example:

```
10 TABLE ／,5,5,10,30,25;3,4,7,7
```
..... Forms a grid as shown below.



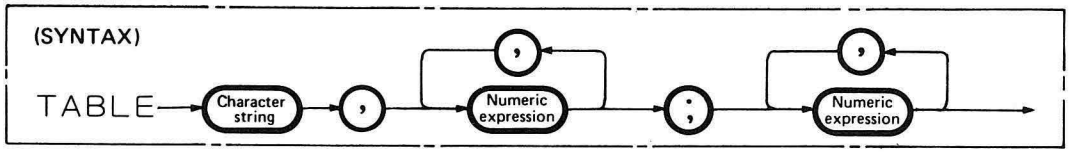
● **Function 2:** Forms a grid with any characters.

Format: TABLE C, {X} ; {Y}

C: Character string which forms the grid

X: X-coordinate of the grid point (X ≧ 1)

Y: Y-coordinate of the grid point (Y ≧ 1)

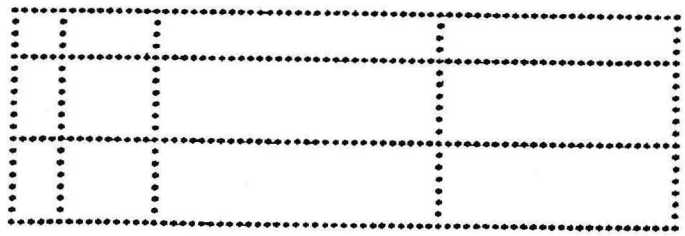


Explanation:

Forms a grid by using the first character of character string C. The grid point positions and intervals (in characters) of each grid are the same as in function 1.

Simple example:

```
10 TABLE "*",5,5,10,30,25:3,4,7,7
```
..... Forms a grid as shown below.

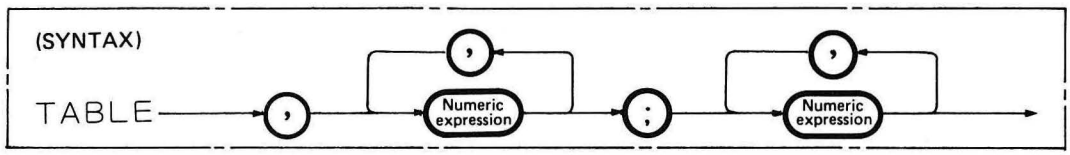


● **Function 3:** Forms a grid with ruler lines.

Format: TABLE, (X) ; (Y)

X: X-coordinate of the grid point (X ≧ 1)

Y: Y-coordinate of the grid point (Y ≧ 1)

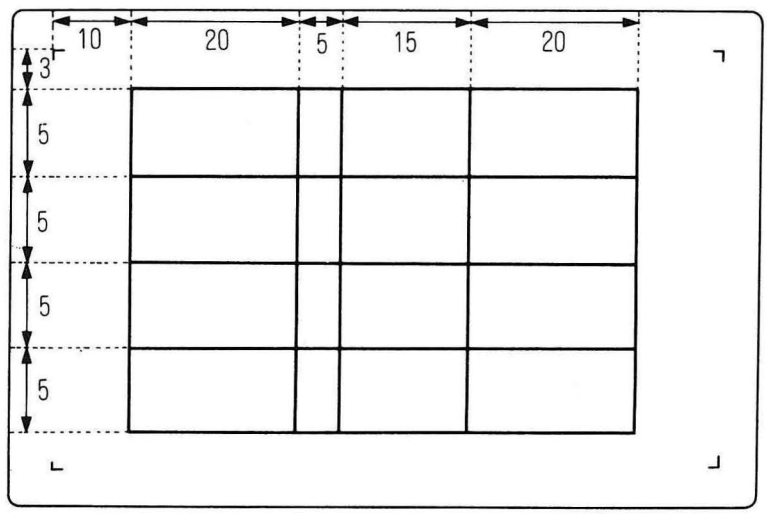


Explanation:

Forms a grid with ruler lines. The positions of grid points are based on function 1. However, the intervals (in characters) of each grid are the second and subsequent values of numeric expression X and Y. The bottom row of the character display dot matrix is used for horizontal ruler line display, and the rightmost column is used for vertical ruler line display. (See the LINE statement (page 225).)

Simple example:

```
10 TABLE ,10,20,5,15,20:3,5,5,5,5
```
..... Forms a grid as shown below.



SCROL

Scrolls (moves) the specified rectangular block in the specified direction.

〈Purpose of use〉
For efficient use of the CRT display, and to give regibility.

Format: SCROL [@] Z, U, V, X, Y

@: Specification of ruler line and character display state

Z: Scroll direction (0, 1, 2, or 3)

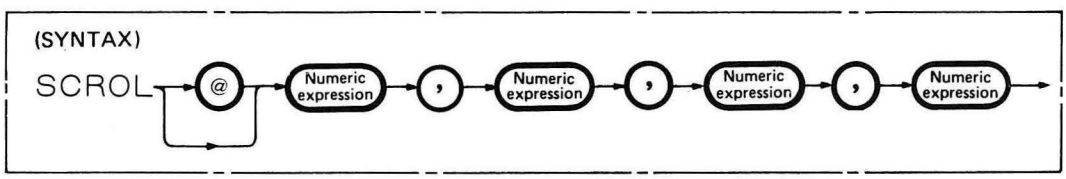
Value of expression Z	Scroll direction
0	↑
1	↓
2	←
3	→

U: X-coordinate of the upper left corner of the block to be scrolled

V: Y-coordinate of the upper left corner of the block to be scrolled

X: X-coordinate of the lower right corner of the block to be scrolled (X ≧ U)

Y: Y-coordinate of the lower right corner of the block to be scrolled (Y ≧ V)



Explanation:

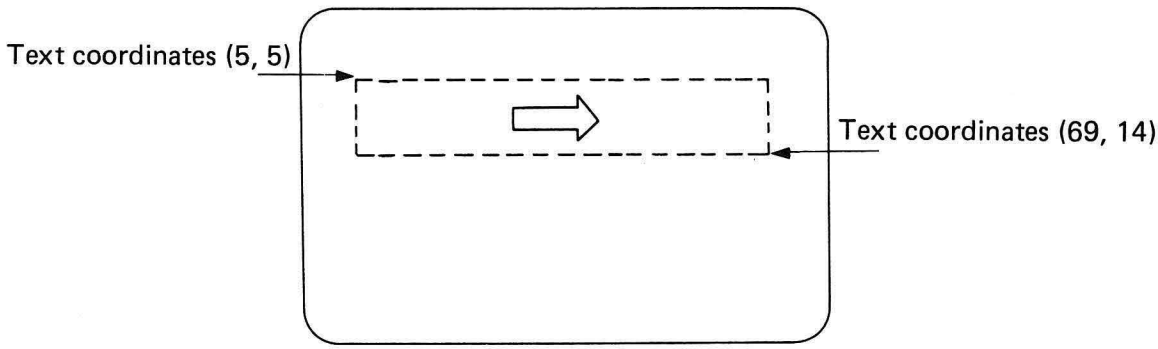
1. Scrolls the rectangular block defined by its upper left corner at text coordinates (U, V) and lower right corner at text coordinates (X, Y) one row or one column in the direction specified by numeric expression Z.
2. When @ is specified, ruler lines and character display states (blinking, reverse, color) are also scrolled. When @ is omitted, neither ruler lines nor character display states (blinking, reverse, color) are scrolled, only characters are scrolled.

Simple example:

1. `100 SCROL @3,5,5,60,14` ..... Scrolls a rectangular block defined by its diagonal points at text coordinates (5, 5) and (60, 14) one column to the right.

2. 100 SCROL 0,5,5,69,14 ..... Scrolls only the characters in the rectangular block defined by its diagonal points at text coordinates (5, 5) and (69, 14) one row upward. (Ruler lines and character display states (blinking, reverse, color) are not scrolled.)

Area and direction of scrolling as in example 2:



CHANGE DISP

Selects the character display mode or monochrome/color mode.

〈Purpose of use〉

To switch the character display mode or monochrome/color mode.

Format: CHANGE DISP S

S: Character constant of the screen mode to be switched to

Character constant	Screen mode	Character constant	Screen mode
"80"	80 x 25 monochrome	"80C"	80 x 25 color
"80/"	80 x 20 monochrome	"80/C"	80 x 20 color
"40"	40 x 25 monochrome	"40C"	40 x 25 color
"40/"	40 x 20 monochrome	"40/C"	40 x 20 color

(SYNTAX)



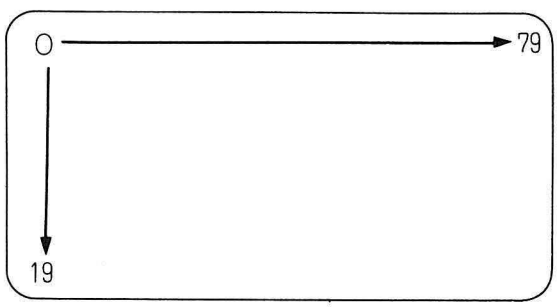
Explanation:

1. Selects the character display mode or monochrome/color mode according to the specified character constant S.
2. When power is turned ON, the 80 x 25 monochrome mode is set automatically.

Simple example:

```
10 CHANGE DISP "80/C"
```

..... Sets the screen mode to the 80 x 20 characters in color.



Note

1. For the graphic mode, see the SDISP statement (page 239) and the ODISP statement (page 241).
2. When color display is specified by the DISP statement (page 107), color display is obtained without specifying color display using this statement. A color display device must be used to obtain color display.

CHANGE PRINT

Changes the printer line feed pitch.

〈Purpose of use〉
① To print out tables created using pseudo graphic symbols.
② To return the line feed pitch from graphic mode to normal mode.

Format: CHANGE PRINT S

S : Line feed pitch specification
"N": Normal mode
"G": Graphic mode

(SYNTAX)



Explanation:

1. Sets the printer line feed pitch according to the value of the character constant S. The line feed pitch is 1/6" in the normal mode but in the graphic mode the line feed pitch depends on the printer used.
2. When pseudo graphic symbols are used, set to the graphic mode to have the symbols connected to each other, thus giving a neat appearance.
3. When power is turned ON, the line pitch is set to the normal mode.

Simple example:

```
10 CHANGE PRINT "G" .....Sets the line pitch to the graphic mode.
```

COLOR

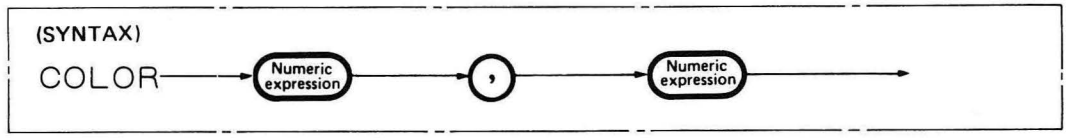
Specifies the color of the characters (including pseudo graphic symbols).

Format: COLOR X, Y

X: Color code (0 to 7)

Color code	0	1	2	3	4	5	6	7
Color	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black

Y: Number of characters



Explanation:

1. Displays Y characters from the current cursor position in the color specified by the color code X.
2. When color is not specified by this statement in the color display mode, the color is automatically set to white.
   (Black (color code = 7) is the same color as the background.)

Simple example:

```
10 COLOR 5,10 ..... Displays 10 characters from the current cursor position in red.
```

Note

1. Before using this statement, the screen must be set to the color display mode by the CHANGE DISP statement (page 235) or the ODISP statement (page 241).
2. This statement is valid when a color display device is connected.
3. Color characters can be blinked but cannot be reversed.

Application example:

Displays COLOR in red in the center of the color display.

```
10 CHANGE DISP "80C" ..... Sets to 80 x 25 color mode.
20 CURSOR 38,12:COLOR 5,5 ..... Sets cursor position and color.
30 DISP " COLOR" ..... Displays COLOR
40 END
```

6. GRAPHIC PROCESSING STATEMENTS

This section describes the statements for handling graphics using the graphic screen.

SDISP
ODISP
GCOLOR
CLS
SCALE
PSET
PRESET
LINE
CIRCLE
PAINT
GENTER
GDISP
GPRINT
GCURSOR
GINPUT
GCHR

All these statements can be used in manual operation. To handle graphics using the graphic processing statements, the graphic board and graphic memory (options) are needed. (Refer to the Owner's Manual.)

SDISP

Specifies the output screen*) on the CRT.

〈Purpose of use〉
① To display the graphic screen on the CRT.
② To display the text screen and graphic screen in combination on the CRT.

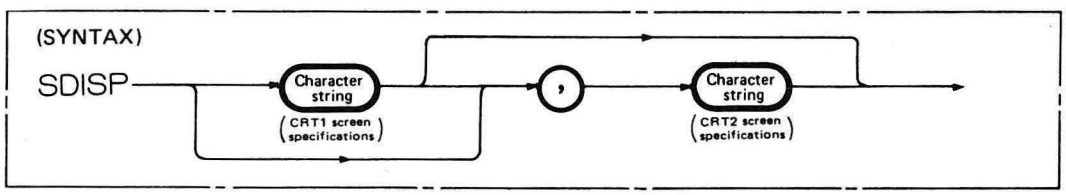
Format: SDISP C [, D] | [C], D

C : CRT-1 output screen specifications
D : CRT-2 output screen specifications
Specify the screen specifications as follows:
"Screen-No. Color Modification"
① ② ③

① Screen No.
0 ····· Text screen
1 ····· Graphic screen 1 (blue)
2 ····· Graphic screen 2 (red)
3 ····· Graphic screen 3 (green)
xy ····· Overlays of two screens
xyz ····· Overlays of three screens
xyzw ····· Overlays of four screens
(x, y, z, w: 0, 1, 2, or 3)

②Color
C ····· Color mode specification

③Modification
A ····· Specification of character modification (blinking, reverse, or ruler line) on CRT-2. However, A can be specified only in character string D.
(CRT-1 can always be modified.)



Explanation:

1. The specified text screen and graphic screens (3) can be displayed on two CRT displays: CRT-1 and CRT-2.
Specify the CRT-1 output screen by the screen specifications C, and the CRT-2 output screen by the screen specifications D.
Screen specifications include screen number, color, and modification specifications.

*) Output screen means the text and/or graphic screens displayed on the CRT. Separates the plot screen output to the video RAM area.

2. Select the output screen (text screen, three graphic screens (blue, red, green), or their combinations) in the screen number specification. When two or more screen numbers are specified, all the specified screens are displayed overlaid. For example, when all three graphic screens are specified, a combination of screens 1, 2, and 3 are displayed.
3. When color C is specified in screen specification C or D, both CRT-1 and CRT-2 can display in color (color mode).
   When color display is specified by the CHANGE DISP statement (page 235), color display can be made without specifying color display in this statement.
4. Specification of character modification and cursor display is valid only for CRT-2 screen specification D. (The cursor is displayed and character modification can always be made on CRT-1.)
5. When screen specifications C or D are omitted, the previous state is retained.
6. At initialization, the text screens, CRT-1 and CRT-2, are set so they can be modified and have the cursor displayed in the monochrome mode (not color).

Simple example:

1. `10 SDISP"1","0A"` .... Displays graphic sreen 1 (blue) on CRT-1. Displays the text screen on CRT-2 in such a way that characters can be modified and the cursor is displayed.
2. `10 SDISP"01C","123"` ..... Overlays the text screen and graphic screen 1 (blue) on CRT-1 and displays in the color mode, and overlays graphic screens 1, 2, and 3 (blue, red, and green) on CRT-2 and displays in the color mode.
3. `10 SDISP,"0"` ..... Retains CRT-1 in the current state, and displays the text screen on CRT-2.

Note

1. To display 8-color graphics, overlay of screen Nos. 1 to 3 must be specified. For color display on the text screen, no overlay is needed.
2. Monochrome mode and color mode cannot be specified for two CRT's separately.
3. When using a color display and a green display connected in the color mode, the green display must be connected to CRT-2.
   In this case, character modification (A) cannot be specified in this statement.

ODISP

Specifies the graphic plot screen*)

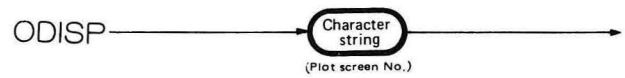
〈Purpose of use〉
To plot graphics.

Format: ODISP C

C : Plot screen No.
- 1 ..... Graphic screen 1 (blue)
- 2 ..... Graphic screen 2 (red)
- 3 ..... Graphic screen 3 (green)
- xy ..... Two screens
- xyz ..... Three screens

(x, y, z: 1, 2, or 3)

(SYNTAX)



Explanation:

1. Graphics are drawn in the video RAM area(s) which correspond(s) to the graphic screen (screen 1, 2, and/or 3) specified as the plot screen. (This process is called plotting.)
   The video RAM areas store the information on the text screen and graphic screens separately. This statement works on the graphics video RAM areas. The plot screen is selected by character string C from the three graphic screens. The text screen is always handled as a plot screen.
2. The graphic statement (PSET, LINE, CIRCLE, PAINT, GDISP, or GCHR) execution result is plotted on the graphic screen specified by this statement.
3. To display the plotting on the CRT, specify the plot screen number by the SDISP statement (page 239). When the plot screen specified by this statement has been specified by the SDISP statement, the plotting is displayed on the CRT as it is done.
4. At initialization, the plot screen is set to screen 1 (blue).

Simple example:

```
10 ODISP "23"
```
..... Specifies graphic screens 2 (red) and 3 (green) as the plot screens.

Note

To plot graphics in eight colors, specify three screens with plot screen Nos. 1 to 3. When screen Nos. 1 to 3 are specified by the SDISP statement (page 239), 8-color graphics are displayed on the specified CRT display.

*) Plot screen means the screen plotted in the video RAM area (memory) which corresponds to each of the three graphic screens in. Separate the output screen which is displayed on the CRT.

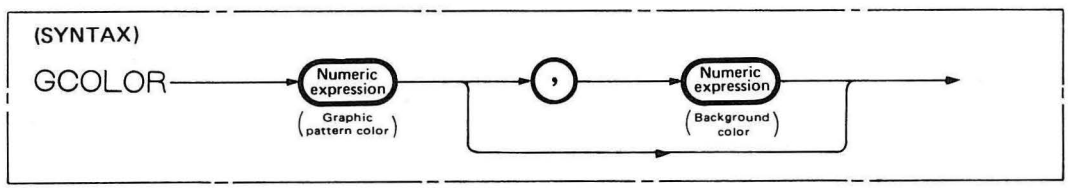
GCOLOR

Specifies the graphic patterns*) and the background color on a graphic screen.

〈Purpose of use〉
① When the color of the graphic pattern to be displayed was omitted in the graphic statement. (When color need not be changed.)
② When set the graphic pattern and its background as desired.

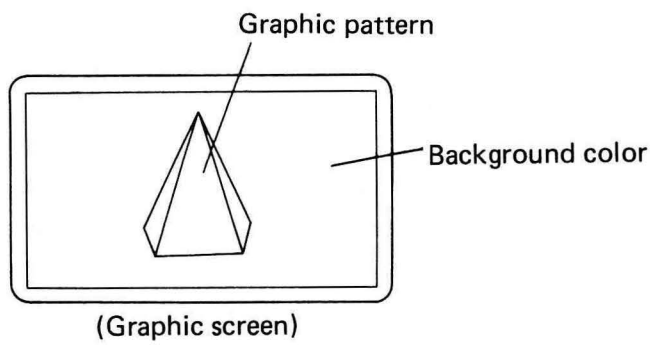
Format: GCOLOR X [, Y]

X : Graphic pattern color (0 to 7)
For color codes, see the COLOR statement (page 237).
Y : Background color (0 to 7)



Explanation:

1. Displays the graphic pattern in the color specified by the value of numeric expression X, background in the color specified by the value of numeric expression Y on a graphic screen.



2. When background color Y is omitted, the background color specified by the previous GCOLOR statement is used.
3. At initialization, the graphic pattern color is white, and the background color black.

*) Graphic pattern: A set of dots displayed on a graphic screen.

Simple example:

1. 10 GCOLOR 5,3 ..... Sets the graphic pattern color to red (color code = 5), and the background color to green (color code = 3).
2. 10 GCOLOR 5,3 ..... Sets the graphic pattern color to red, and the background color to green.

   ⋮

   50 GCOLOR 1 ..... Sets the graphic pattern color to yellow (color code = 1), and the background color unchanged (green).

   ⋮

Note

1. This statement is valid when the color mode is specified by the CHANGE DISP statement (page 235) or SDISP statement (page 239). This statement is invalid in the monochrome mode.
2. When drawing graphics by using a graphic statement, the graphics are drawn in the color specified in the graphic statement if it is specified; otherwise, the color specified in this statement is used.
3. If the background color is specified different by to the current colors, the background color on the CRT is changed immediate upon execution of this statement. However, the graphic pattern color is valid for the graphic statements executed after this statement. The graphic pattern color specification in this statement does not change the already drawn graphic pattern.
4. On a color screen, the color of the last graphic pattern has priority when graphic patterns are overlaid.
5. To obtain an 8-color graphic display, a graphic board and two graphic memories (options) must be added.
   For details, refer to (page 13) of the Owner's Manual.

CLS

Clears the graphic screen specified by ODISP statement.

〈Purpose of use〉
① To clear the contents plotted on a graphic screen.
② To clear a part of a graphic screen.

● **Function 1:** Clears the entire screen.
Format: CLS

(SYNTAX)



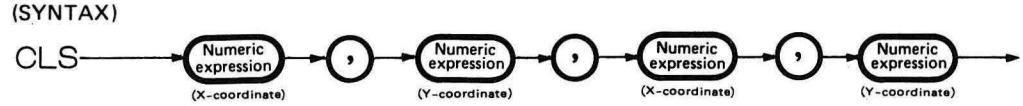
Explanation:

Clears the entire graphic screen (plot screen) specified by the ODISP statement (page 241).

● **Function 2:** Clears a part of the screen.
Format: CLS X, Y, U, V

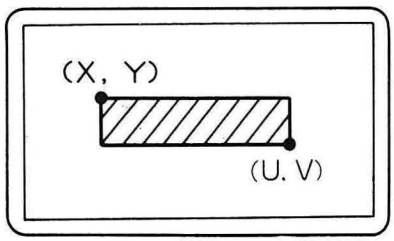
X : X-coordinate of the upper left corner
Y : Y-coordinate of the upper left corner
U : X-coordinate of the lower right corner
V : Y-coordinate of the lower right corner

(SYNTAX)



Explanation:

Clears is the area defined by the upper left coordinates (X, Y) and lower right coordinates (U, V) specified in this statement on the graphic screen (plot screen) specified by the ODISP statement (page 241), i.e., the shaded part in the figure below is cleared.



(Graphic screen)

Simple example:

```
10 CLS 10,10,100,100
```
··Clears the graphic screen area defined by the upper left coordinates (10, 10) and lower right coordinates (100, 100).

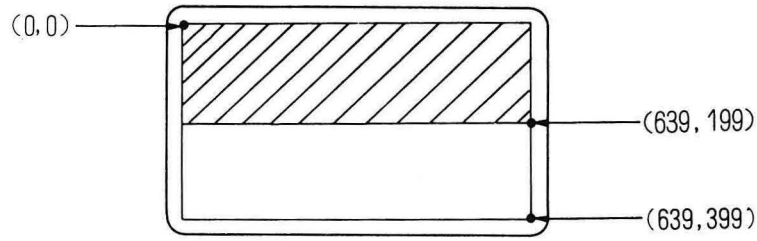
Application example:

Clears the upper half of the three graphic screens.

```
10 OUTPUT "123" ..... Sets graphic screens 1, 2, and 3 as the plot screens.
20 SCALE OFF ..... Returns the graphic coordinate to their initial state.
30 CLS 0,0,639,199 ·· Clears the top half of graphic screens 1 ~ 3.
40 END
```

(Execution result)

The shaded part in the figure below is cleared.



(Graphic screen)

SCALE

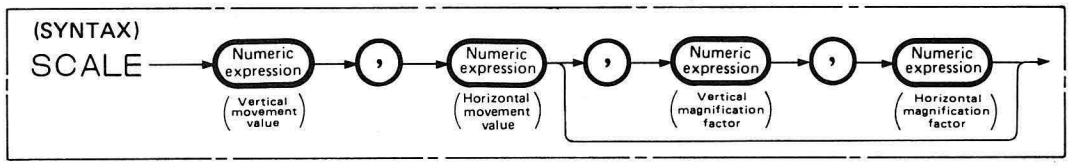
Sets the position of the origin and scales of graphic coordinates, or returns them to the initial state.

〈Purpose of use〉
① To change the position of the origin of the graphic coordinates.
② To change the scale of the X- or Y-coordinate (graphic coordinate).

● **Function 1:** Sets the position of the origin and scale of coordinates.

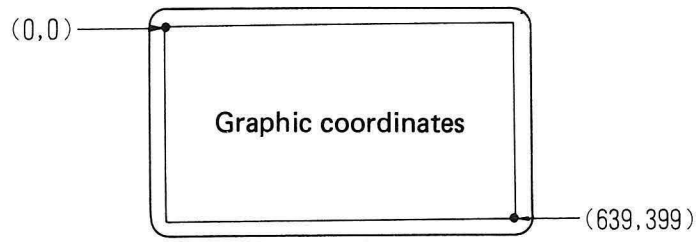
Format: SCALE X, Y [, U, V]

X : X-movement of origin
Y : Y-movement of origin
U : Scale of X-coordinate
V : Scale of Y-coordinate



Explanation:

1. Moves the graphic coordinate origin for the value of numeric expression X to the right and for the value of numeric expression Y downward. (When the value of numeric expression X is negative, the origin is moved to the left, and when the value of numeric expression Y is negative, the origin is moved upward.) The X-coordinate is scaled at the interval specified by the value of expression U, and the Y-coordinate is scaled at the interval specified by the value of expression V.
2. When numeric expressions U and V are omitted, the scales of coordinates stay the same as before.
   When power is turned ON, the scales of coordinates are set to 1.
3. At initialization, graphic coordinate origin (0, 0) indicates the upper left corner of the screen and coordinates (639, 399) indicate the lower right corner. (see the figure below.)



[Graphic screen]

Simple example:

```
10 SCALE 100,100,10,10
```

..... Moves the current graphic coordinate origin (0, 0) 100 to the right and 100 downward, then scales up X- and Y-coordinates 10 times.

● **Function 2:** To return the position of origin and scales of coordinates to the initial states.

Format: SCALE OFF

(SYNTAX)

SCALE OFF ——————→

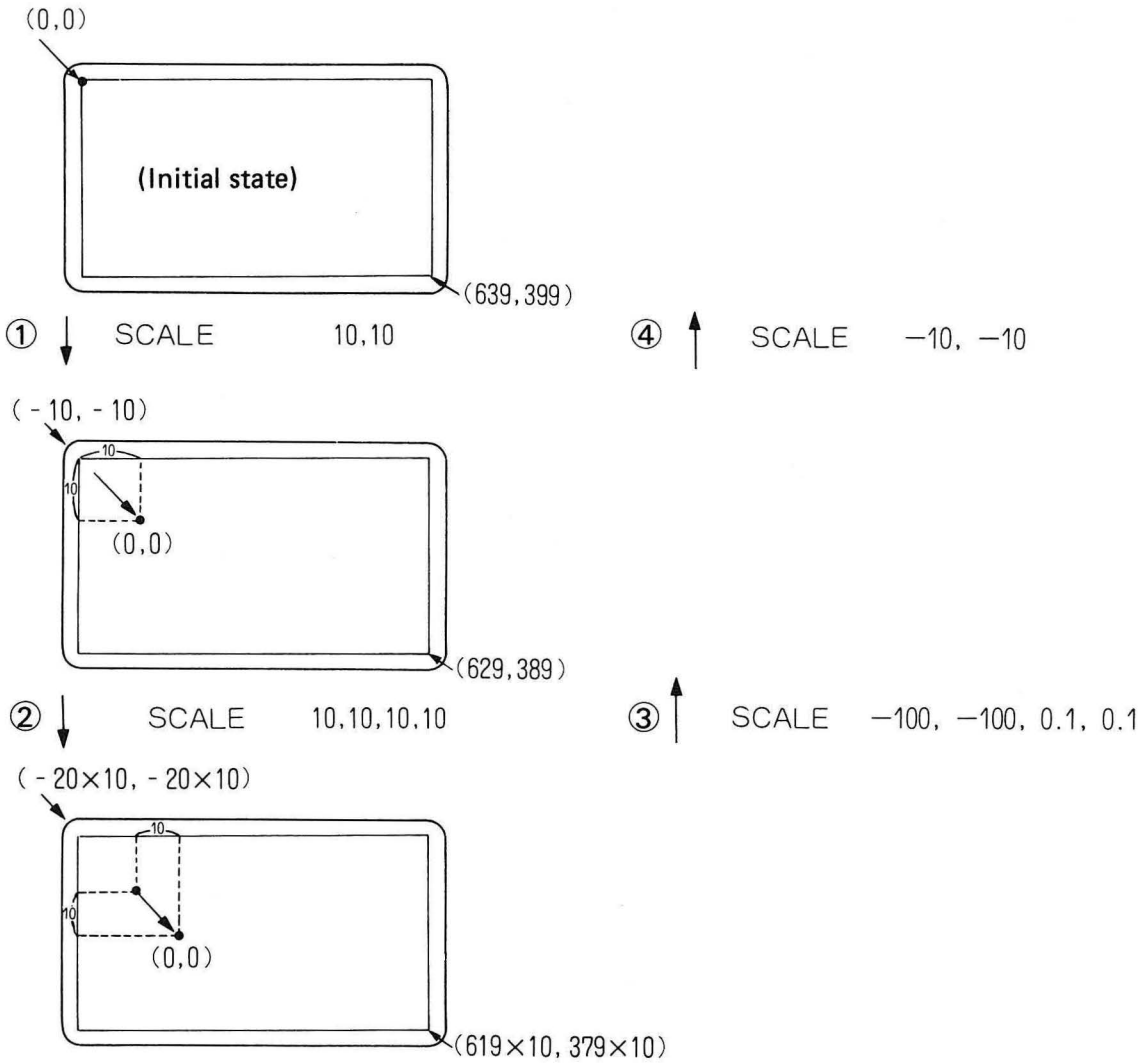
Explanation:

Returns the scale of the graphic coordinates to the initial state (power ON time). The graphic coordinates of the upper left corner are (0, 0) and the graphic coordinates of the bottom right are (639,399).

Application example:

Set the coordinates (upper left corner (-200,-200), lower right corner (6190, 3790)) and returns the coordinates to the initial state later by using this statement.

(The screen returns to its initial state after executing ① to ④ in this order.)



To return to the initial state in one step, SCALE OFF can be used. However, like in this example to sequentially return to the previous state, the coordinates of the origin and the scales of coordinate intervals must be saved.

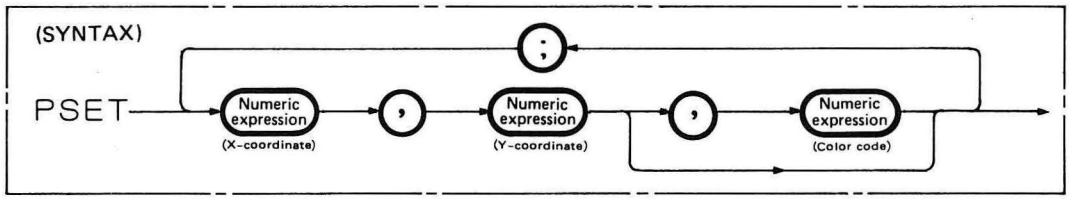
PSET

Lights up dots on a graphic screen.

〈Purpose of use〉
To display graphics by lighting up several points on a graphic screen.

Format: PSET 《X, Y 〔, Z〕》

X : X-coordinate
Y : Y-coordinate
Z : Color code (0 to 7)
For color codes, see the COLOR statement (page 237).
Graphics cannot be displayed in black directly on the background.



Explanation:

1. Lights up the dot at coordinates (X, Y) on a graphic screen.in a color specified by color code Z.
2. When color code Z is omitted, the color specified in the previous GCOLOR statement (page 242) is used. When several dots are specified in this statement, color code Z is omitted for certain dots succeeding a dot specified by color code Z, the color of the preceeding dots become in color Z as specified in the succeeding dot. (See simple example 2)

Simple example:

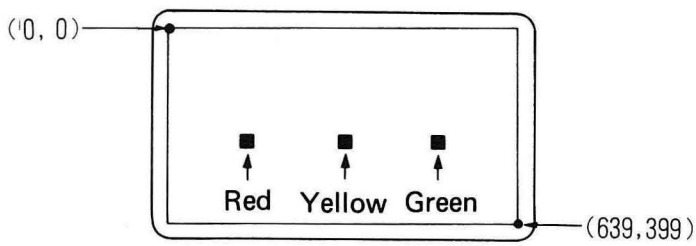
1. `10 PSET 100,100,1` ..... Lights up a dot at graphic coordinates (100, 100) in yellow (color code = 1).
2. `10 PSET 10,10;20,20;30,30,5`
..... Lights up dots (10, 10), (20, 20), and (30, 30) in red (color code = 5).
3. `10 PSET 10,10;10,15,1;10,20`
..... Light up dots (10, 10) and (10, 15) in yellow, and dot (10, 20) in the color specified by the GCOLOR statement located immediate before this statement.

Application example:

Displays three regular squares in red, yellow, and green on the graphic screen.

```
10 SDISP "123C":ODISP "123":CLS ..... Graphic screen initial settings
20 FOR I=1 TO 10
30 FOR N=1 TO 10
40 PSET N+69,I+194,5:N+313,I+194,1:N+559,I+194,3
50 NEXT N
60 NEXT I
70 END
```

(Execution result)



PRESET

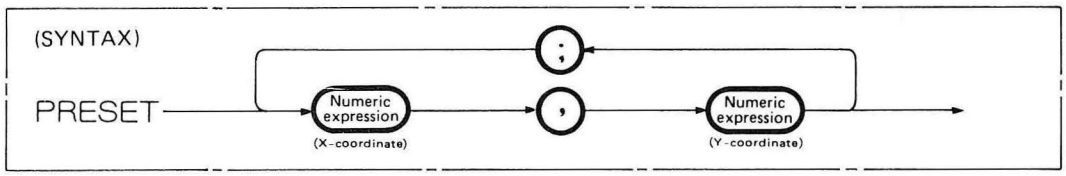
Resets dots on the graphic screen.

〈Purpose of use〉
Resets dots on the graphic screen.

Format: PRESET 《X, Y》

X : X-coordinate

Y : Y-coordinate



Explanation:

Turns off the dot at coordinates (X, Y) on the graphic screen (plot screen). If the dot is not turned on, it is not changed by this statement.

Simple example:

```
10 PRESET 10,10:100,100
```
..... Turns off dots at (10, 10) and (100, 100).

LINE

Plots continuous straight lines on a graphic screen, or erases them.

〈Purpose of use〉

To plot a straight line or a polygon comprising a set of straight lines.

Format: LINE [RESET] [X, Y] ; [[*]U, [*]V] [, C][, Z | [Z], CH]

RESET: Erase mode specification

X : X-coordinate of starting point

Y : Y-coordinate of starting point

* : Relative coordinate specification

U : X-coordinate of point to be connected

V : Y-coordinate of point to be connected

C : Line type

Specify character string C as follows:

A ····· Dotted line ----------

B ····· One-dot line ——·——·——

C ····· Two-dot line ——··——··——

Z : Color code (0 – 7)

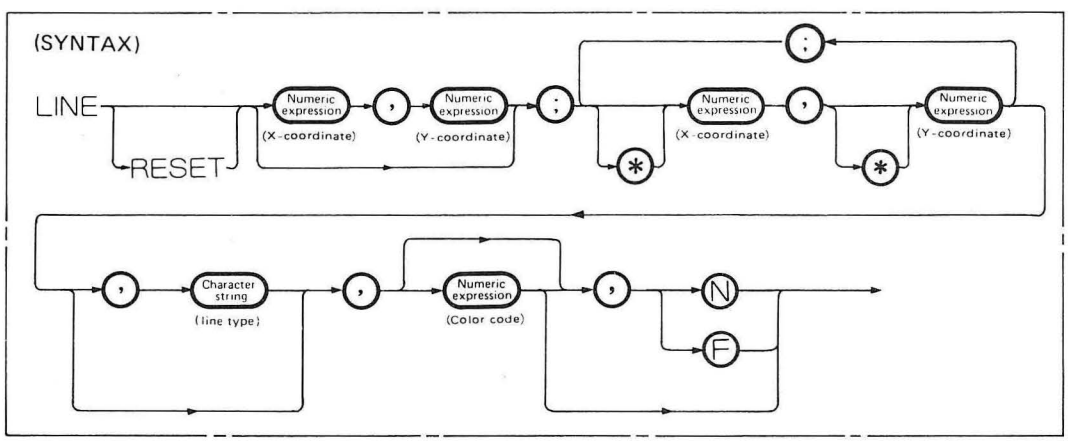
See COLOR statement (page 237) for the color code.

Graphics cannot be displayed in black directly on the background.

CH : Straight line plotting mode

N ····· Plots straight lines by connecting each point (the start point is not connected to the end point). (Normal mode)

F ····· Plots straight lines by connecting each point (the start point is connected to the end point). (Fixed mode)

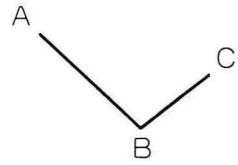


Explanation:

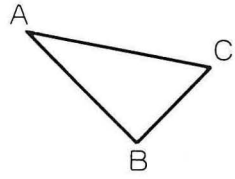
1. On the graphic screen, plots a continuous straight line by sequentially connecting the each point of coordinates (U, V) starting from graphic coordinates (X, Y) in plotting mode CH.
   Character string C specifies the straight line type for plotting a line, and color code Z specifies straight line color. When RESET is specified, the specified straight line is erased.
2. Symbol * gives relative coordinates.
   For example, LINE 100, 100 ; 101, 99 is given as follows with relative coordinates:

```
LINE 100,100;*1,*-1
```

3. Plotting mode CH specifies the mode for plotting straight lines by connecting the specified each point sequentially. For example, points A, B, and C are connected as follows:
   (1) Plotting mode N (Normal mode)



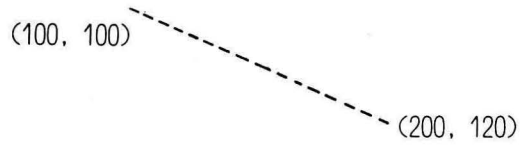
   (2) Plotting mode F (fixed mode)



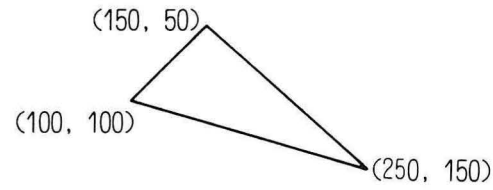
4. When starting point (X, Y) is omitted, the end point of the straight line plotted with the preceding LINE statement is used as the starting point. If no LINE statements have been executed, origin (0, 0) is used as the straight line start point.
5. If the line type specification〔,C〕is omitted, solid lines are plotted.
6. When color code Z is omitted, the color specified in the preceding GCOLOR statement (page 242) is used.
7. When straight line plotting mode CH is omitted, N is assumed to be specified (normal mode).
8. To erase a straight line by specifying RESET (erase mode), specify the line type as a solid line, omit color code Z. The erased part becomes the background color specified in the GCOLOR statement.

Simple example:

1. `10 LINE 100,100:200,120,"A",3,N` ..... Connects starting point (100, 100) to point (200, 120) in plotting mode N (normal mode). The line type is a broken line and the color is green. (color code 3).



2. `10 LINE 100,100:*50,*-50:250,150,,F`
..... Connects starting point (100, 100) and points (150, 50) and (250, 150) in plotting mode F (fixed mode). The line type is a solid line, and the color specified in the GCOLOR statement is used.



3. `10 LINE RESET 100,100:200,120` ..... Erases the straight line (broken line) plotted in Simple example 1. (Specify a solid line for erasing.)

4. `10 LINE RESET 100,100:*50,*-50:250,150,,F`
..... Erases the straight lines (broken line) plotted in Simple example 2.

Note

1. When erasing is specified, the specified lines are erased even if the specified color is not the same as that actually used on the screen. Therefore, omit the color for erasing.
2. When straight line starting point (X, Y) is omitted in this statement, the end point of the straight line plotted with the preceding LINE statement is used as the starting point. In this case, the end point is decided according to the specified plotting mode CH. When the plotting mode is N (normal mode), the last point (U, V) specified in the preceding LINE statement becomes the end point; when the plotting is F (fixed mode), the starting point (X, Y) specified in the preceding LINE statement becomes the end point.

Application example:

Plot a bar graph by inputting the scores (100-point basis) for individual subjects from the keyboard.

Use yellow for English, cyan for mathematics, green for linguistics, magenta for science, and red for current affairs.

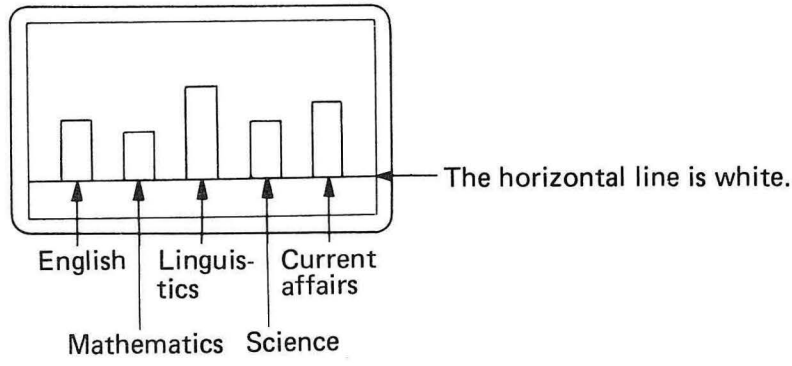
```
10 SDISP "0123C":DDISP"123"
20 DIM A%(5)
30 INPUT "ENGLISH";A%(1)
40 INPUT "MATHEMATICS";A%(2)
50 INPUT "LINGUISTICS";A%(3)
60 INPUT "SCIENCE";A%(4)
70 INPUT "CURRENT AFFAIRS";A%(5)
80 DISP CLEAR
90 CLS
100 LINE 0,300;639,300,0
110 FOR I=1 TO 5
120 A%(I)=A%(I)*2
130 LINE 100*I,300;*0,*-A%(I);*40,*0;*0,300,I,N
140 NEXT
150 END
```

- Displaying the text and graphic screens in colors on the CRT 1.
- Inputs the subjects points from the keyboard.
- Clears the screen.
- Plots a horizontal solid line in white (color code 0).

(Input data)

(English)	60
(Methematics)	35
(Linguistics)	80
(Science)	45
(Current affairs)	70

(Execution results)



CRT 1

CIRCLE

Plots or erases a circle, an ellipse, or a part of these them.

〈Purpose of use〉

For plotting a circle, ellipse, or a sector.

● **Function 1:** Plots or erases a circle or an ellipse.

Format: CIRCLE [RESET] X, Y, R [, W] [; ; Z | [Z], CH]

RESET: Erase mode specification

X : X-coordinate of center

Y : Y-coordinate of center

R : Radius of circle or major axis of ellipse

W : Axis ratio of ellipse

Axis ratio ≧ 1 Major axis in vertical direction

Axis ratio < 1 Major axis in horizontal direction

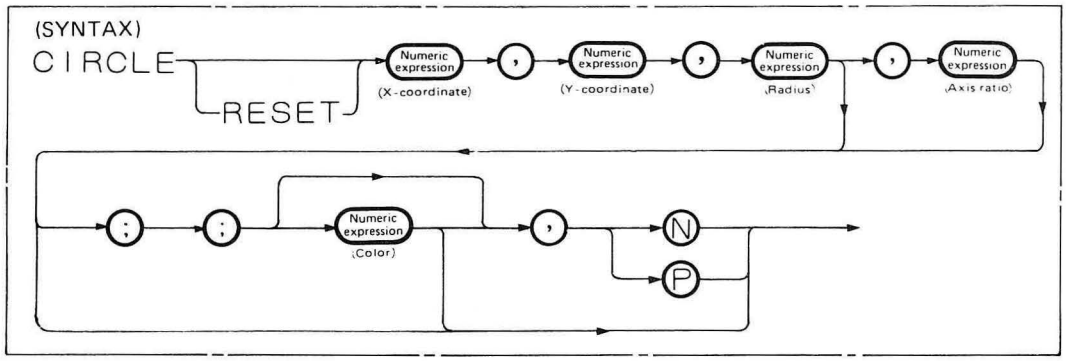
Z : Color code (0 to 7)

See the COLOR statement (page 237) for the color code.

CH : Plotting mode

N . . . . . Plots a circle or an ellipse (normal mode).

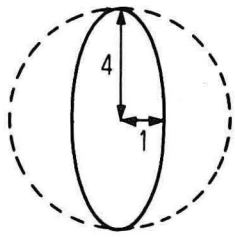
P . . . . . Plots a circle or an ellipse and paints the inside (paint mode).



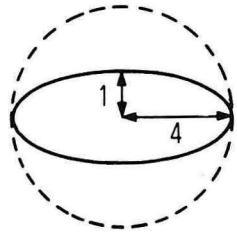
Explanation:

1. When axis ratio W is omitted, a circle of radius R is plotted, centered at graphic coordinates (X, Y), in color Z and plotting mode CH.
2. When axis ratio W is specified, an ellipse is plotted.
   When the axis ratio (W = vertical axis/horizontal axis) is greater than 1, the major axis is in the vertical direction; when the axis ratio is less than 1, the major axis is in the horizontal direction.

See the following examples:



Axis ratio = 4 (≧1)



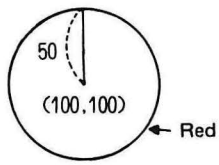
Axis ratio = 0.25 (<1)

If the axis ratio is inverted (1/W), the ellipse is turned through 90°.

3. When color code Z is omitted, the color specified in the preceding GCOLOR statement (see page 242) is used.
4. When plotting mode CH is omitted, N (normal mode) is assumed.
   See simple examples 1 and 2 for the plotting mode.
5. When RESET (erase mode) is specified, the specified circle or ellipse is erased. In such cases, color code Z may be omitted. (The color of the erased part becomes the background color specified in the GCOLOR statement.)

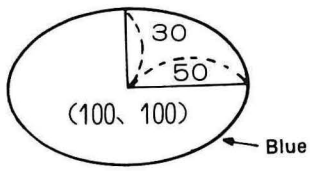
Simple example:

1. `10 CIRCLE 100,100,50;;5,N`



..... Plots a circle of radius 50, centered at (100, 100), in red (color code 5) and normal mode (N).

2. `10 CIRCLE 100,100,50,0.6;;6,P`



..... Plots an ellipse of major axis 50 and axis ratio 0.6, centered at (100, 100), in blue (color code 6) and paint mode (P). (the inside of the ellipse is painted.)

3. `20 CIRCLE RESET 100,100,50`

..... Erases the ellipse plotted in simple example 1.

4. `20 CIRCLE RESET 100,100,50,0.6;;,P`

..... Erases the ellipse plotted in simple example 2.

● **Function 2:** Plots or erases a sector (part of a circle or ellipse).

Format: CIRCLE [RESET] X, Y, R [, W] ; U | [U], V [; Z | [Z][, CH]]

RESET: Erase mode specification

X : X-coordinate of center

Y : Y-coordinate of center

R : Radius of circle or major axis of ellipse

W : Axis ratio of ellipse (vertical axis/horizontal axis)

U : Starting angle (0 to 360)

V : End angle (0 to 360)

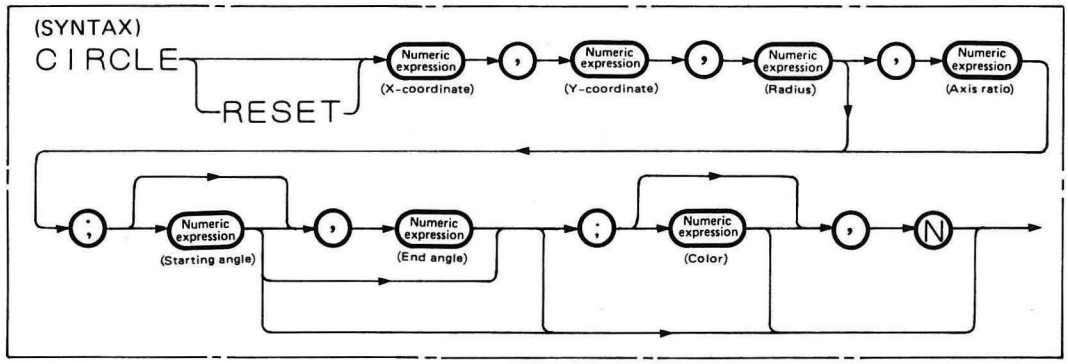
Z : Color code (0 to 7)

See the COLOR statement (page 237) for the color code.

Graphics cannot be plotted in black directly on the background.

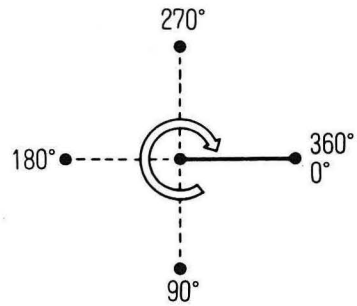
CH : Plotting mode

N ..... Plots a sector by connecting the center and the circle or ellipse arc (normal mode).



Explanation:

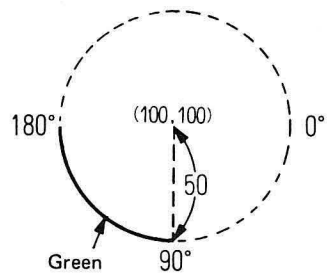
1. When axis ratio W is omitted, an arc of a circle with starting angle U and end angle V with radius R, centered at graphic coordinates (X, Y) is plotted in plotting mode CH and color Z. (Plotting mode N)
2. When axis ratio W is specified, the sector of the ellipse centered at graphic coordinates (X, Y) is plotted with starting angle U and end angle V, with each end connected to the center and in color Z. (plotting mode N)
3. When the plotting mode specification is omitted, an arc or ellipse arc is plotted.
4. Starting angle U and end angle V are set in the direction of the arrow from the horizontal line at center (X, Y) as shown in the figure below. Starting angle U and end angle V are specified with integers in the range 0 to 360, and the U value may be greater than the V value. (See Simple example 2.)



5. When starting angle U is omitted, 0 is assumed; when end angle V is omitted, 360 is assumed.
6. When color code Z is omitted, the color specified in the preceding GCOLOR statement (see page 242) is used.
7. When RESET (erase mode) is specified, the specified circle or ellipse arc is erased. Omit color code Z in this case. (The erased part becomes the background color specified in the GCOLOR statement.)

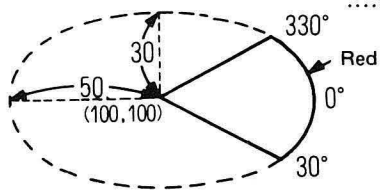
Simple example:

1. `10 CIRCLE 100,100,50;90,180,3`



..... Plots a circle arc of radius 50 from start angle 90° to end angle 180°, centered at (100, 100), in green (color code 3).

2. `10 CIRCLE 100,100,50,0.6;330,30;5,N`



..... Plots a sector of an ellipse of major axis 50 and axis ratio 0.6 from start angle 330° to end angle 30°, centered at (100, 100), in red (color code 5).

3. `20 CIRCLE RESET 100,100,50;90,180`

..... Erases the circle arc plotted in simple example 1.

4. `20 CIRCLE RESET 100,100,50,0.6;330,30;,N`

..... Erases the ellipse arc plotted in Simple example 2.

Note

When specifying RESET (erase mode) in this statement, note the following:

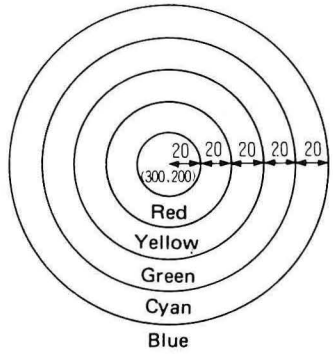
If erasing in plotting mode N (Normal mode) is specified for a circle, or an ellipse (inside painted) plotted in plotting mode P (paint mode), only the outside lines are erased and the painted area remains unerased. (And error does not occur.)

If erasing in plotting mode P (paint mode) is specified for a circle, an ellipse, or a sector (not painted) plotted in plotting mode N (normal mode), the lines and the inside are to be erased. Therefore, if there are graphic patterns plotted inside, they are erased at the same time.

Application example:

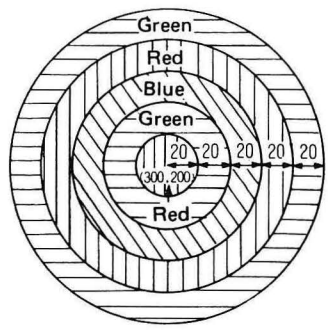
1. Plot the following five concentric circles. The colors are red, yellow, green, cyan, and blue from inside putwards.

```
10 SDISP "123C":ODISP "123":CLS ··· Graphic screen initialization
20 X=300:Y=200:R=20 ····· Center (300, 200)
30 CIRCLE X,Y,R;;5 ····· Circle of radius 20 in red (color code 5)
40 CIRCLE X,Y,R*2;;1 ····· Circle of radius 40 in yellow (color code 1)
50 CIRCLE X,Y,R*3;;3 ····· Circle of radius 60 in green (color code 3)
60 CIRCLE X,Y,R*4;;2 ····· Circle of radius 80 in cyan (color code 2)
70 CIRCLE X,Y,R*5;;6 ····· Circle of radius 100 in blue (color code 6)
80 END
```



2. Draws the above concentric circles and paints them from the outside inwards. Plots a circle at 5 seconds interval and changes the color at the same time. The colors are red, green, blue, red, and green from inside.

```
10 SDISP "123C":ODISP "123":CLS ····Graphic screen initialization
20 X=300:Y=200:R=20 ····· Center (300, 200)
30 CIRCLE X,Y,R*5;;3,P:WAIT 50 ····· Paints a circle of radius 100, in green
                                      (color code 3).
40 CIRCLE X,Y,R*4;;5,P:WAIT 50 ····· Paints a circle of radius 80, in red
                                      (color code 5).
50 CIRCLE X,Y,R*3;;6,P:WAIT 50 ····· Paints a circle of radius 60, in blue
                                      (color code 6).
60 CIRCLE X,Y,R*2;;3,P:WAIT 50 ····· Paints a circle of radius 40, in green
                                      (color code 3).
70 CIRCLE X,Y,R;;5,P:WAIT 50 ····· Paints a circle of radius 20, in red
                                      (color code 5).
80 END
```



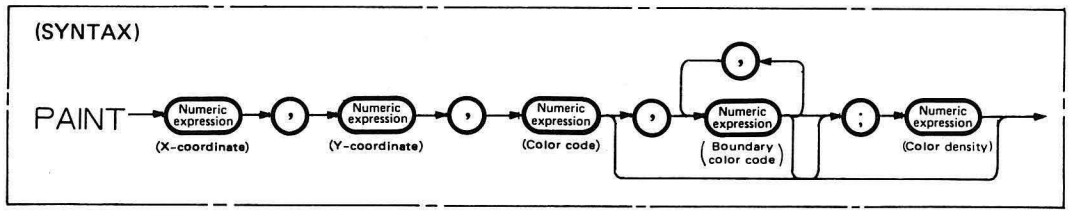
PAINT

Paints the inside of a closed curve.

〈Purpose of use〉
① For painting the inside of a plotted graphics.
② For painting different color densities.

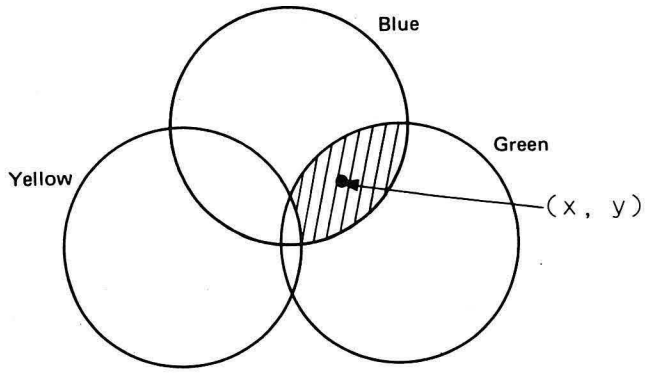
Format: PAINT X, Y, Z [, {U}] [; V]

X : X-coordinate of painting start point
Y : Y-coordinate of painting start point
Z : Painting color code (0 – 7)
See the COLOR statement (page 237) for the color code.
Graphics cannot be plotted in black directly on the background.
U : Boundary color code (0 – 7)
V : Color density
0 . . . Painting 100%
1 . . . Painting 75%
2 . . . Painting 50%
3 . . . Painting 25%
V-specification does not make any difference for black painting.



Explanation:

1. Starts painting from the specified graphic coordinates (X, Y) and paints the inside of the boundary line of color U in color Z and color density V. The boundary line is not painted.
2. An arbitrary number of colors can be specified for boundary color U. The smallest closed curve enclosed by the boundary line in one of these colors is painted. (see the figure below.)

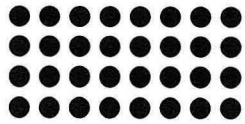


If the paint start point is (x, y) and boundary colors U are yellow, green, and blue, the shaded part in the figure on the left is painted.

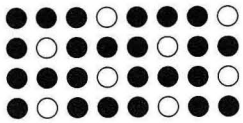
3. The color densitites indicated with the value of numeric expression V are classified as follows according to the painted dot ratio:

○ indicates a dot.
● indicates a painted dot.

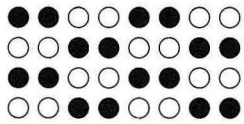
1 100% painting (when V = 0)



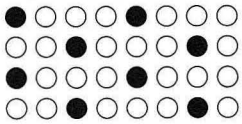
2 75% painting (when V = 1)



3 50% painting (when V = 2)



4 25% painting (when V = 3)

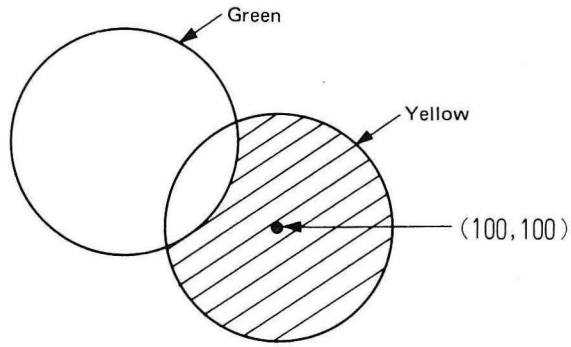


4. When boundary color U is omitted, all color codes 0 to 7 are assumed.
5. When color density V is omitted, 100% painting (V = 0) is assumed.

Simple example:

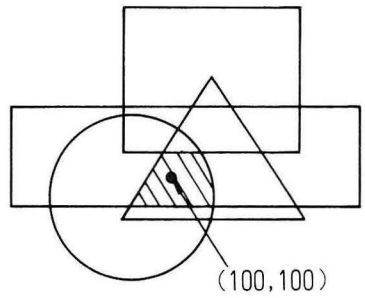
1. `10 PAINT 100,100,5,1,3;0` ..... Starts painting from point (100, 100) and paints the inside of the boundary lines of yellow (color code 1) or green (color code 3) in red (color code 5) in 100% density.

For example, if the green circle, yellow circle, and painting start point (100, 100) are positioned as follows, the shaded part is painted:



2. 10 PAINT 100,100,1,2 ..... Starts painting from point (100, 100) and paints the inside of the boundary lines in yellow (color code 1) at 50% density.

In this example, the boundary line color is omitted. Therefore, only the inside of the smallest closed curve (shaded-part) around the painting start point (100, 100) is painted.

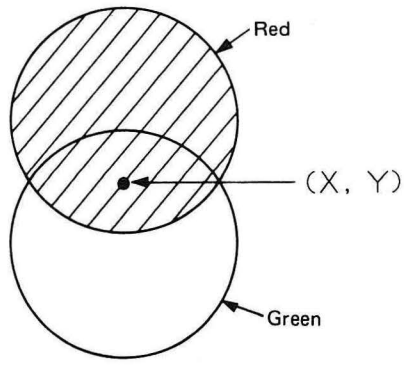


Note

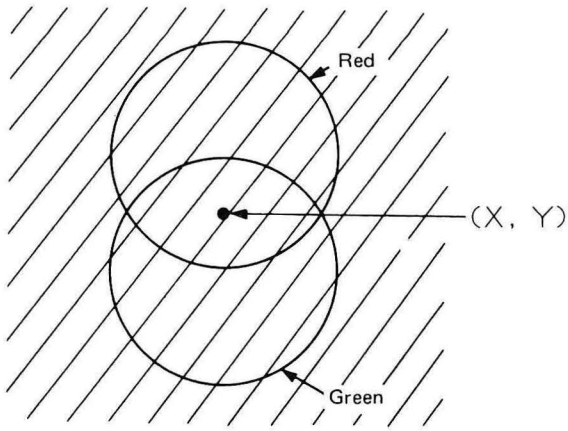
When several closed curves are overlapped, the painted part (hatched area in the figure) varies according to the overlapping sequence and the cross-point color (whether this is black or not).

Let the painting start point be (X, Y), and red be specified as the boundary color.

1. When a red closed curve is overlapped on a green closed curve, the shaded part is painted as follows:



2. When overlapping a green closed curve on a red closed curve, the display is as follows:



Generally, if there is a different color point on a closed curve of the boundary color, the closed curve is cut off at this point and is not assumed to be a boundary line.

Application example:

As shown in the following figure, an equilateral triangle is plotted in white, circles in green, centered at these vertices, then paints the hatched area.

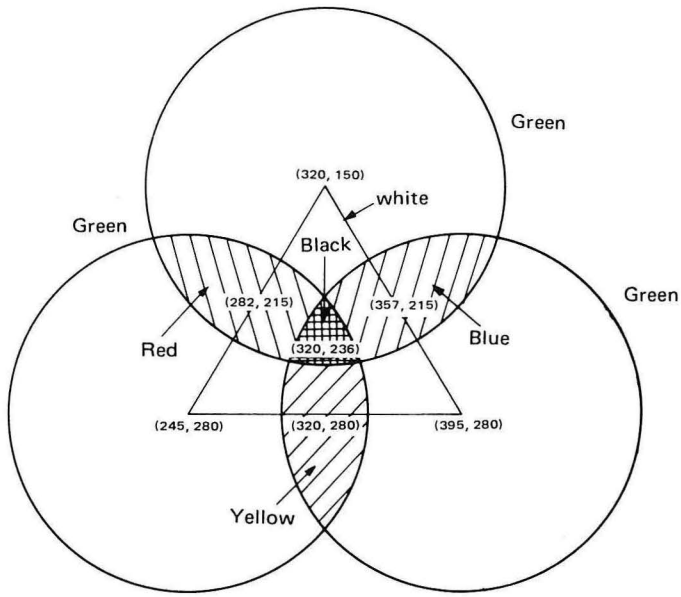
Uses a red color of 50% density for the part centered at the point (282, 215).

Uses a yellow color of 50% density for the part centered at the point (320, 280).

Uses a blue color of 50% density for the part centered at the point (357, 215).

Uses a black color of 100% density for the part centered at the point (320, 236).

The coordinates of the triangle vertices are not the accurate values (they are approximate values).



```
10 SDISP "123C":ODISP "123":CLS
```
Initializes the graphic screen.

```
20 LINE 320,150:245,280:395,280,0,F
```
Plots an equilateral triangle in white (color code 0) by connecting the three vertices.

```
30 CIRCLE 320,150,100:;3
```
Plots a circle of radius 100 in green (color code 3), centered at point (320, 150).

```
40 CIRCLE 245,280,100:;3
```
Plots a circle of radius 100 in green, centered at (245, 280).

```
50 CIRCLE 395,280,100:;3
```
Plots a circle of radius 100 in green, centered at point (395, 280).

```
60 PAINT 320,236,0,3;0
```
Starts painting from point (320, 236) and paints the inside of the green boundary line in white (color code 0) at color density 100%.

```
70 PAINT 282,215,5,3;2
```
Starts painting from point (282, 215) and paints the inside of the green boundary lines in red (color code 5) at color density 50%.

```
80 PAINT 320,280,1,3;2
```
Starts painting from point (320, 280) and paints the inside of the green boundary line in yellow (color code 1) at color density 50%.

```
90 PAINT 357,215,6,3;2
```
Starts painting from point (357, 215) and paints the inside of the green boundary line in blue (color code 6) at color density 50%.

```
100 END
```

GENTER

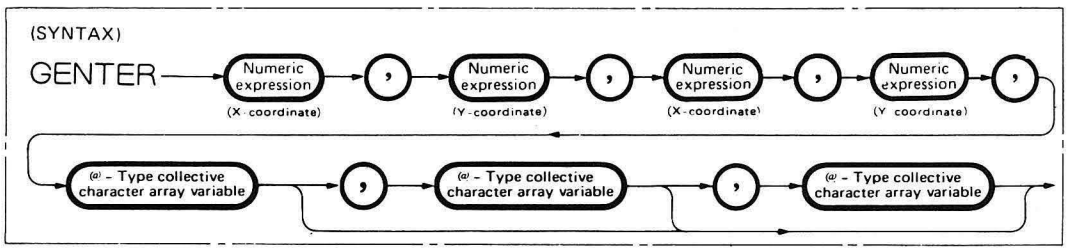
Inputs the graphic pattern displayed in the specified area of the graphic screen.

〈Purpose of use〉

To read the graphic pattern displayed on the graphic screen into the specified variables and output it to the CRT or printer.

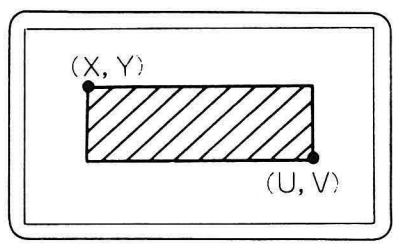
Format: GENTER X, Y, U, V, A@(*) [, B@(*) [, C@(*)]]

X : X-coordinate of upper-left corner
Y : Y-coordinate of upper-left corner
U : X-coordinate of lower-right corner
V : Y-coordinate of lower-right corner
A @ (*) : @-type generalized character array variable
B @ (*) : @-type generalized character array variable
C @ (*) : @-type generalized character array variable



Explanation:

1. Reads the graphic pattern drawn in the area (shaded part in the figure) specified by the upper-left coordinates (X, Y) and lower-left coordinates (U, V) on the graphic screen (displayed on the CRT) into the generalized character array variable.



[Graphic screen]

2. When three screens are displayed overlaid, specify three @-type generalized character array variables; A@(*), B@(*), and C@(*).
3. Decide the size of @-type generalized character array variable A@(*) as follows:

   Monochrome mode: A@(V − Y + 1) *m (1-dimensional)
   Color mode: A@(3, V − Y + 1) *m (2-dimensional)

   Specify the above value in the DIM statement (see page 128).
   Where, m = (U − X + 1)/8. (The fraction is rounded up.)

4. The graphic pattern read into the variable can be displayed on the CRT with a GDISP statement, or output to the printer with a GPRINT statement.

Simple example:

1. `20 GENTER 10,20,100,120,A@(*)`
   ..... Reads the graphic pattern in the area specified with the upper-left coordinates (10, 20) and lower-right coordinates (100, 120) on the graphic screen into A@(*).

Before executing this statement, declare the size of A@(*) in the DIM statement (see page 128) as follows:

1 Monochrome mode

`DIM A@(101)*12` ..... Declares the character array variable (1-dimensional) of @-type.

{ Number of elements = 120 − 20 + 1 = 101
Element size = (100 − 10 + 1)/8 = 11.375
.... ≒ 12 (The fraction is rounded up.)

2 Color mode

`DIM A@(3,101)*12` ..... Declares the character array variable (2-dimensional) of @ type.

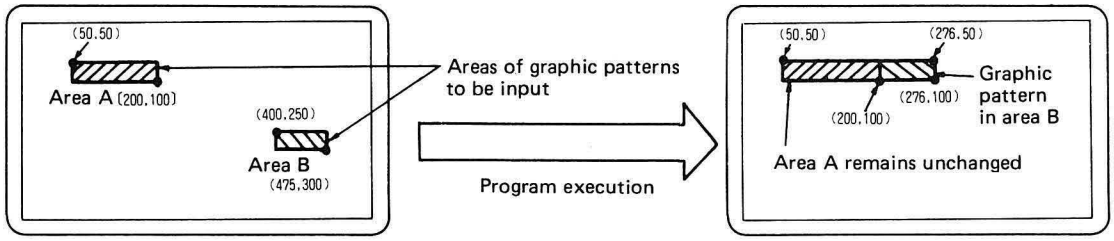
2. `20 GENTER 10,10,450,280,A@(*),B@(*),C@(*)`
   ..... Reads the graphic pattern in the area specified by the upper-left coordinates (10, 10) and lower-right coordinates (450, 280) on graphic screen 1 into A@(*), that on graphic screen 2 into B@(*), that on graphic screen 3 into C@(*). (Three screens are overlaid on the CRT.)

Note

1. If the defined size of A@ (*) is insufficient for reading all graphic patterns in the area specified by the upper-left coordinates (X, Y) and lower-right coordinates (U, V) on the graphic screen, the graphic patterns of up to the size of A@ (*) are read from the beginning of the specified area. If there are characters in the specified area, these characters are ignored (not read).
2. If a 1-dimensional array variable A@ (*), B@(*), and C@(*) are declared in the color mode, the graphic patterns on the blue, red, and green graphic screens are read into A@(*), B@(*), and C@(*), respectively.
3. If three of 1-dimensional array variables are declared, the graphic patterns on screen 1 (blue), screen 2 (red), and screen 3 (green) are read into these variables in this order. If two of such variables are declared, the graphic patterns on screens 1 and 2 are read and, if one such variable is declared, the graphic patterns on screen 1 are read.

Application example:

Inputs the graphic patterns (color display) in the two areas shown in the figure on the left and displays the graphic pattern as shown in the figure on the right.




```
10 DIM A@(3,51)*19 ..... Declares the 2-dimensional @-type character array variable (color mode) for area A.
20 DIM B@(3,51)*10 ..... Declares the 2-dimensional @-type character array variable (color mode) for area B.
30 GENTER 50,50,200,100,A@(*) ..... Reads the graphic pattern in area A into A@(*).
40 GENTER 400,250,475,300,B@(*) .. Reads the graphic pattern in area B into B@(*).
50 CLS ..... Clears the entire screen.
60 GDISP 50,50,200,100,A@(*) ..... Redisplays area A.
70 GDISP 201,101,276,100,B@(*) ..... Displays the graphic pattern, that was in area B, in the area following area A.
80 END
```

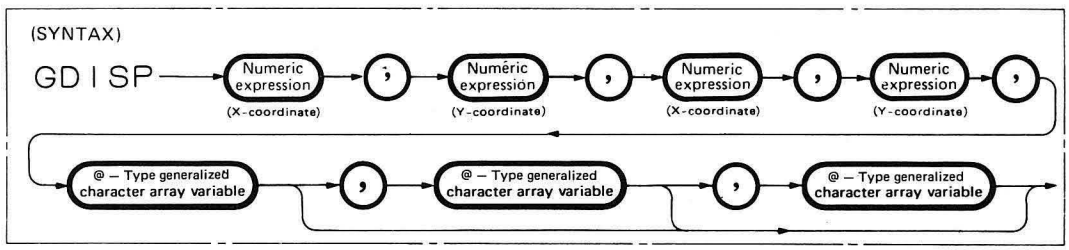
GDISP

Displays graphic patterns on the CRT.

〈Purpose of use〉
① To display graphic patterns displayed on the CRT, in another area of the same CRT.
② To display on the CRT arbitrary graphic information assigned to an array variable.

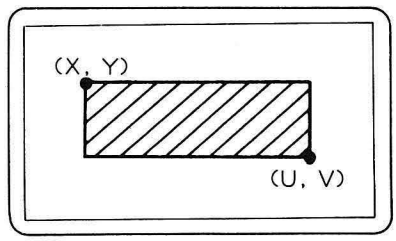
Format: GDISP X, Y, U, V, A@(*) [, B@(*) [, C@(*)]]

X : X-coordinate of upper-left corner
Y : Y-coordinate of upper-left corner
U : X-coordinate of lower-right corner
V : Y-coordinate of lower-right corner
A @ (*) : @ type generalized character array variable
B @ (*) : @ type generalized character array variable
C @ (*) : @ type generalized character array variable



Explanation:

1. Displays the graphic pattern information stored in the @-type generalized character array variable A @ (*), in the area (shaded part in the figure) specified by the upper-left coordinates (X, Y) and lower-right coordinates (U, V) on the graphic screen.



[Graphic screen]

The information of variable A @ (*) is the screen information read by the GENTER statement (page 265).

2. When three of the @-type generalize character array variables are specified, the graphic pattern stored in each variable is displayed overlaid.

Simple example:

```
30 GDISP 10,10,100,100,A@(*)
```

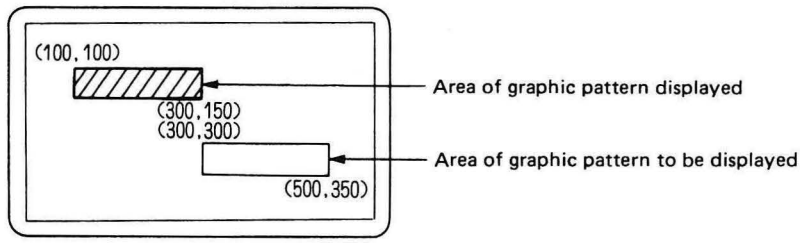
.....Displays the graphic pattern information stored in A @ (*), in the area specified by the upper-left coordinates (10, 10) and lower-right coordinates (100, 100) on the graphic screen.

Note

If A@(*) is greater than the size of the area specified by the upper-left coordinates (X, Y) and lower-right coordinates (U, V) on the graphic screen, information that can be stored in the area specified from the beginning of A@(*) is displayed on the screen. (No error occurs.) However, if A@(*) is less than the size of the specified area when smaller in the vertical (Y) direction, it results in an error. When smaller in the horizontal (X) direction, it is filled with spaces.

Application example:

Displays the graphic pattern (color display) displayed on the CRT, in another area of the same CRT.




```
10 DIM A@(3,51)*26
20 GENTER 100,100,300,150,A@(*)
30 GDISP 300,300,500,350,A@(*)
40 END
```

10 DIM A@(3,51)*26 ..... See Explanation 2 on the GENTER statement (page 265) for deciding the size of array A @.

20 GENTER 100,100,300,150,A@(*) ..... Reads the graphic pattern information in the area specified by the upper-left coordinates (100, 100) and lower-right coordinates (300, 150), into A @ (*).

30 GDISP 300,300,500,350,A@(*) ..... Displays the graphic pattern information read into A @ (*), in the area specified by the upper-left coordinates (300, 300) and lower-right coordinates (500, 350).

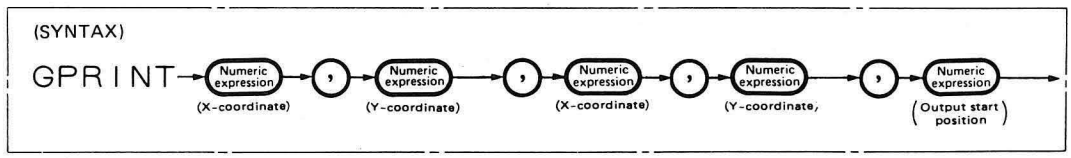
GPRINT

Outputs a graphic pattern to the printer.

〈Purpose of use〉
To output the graphic pattern displayed on the CRT to the printer.

Format: GPRINT X, Y, U, V, W

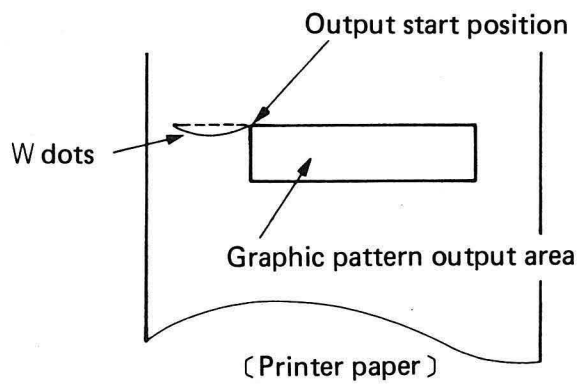
X : X-coordinate of the upper-left corner
Y : Y-coordinate of the upper-left corner
U : X-coordinate of the lower-right corner
V : Y-coordinate of the lower-right corner
W: Output start position



Explanation:

Outputs the graphic pattern displayed in the area specified by the upper-left coordinates (X,Y) and the lower-right coordinates (U,V) on the CRT from the output start position W (in dots) to the printer.

The graphic pattern is output to the following area:



Simple example:

```
20 GPRINT 100,100,500,200,10
```

·····Outputs the graphic information in the area specified by the upper-left coordinate (100, 100) and lower-right coordinates (500, 200) from the 10th dot (output start) position, to the printer.

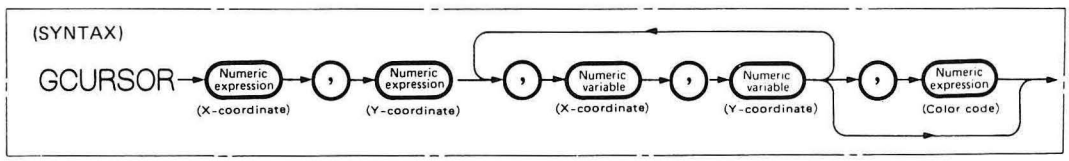
GCURSOR

Positions the graphic cursor*) and reads the graphic coordinates of the specified position.

〈Purpose of use〉
For ascertaining the position (coordinates) of a certain point on the graphic screen.

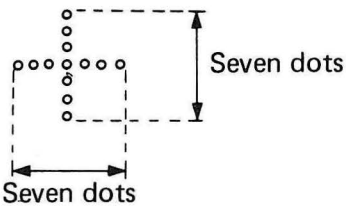
Format: GCURSOR X, Y, [A, B] [, Z]

X: X-coordinate
Y: Y-coordinate
A: Numeric variable for reading X-coordinate
B: Numeric variable for reading Y-coordinate
Z: Graphic cursor color code (0 to 7)
See the COLOR statement (page 237) for the color code.
Black cannot be displayed directly on the background.



Explanation:

1. On the graphic screen, moves the graphic cursor to the coordinates (X, Y) and displays it in color Z. Since input is now expected, move the graphic cursor with arrow keys ([→], [←], [↑], and [↓] keys), then press the [ENTER] key to read the graphic coordinates of the specified position into A and B.
2. The graphic cursor format is a cross of seven vertical dots and seven horizontal dots. The cursor moves until this crosspoint is on the point of graphic coordinates (X, Y).



Graphic cursor

3. Input is expected the same number of times (A, B) is specified. Move the graphic cursor and press the [ENTER] key each time input is required.
4. When graphic cursor color code Z is omitted, the color specified in the preceding GCOLOR statement (see page 242) is used.
5. This statement is valid only for a graphic screen displayed on CRT-1.

*) Graphic cursor: Cursor used on a graphic screen

Simple example:

1. `10 GCURSOR 50,50,A,B,3`

..... Moves the graphic cursor to the position of coordinates (50, 50) and displays it in green (color code 3). Since input is expected in this case, move the graphic cursor to the specified position, then press the [ENTER] key. By this operation, the graphic coordinates of this position are read into (A, B).

2. `10 GCURSOR 100,100,A1,B1,A2,B2,1`

..... Moves the graphic cursor to the position of coordinates (100, 100) and displays it in yellow (color code 1). Since input is expected in this case, move the graphic cursor to the specified position, then press the [ENTER] key. By this operation, the graphic coordinates of this position are read into (A1, B1). Since input is expected again, read the graphic coordinates into (A2, B2) again in the same way.

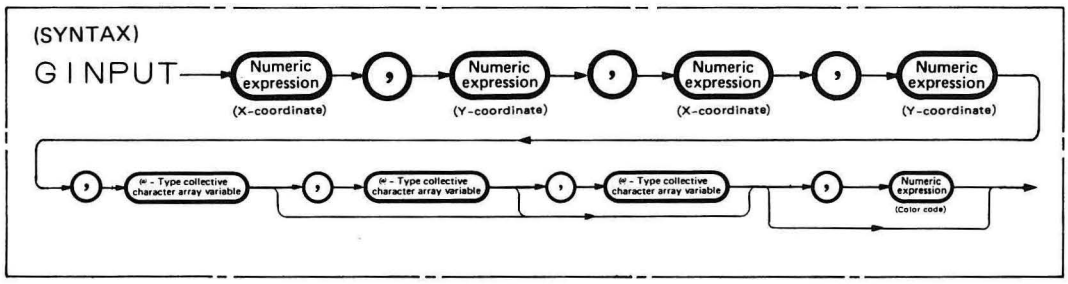
GINPUT

Inputs a graphic pattern with the keyboard and reads (stores) it into a variable.

〈Purpose of use〉
For inputting a graphic pattern with the keyboard (plotting graphic, etc.) and reading it to the specified array variable.

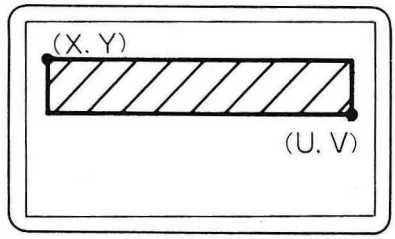
Format: G I NPUT X, Y, U, V, A@(*) [, B@(*) [, C@(*)]] [, Z]

X : X-coordinate of upper-left corner
Y : Y-coordinate of upper-left corner
U : X-coordinate of lower-right corner
V : Y-coordinate of lower-right corner
A@(*): @-type collective character array variable
B@(*): @-type collective character array variable
C@(*): @-type collective character array variable
Z: Graphic pattern color code (0 to 7)
See the COLOR statement (page 237) for the color code.
Black cannot be displayed on the background.



Explanation:

1. Inputs the graphic pattern from the keyboard to the area (hatched area in the figure below) specified with the coordinates (X, Y) for the upper-left and coordinates (U, V) of the lower-right of the graphic screen, then reads the specified area contents from the upper-left sequentially into the @-type generalized character array variable.
   For deciding the size of the @-type collective character array, see Explanation 3 of the GENTER statement (page 265).



[Graphic screen]

2. When this statement is executed, the graphic cursor is displayed at the upper-left of the specified area on the screen and input of the graphic pattern with the keyboard is enabled.

   Use the following keys for inputting graphic patterns:

   [S] key : Display mode
   [R] key : Erase mode
   [M] key : Move mode
   Arrow keys ([↑], [↓], [←] and [→] keys)
   : Move the graphic cursor horizontally and vertically.
   [ENTER] key : Completes input of graphic pattern.
   [0] to [7] keys : When one of these keys is pressed, the pattern color is changed to the color of the specified color code. (The color code is the same as that specified in the COLOR statement.)
   Black cannot be displayed directly on the background.

   See "Helpful hint" for the graphic pattern input method.
3. When the color Z of the graphic pattern is omitted, the color specified by the previous GCOLOR statement (page 242) is used.

Simple example:

1. `20 GINPUT 10,10,50,50,A@(*),5` ··· Inputs a graphic pattern in red (color code 5) from the keyboard to the area specified with the upper-left (10, 10) and lower-right (50, 50) then reads the area contents into the generalized array variable A@(*).

2. `20 GINPUT 100,100,110,105,B@(*)`
   ····· Inputs graphic pattern from the keyboard into the area specified with the upper-left (100, 100) and lower-right (110, 105) then reads the area contents into B@(*). (The graphic pattern has the color specified in the GCOLOR statement.)

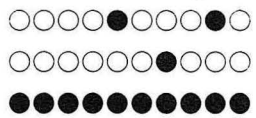
Note

When this statement is executed, the entire screen information displayed in the specified area is to be input, even when no data is input with the keyboard.

<Detailed information>

Input a graphic pattern as follows. (See the figure below.)

(1,1) [Status before input]

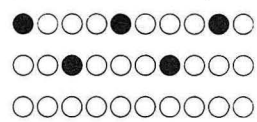


(10, 3)

In the above figure, all dot patterns (painted parts) are in red.

Graphic pattern input

(1,1) [Status after input]



(10, 3)

In the above figure, all dot patterns except the two points in line 2 ((3, 2) and (7, 2)) displayed on the CRT are in red. For the two dots in line 2, (3, 2) is in blue and (7, 2) is in green.

```
GINPUT 1,1,10,3,A@(*),5
```

When the above statement is executed in a program, the graphic cursor is displayed at the upper-left (1, 1) in the specified area.

① Set the display mode by pressing the [S] key, then move the graphic cursor one column right (to (2, 1)) with arrow key [→]. Then, set the move mode by pressing the [M] key and move the graphic cursor eight columns right (to (10, 1)) with arrow key [→].

② To input the second line, move the graphic cursor one line downward (to (10, 2)) with arrow key [↓], then seven columns left (to (3, 2)) with arrow key [←]. Set the display mode by pressing the [S] key, then move one column left (to (2, 2)) with arrow key [←]. Set the move mode by pressing the [M] key, then move two columns left (to (1, 2)) with arrow key [←].

③ When the graphic cursor is placed at the left end (1, 3) of the third line, press the [R] key to set the erase mode, then move nine columns right (to (10, 3)) with arrow key [→]. Thus, inputting graphic patterns to lines 1 to 3 is completed. All graphic patterns currently input are displayed in red (because color code 5 (red) is specified in the GINPUT statement). To display (3, 2) in blue and (7, 2) in green in the second line, set the move mode and move the current graphic cursor (10, 3) one line upward (to (10, 2)) with arrow key [↑], move three columns leftwatd (to (7, 2)) with arrow key [←], and set the display mode by pressing the [S] key, then change the color to green by pressing the [3] key. Move the graphic cursor one column left (to (6, 2)) with arrow key [←], set the move mode with the [M] key, then move three columns left (to (3, 2)) with arrow key [←]. Set the display mode by pressing the [S] key, then set the color to blue by pressing the [6] key. Thus, input of all dot patterns is completed. When the [ENTER] key is pressed, the dot patterns are read sequentially from the upperleft end (1, 1) of the specified area into array variable A@ (*).

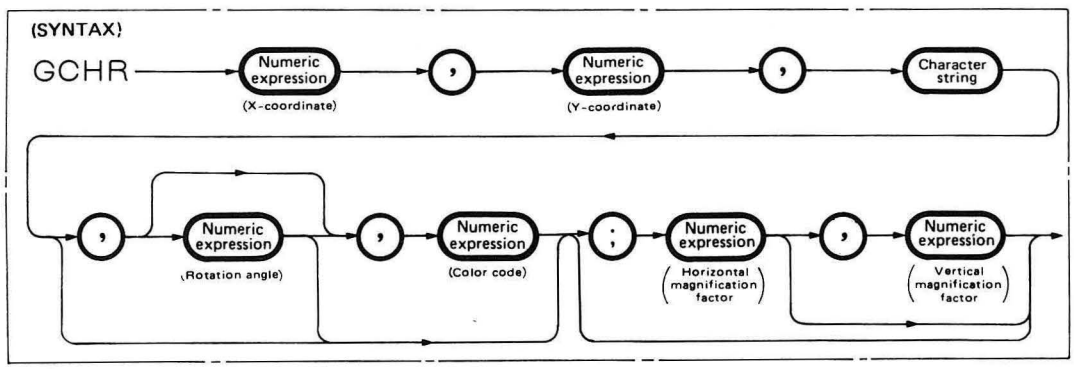
GCHR

Displays graphic characters*).

〈Purpose of use〉
① For using characters displayed on a text screen also on a graphic screen.
② For expanding or rotating characters on the display.

Format: GCHR X, Y, C [, W [, Z] | [W], Z] [; U [, V] | [U], V]

X : X-coordinate of lower-left corner
Y : Y-coordinate of lower-left corner
C : Character to be displayed
W: Rotation angle
0 . . . . . 0° (no rotation)
1 . . . . . 90° rotation
2 . . . . . 180° rotation
3 . . . . . 270° rotation
Z : Graphic character color code (0 to 7)
See the COLOR statement (page 237) for the color code.
Black cannot be displayed directly on the background.
U : Magnification factor in vertical direction (1 to 16)
V : Magnification factor in horizontal direction (1 to 16)

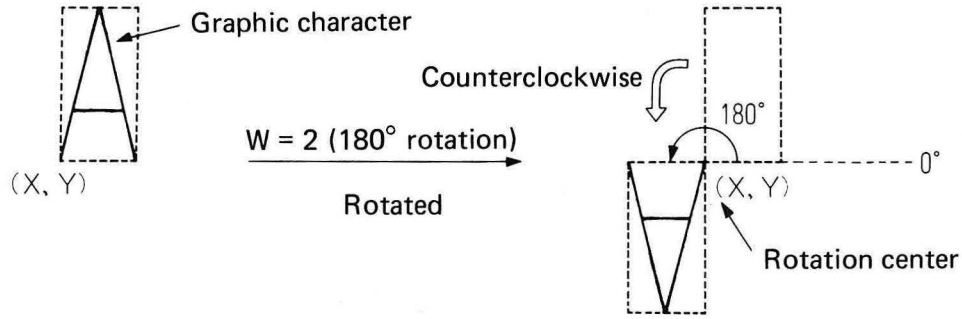


Explanation:

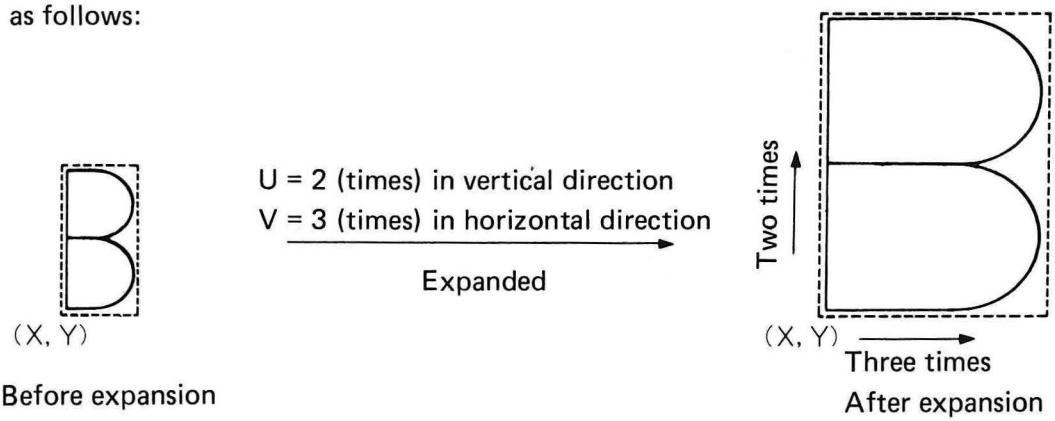
1. Positions the lower-left end of the graphic character to be displayed on the graphic screen at the coordinate (X, Y) and displays the character specified in character string C in color Z. Rotates the specified graphic character by the rotation angle specified in numeric expression W and expands the graphic character U times in the vertical direction, and V times in the horizontal direction.

*) Graphic character: ASCII character displayed on the graphic screen

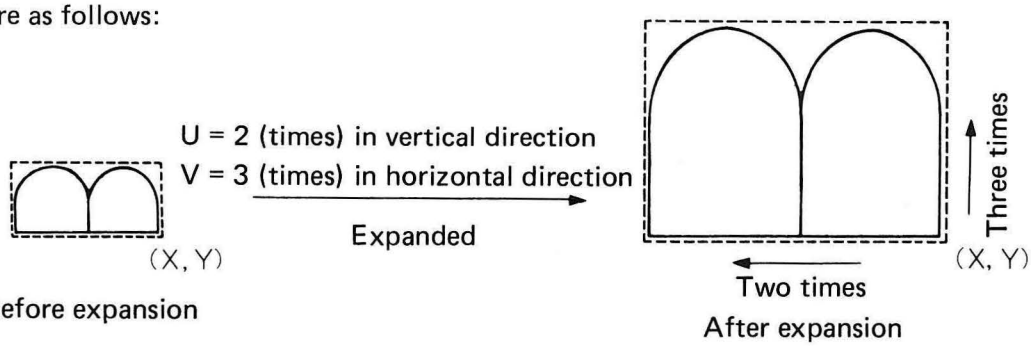
2. When rotating the graphic character, this character is rotated, centered at the graphic character lower-left end position (X, Y) counterclockwise. (See the figure below.)



3. When expanding the graphic character, the graphic character is expanded from the lower-left end of the graphic character (X, Y) in the vertical and horizontal directions as follows:



When expanding a rotated graphic character, the graphic character lower-left position (X, Y), expansion in the vertical direction, and expansion in the horizontal direction are as follows:



The lower-left position (X, Y) of the graphic character is based on the condition before rotating the graphic character.

4. When unmeric expression W specifying the rotation angle is omitted, 0° (no rotation) is assumed.
5. When graphic character color code Z is omitted, the color specified in the preceding GCOLOR statement (see page 242) is used.
6. When magnification factor U in the vertical direction or V in the horizontal direction for expanding the graphic character is omitted, magnification 1 is assumed. (no expansion).

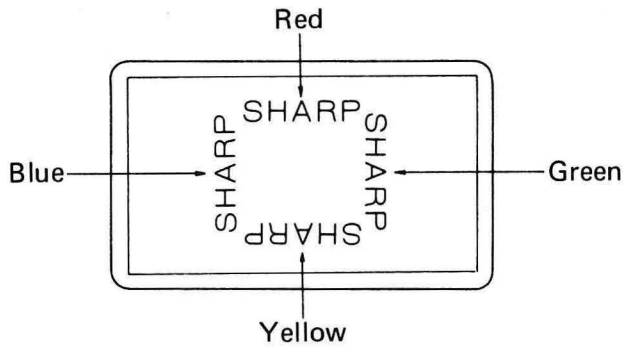
Simple example:

1. `10 GCHR 100,100,"A",2,5` ..... Rotates graphic character A centered at the lower-left end position (100, 100) and displays it in red (color code 5).
   (See the figure of Explanation 2.)

2. `10 GCHR 100,100,"B",0,1:2,3`
   ..... Expands graphic character B two times in the vertical direction and three times in the horizontal direction, from the lower-left end position (100, 100), and displays it in yellow (color code 1).
   (See the figure of Explanation 3.)

Application example:

Display the following graphic characters:



Graphic screen

```
10 SDISP "123C":ODISP "123":CLS
20 SCALE OFF
30 GCHR 199,79,"SHARP",0,5;2,6
40 GCHR 199,319,"SHARP",1,6;6,2
50 GCHR 439,319,"SHARP",2,1;2,6
60 GCHR 439,79,"SHARP",3,3;6,2
70 END
```

Initializes the graphic screen.

Displays graphic characters SHARP with no rotation in red (color code 5) by expanding two times in the vertical direction and six times in the horizontal direction. The lower-left end of the graphic characters is (199, 79).

Displays graphic characters SHARP with 90° rotation in blue (color code 6) by expanding two times in the vertical direction and six times in the horizontal direction. The lower-left end of the graphic characters is (199, 319).

Displays graphic characters SHARP with 180° rotation in yellow (color code 1) by expanding two times in the vertical direction and six times in the horizontal direction. The lower-left end of the graphic characters is (439, 319).

Displays graphic characters SHARP with 270° rotation in green (color code 3) by expanding two times in the vertical direction and six times in the horizontal direction. The lower-left end of the graphic characters is (439, 79).

Note

This statement can display the characters from ASCII codes &20 to &7A. (Refer to Appendix 1 "ASCII code List" in the Owner's Manual.)

7. STATEMENTS RELATED TO MACHINE LANGUAGE

This section explains the statements related to machine language. Be sure to have a through understanding of machine language before using these statements.

NEW ※
ERASE ※
POKE (PEEK) ※
USR

The statements marked with ※ can also be used in manual operations.
The USR statement can be used only in a program.

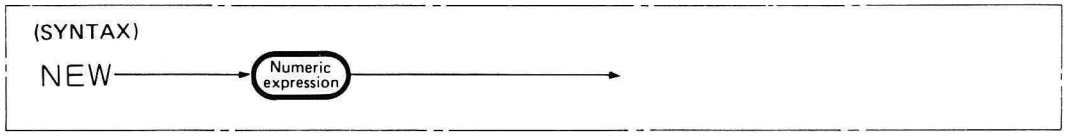
NEW

Erases all program lines and variables and reserves the machine language program user area.

〈Purpose of use〉
To reserve the user micro area*) before using machine language.

Format: NEW X

X: Last address (&9600 ≦ X ≦&BFFF)

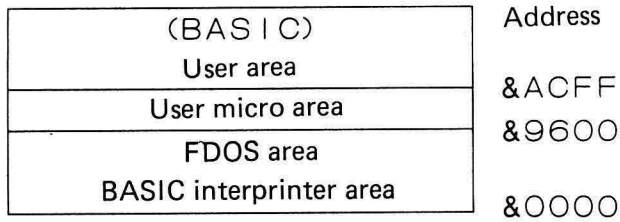


Explanation:

1. Erases all program lines and variables and reserves the area from memory address &9600 to the address of the numeric expression X value as the user micro area. BASIC programs and their data cannot be written in this area. The numeric expression X value must be in the range &9600 to &BFFF.
2. To return the user micro area to the normal user area, execute this statement specifying 0 in the numeric expression X value.

Simple example:

1. NEW &ACFF [ENTER] ····· Reserves the area of memory from &9600 to &ACFF as the user micro area and, at the same time, erases all program lines and variables. (See the figure below).



2. NEW 0 [ENTER] ····· Returns the area of memory following address &9600 to the normal user area and, at the same time, erases all program lines and variables.

Note

1. When this statement is executed, all the program lines are erased. To reserve the user micro area in a program, use the ERASE statement (explained next).
2. The start address of the user micro-area may be changed when the systems software is reversed, and also using when the graphic RAM is extended.
   Check this address using the DEBUG command of FDOS (page 55).

*) User micro area: Area for storing user machine program

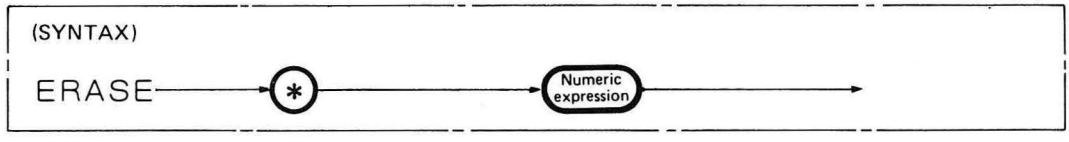
ERASE

Erases all variables and reserves the machine language program user area.

〈Purpose of use〉
For reserving the user micro area before using machine language.

Format: ERASE * X

X: Last address (&9600 ≦ X ≦ &BFFF)



Explanation:

1. Erase all variables and reserves the area from memory address &9600 to the address indicated by the valve of numeric expression X as the user micro area.
   Therefore, BASIC programs and their data cannot be written in this area.
   In this case, the program is not erased.
   The value of numeric expression X must be in the range of &9600 to &BFFF.
2. To return the user micro-area to the normal user area, execute this statement by specifying 0 in the numeric expression X value.

Simple example:

1. `10 ERASE *&ACFF` ..... Reserves the area from memory address &9600 to &ACFF as the user micro area and, at the same time, erases all variables.
2. `100 ERASE *0` ..... Returns the area from memory address &9600 to the normal user area and, at the same time, erases all variables.

Note

1. This statement cannot be used in a subprogram (page 157).
2. The start address of the user micro area may be changed when the systems software is revised, and also using, when the graphic RAM is extended. Check the start address using the DEBUG command of FDOS (page 55).

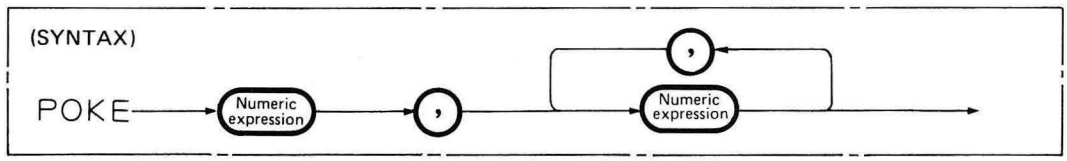
POKE (PEEK)

Writes (Inputs) data to the specified address.

Format: POKE X, (Y)

X: Memory address (&9600 to &CFFF)

Y: Input data (&00 (0) to &FF (255))



Explanation:

1. Writes data of numeric expression Y to the memory address specified in numeric expression X. See the memory map (refer to the Appendix) for memory addresses. Since the POKE statement can be executed for memory address &9600 to &CFFF, the value of numeric expression X must be in the range of &9600 to &CFFF. The value of write data numeric expression Y must be in the range &00 (0) to &FF (255).
2. When numeric expression Y is repeated, data is written sequentially at memory address X, X + 1, X + 2, . . . (incremented by 1). Therefore, if the maximum value &CFFF is specified in numeric expression X, numeric expression Y cannot be repeated.
3. The PEEK function (see page 29) has the opposite function of the POKE statement. The PEEK function gives the data of the specified memory address.

Simple example:

```
10 POKE &A000,&C1
```
····· Writes data &C1 (193) to memory address &A000.

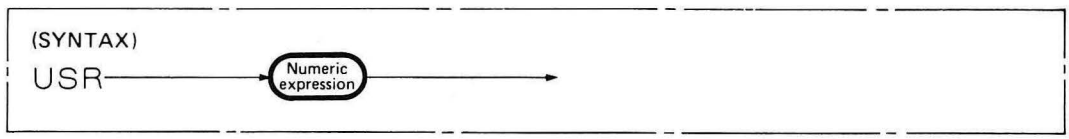
USR

Calls a machine language routine.

● **Function 1:** When the machine language routine has no arguments.

Format: USR X

X: Memory address



Explanation:

Transfers control to the machine language routine beginning at memory address X.

Simple example:

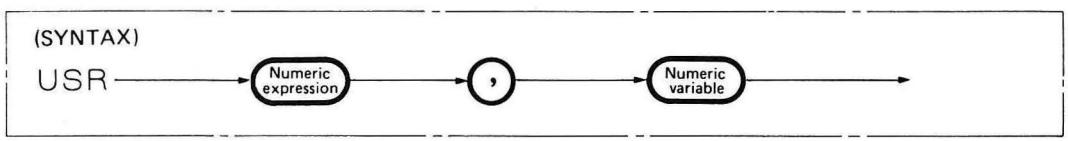
```
10 USR &9800
```
..... Transfers control to the machine language routine beginning at memory address &9800.

● **Function 2:** When the machine language routine has an argument.

Format: USR X, A

X: Memory address

A: Argument



Explanation:

Transfers control to the machine language routine beginning at memory address X with argument A.

The argument value is converted to a binary value and set in the HL register. The most segnificant digit of the HL register indicates the sign; 0 indicates plus and 1 indicates minus.

① When returning from the machine language routine to the main routine, specify RET.

② When setting a value to the argument, return by setting the carry flag.
Set the argument value in the HL register.

```
AND A
RET
```

③ If HL = 0Q20H is specified for example, 32 is set as the argument value.
When the argument value is not changed, return by setting the carry flag.

```
SCF
RET
```

Simple example:

USR &A000, B ..... Transfers control to the machine language routine beginning at memory address &A000 with argument B. For example, when B = 90, jumps to memory address &A000 assuming that HL = 005AH.
When B = –90, jump address is &A000 and HL = 805AH.

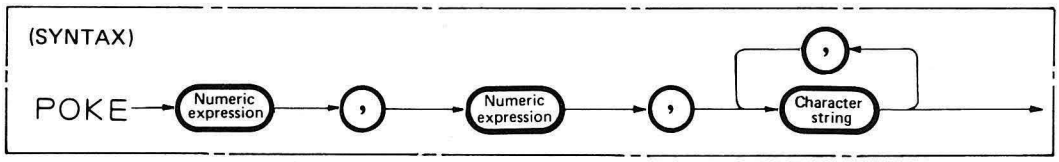
● **Function 3:** When the machine language routine has multiple arguments.

Format: USR X, Y, (M)

X : Execution-start memory address

Y : Argument start address

M : Arguments

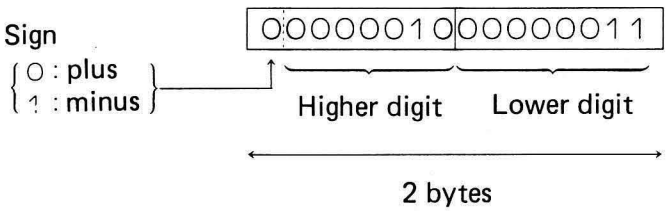


Explanation:

Saves arguments M from start address Y and transfers control to the machine language routine beginning at memory address X. Arguments are saved in the following formats.

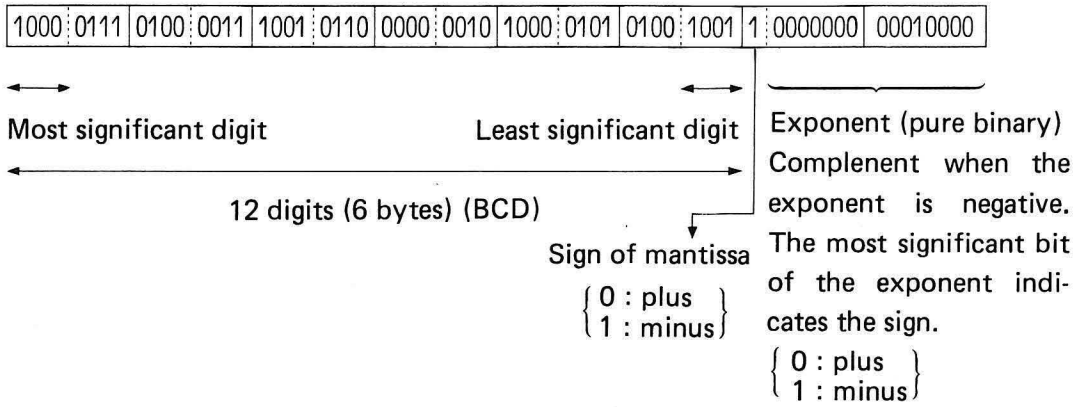
① Integer-type numeric variable

Example: A% = 515



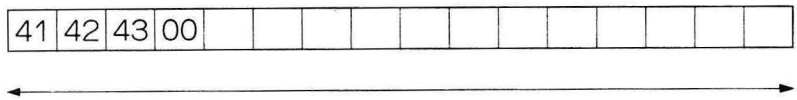
② Real-type numeric variable

Example: A# = –8.74396028549 x 10^{16}



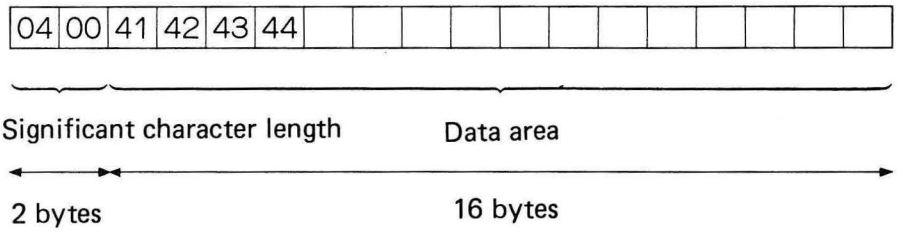
③ $-type character variable

Example: A$ = "ABC"



④ @-type character variable

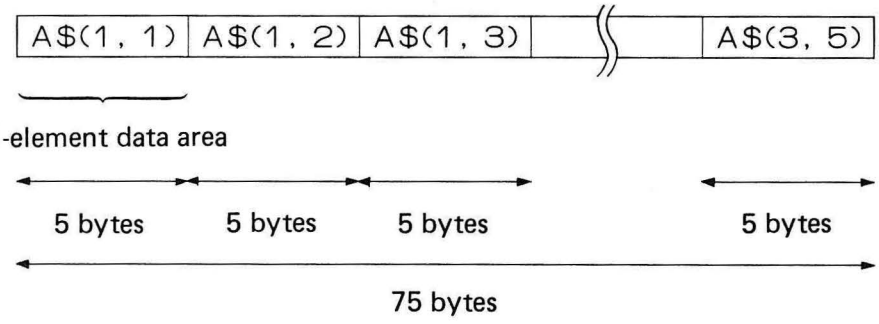
Example: A@ = "ABCD"



If declared, the number of bytes in data area is as declared by DIM.

⑤ Array variable

Example: Variable A$(*) declared by DIM A$ (3, 5) * 5



For an @-type character variable, the significant character length is saved in the 2 bytes preceding each element.

8. RS-232C INTERFACE CONTROL STATEMENTS

This section explains the following statements for controlling the RS-232C interface:

CHANNEL
SEND
RCV
SENREV
OPCHNL
ROLLING
RLIST
TERM

All these statements can also be used in manual operations.
For the explanation and operation of the RS-232C interface, refer to "RS-232C interface" in the Owner's Manual.

CHANNEL

Sets the channel to be used and specifies the data input/output format.

Format: CHANNEL X [, Y, C]

X: Channel number (0, 1, or 2)

Y: Baud rate (110, 200, 300, 600, 1200, 2400, 4800, or 9600)

C: Data format

Specify the data format as follows:

"data-bit-length, parity-format, stop-bit-length"

(1) data-bit-length

7 ..... 7-bit code

8 ..... 8-bit code

(2) Parity format

E ..... Even parity

O ..... Odd parity

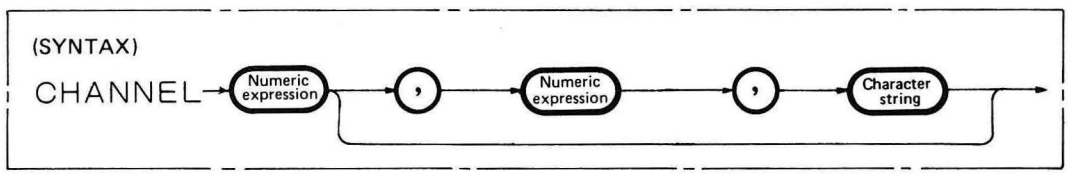
N ..... No parity

(3) Stop-bit-length

1 ..... 1 bit

1.5 ..... 1.5 bits

2 ..... 2 bits



Explanation:

1. Declares the use of channel number X and sets the channel data input/output format (the baud rate in the numeric expression Y value (bits/second) and the data format in the contents of character string C). Once this statement is set, SEND, RCV, and SENREV statements are executed via this channel.
   The data format is specified in three characters indicating the data bit length, parity format, and stop bit length respectively. (Example: "7E1.5")
2. When numeric expression Y and character string C are omitted, the data input/output format previously set for the channel is used. If the data input/output format is not previously set or after a NEW command is executed, the baud rate is set to 300 bits/second, the data format is set to the 7-bit code data, the parity format is set to even parity, and the stop bit length is set to 1.
3. When the power is turned on, the channel number is set to 0.
   (The standard interface of this system is for channel 0.)

Simple example:

1. `10 CHANNEL 0,600,"8N1"` ..... Declares the use of channel number 0 and sets the channel data input/output formats; i.e. baud rate to 600 bits/second, data format to 8-bit code data, the parity format to no parity, and the stop bit length to 1.
2. `10 CHANNEL 0` ..... Declares the use of channel number 0 and sets the channel data input/output formats to that previously specified.

Note

1. When numeric expression Y and character string C are not omitted for inputting or outputting the ASCII 7-bit code data format, the shift-in/shift-out (SI/SO) status is set to SI. When numeric expression Y and character string C are omitted, the SI/SO status is not changed by execution of this instruction.
2. The standard interface of this system is for channel 0. Channels 1 and 2 can be used by installing the RS-232C serial interface [MZ-1E01] (option) on the expansion slot (option). For this installation and channel numbers, refer to the instruction manual of the interface.

[Shift-in/shift-out (SI/SO)
The ASCII 7-bit code system has SI and SO statuses and 7-bit code data can be used as 8-bit code data.
The SO code (&0E) changes from the SI status to SO status and the SI code (&0F) changes from the SO status to SI status. (There are SO and SI codes in both statuses.) Therefore, the SO and SI codes are included in ASCII 7-bit code data. This must be taken into account if the program has a limited data size.
These SI and SO statuses are processed independently for input and output on individual channels.
The SI status is set automatically when the power is turned on, or after a CHANNEL instruction (setting the baud rate or data format) or OPCHNL instruction is executed. (After an OPCHNL instruction is executed, only the input instruction status is set.)

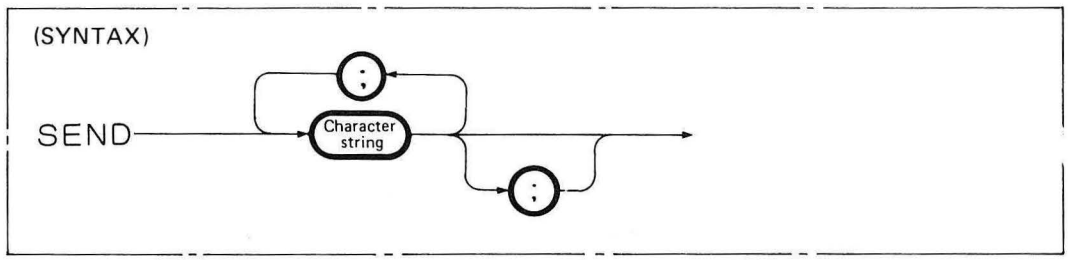
SEND

Outputs data according to the setting by CHANNEL statement.

● **Function 1:** Outputs character data.

Format: SEND 《C》〔;〕

C: Character data



Explanation:

1. When semicolon (;) is omitted at the end of the last character string, the character data indicated in character string C is output and the [CR] code (&0D) and [LF] code (&0A) are output automatically at the end of data.
2. When semicolon (;) is specified at the end of the last character string, only the character data indicated in character string C is output and the [CR] and [LF] codes are not output at the end of data.
3. A generalized array variable (example: A$(*)) can be specified as a character string. In this case, characters are output according to the size specified for each element. [NULL] codes are output to the part after the actual character data.
4. For the output format when repeating character string C, see Function 1 of the PRINT statement (page 110).

Simple example:

1. `10 SEND "ABC"` ..... Outputs character string (character constant) "ABC" followed by the [CR] and [LF] codes.
2. `10 SEND "ABC";` ..... Outputs only character string (character constant) "ABC".
3. `10 SEND A$(*)` ..... Assume that the contents of generalized character array variable A$(*) are as follows:

AA	BBBB	CCC
D	EEE	FF
GGG	H	I I
JJJJ	KK	LLL
M	NNNN	

(Array declared as DIM A$ (3, 5) * 4)

Outputs data in the following format (8-bit code).

[A][A][NL][NL][B][B][B][B][C][C][C][NL][D][NL][NL][NL][E][E][E][NL]
[F][F][NL][NL][G][G][G][NL][H][NL][NL][NL][I][I][NL][NL][J][J][J][J]
[K][K][NL][NL][L][L][L][NL][M][NL][NL][NL][N][N][N][N][NL][NL][NL][NL]
[CR][LF]

4. `10 SEND "ABC";"D";"EF";`

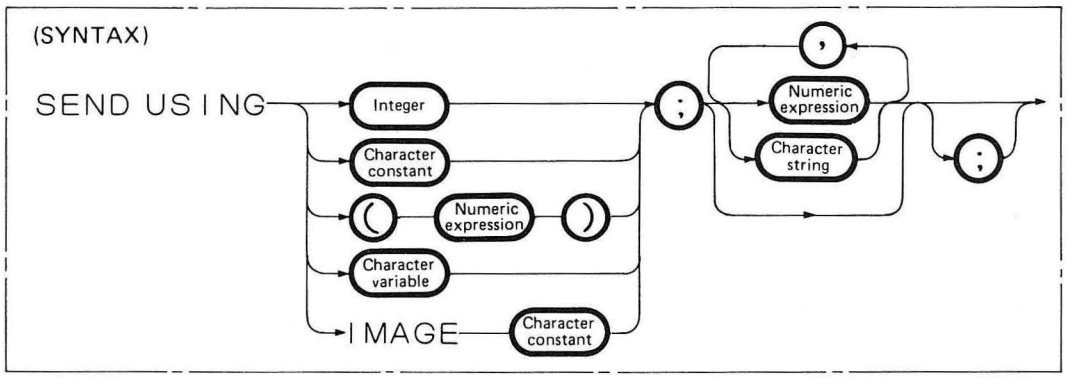
Outputs data in the following format:

[A][B][C][D][E][F]

● **Function 2:** Outputs character or numeric data using the USING statement.

Format: SEND USING n | S | (X) | A$ | IMAGE T;[{Y | C}][;]

n : Line number
S : Line label name
X : Line number (numeric expression)
A$: Character variable indicating image symbols
T : Image symbol
Y : Numeric data to be output
C : Character data to be output



Explanation:

1. Outputs the numeric data indicated by numeric expression Y and character data indicated by character string C in the format specified in the IMAGE statement.
2. The format is the same as that explained in Function 2 for the PRINT statement (page 110). Also see the IMAGE statement (page 113).
3. When semicolon (;) is omitted at the end of the last numeric expression or character string, the [LF] code is output automatically at the end of data. (The [CR] code is not output.) When [Y | C] is omitted, nothing is output.
   When semicolon (;) is specified at the end of the last numeric expression or character string, only the data is output.
   The [CR] and [LF] codes can be specified in the IMAGE statement.

Simple example:

```
10 H$="TRANSISTORS" :K=1320
20 SEND USING 100;H$,K
30 END
100 IMAGE '11A 5X "PRICE" 4X $#,### C'
```

Outputs data in the following format:

[T][R][A][N][S][I][S][T][O][R][S][SP][SP][SP][SP][SP][P][R][I][C][E][SP][SP][SP][SP]
[$][1][.][3][2][0][CR][LF]

2.
```
10 H$="PEN":K=100
20 SEND USING "$";H$,K;
30 END
100 "$":IMAGE '5A 5X "PRICE" 4X $#,### C'
```

Outputs data in the following format:

P E N SP SP SP SP SP SP SP P R I C E SP SP SP SP SP SP $ 1
0 0 CR

3.
```
10 DIM,P$(1)*25:P$(1)='5A 5X "PRICE" 4X $#,### L'
20 H$="PEN":K=100
30 SEND USING P$(1);H$,K
40 END
```

Outputs data in the following format:

P E N SP SP SP SP SP SP SP P R I C E SP SP SP SP SP SP
$ 1 0 0 LF LF

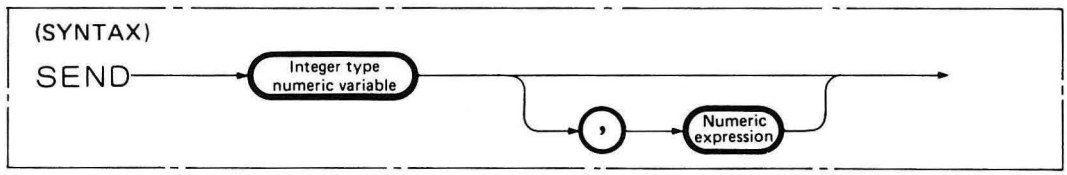
For the above three examples, note the difference in formats. (Referring to the PRINT statement (page 110), the IMAGE statement (page 113), and the image symbol list (page 113).)

● **Function 3:** Outputs the numeric value indicated by the integer-type numeric variable in binary format.

Format: SEND A% [, X]

A%: Numeric data (variable)

X : Number of output bytes



Explanation:

1. When numeric expression X is omitted, outputs one lower byte of the numeric data (2-byte binary value) indicated in integer-type numeric variable A% in binary format. The high order byte is ignored.
   Generalized integer-type numeric array variable (example: B%(*)) can be specified as an integer-type numeric variable. In this case, one lower byte of the numeric data for all elements is output.
   The CR and LF codes are not output at the end of data.
2. When numeric expression X is specified, a generalized integer-type numeric array variable is given as the integer-type numeric variable. In this case, specify the number of elements in numeric expression X as the number of output bytes.
   The CR and LF codes are not output at the end of data.

Simple example:

10 SEND B%(*),10 ..... Outputs one lower byte of the numeric data for 10 elements of integer-type numeric variable B%(*) in binary format.

Note

For processing of Function 3, use 8-bit code data. The [NULL] codes can also be output. (Example: 10 SEND 0%)

[NL] indicates a null code (&00).
[SP] indicates a space character.
[CR] indicates CR (carriage return) code (&0D).
[LF] indicates LF (line feed) code (&0A).

RCV

Inputs data according to the setting in CHANNEL statement.

● **Function 1:** Inputs data into a character variable.

Format: RCV [/] A$, X, C [, | ; D]

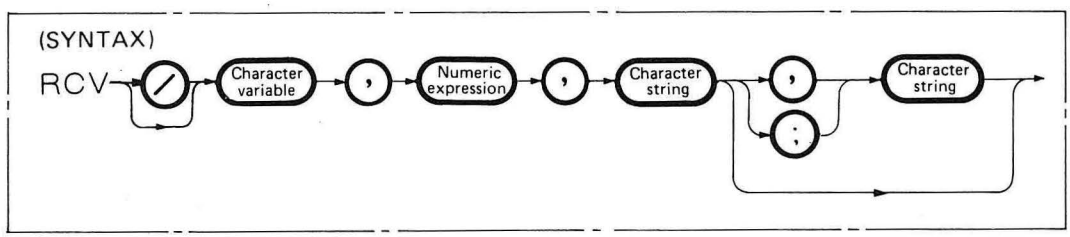
/ : Echo-back function on host side

A$: Character variable for inputting data

X : Waiting time (0 to 255)

C : End code 1

D : End code 2



Explanation:

1. When [/] is omitted, inputs data up to the end code (explained later) specified by character strings C and D into character variable A$. Also the end code is input to the variable. If the variable area is exceeded before inputting the end code, an error (ERROR 109) occurs. Also the [NULL] code is input to the variable (@-type variable). Generalized array variable (example: A$(*)) can be specified as character variable A$. For character strings C and D, only the first character is valid as the end code.
2. When [/] is specified, the echo-back function on the host side*) is enabled.
   To use the echo-back function on the host side for 7-bit code, check the shift-in/shift-out (SI/SO) status on both input and output sides.
3. Numeric expression X indicates the waiting time in 0.1 second units. For example, if the numeric expression X value is 150, the waiting time is 15 seconds. If data is not input within the waiting time, an error (ERROR 103) occurs. The numeric expression X value must be in the range of 0 to 255. If 0 is specified as the X value, the system waits until data is input (waiting time: ∞).
   In any case, the waiting status can be reset with the HALT button.
4. When character string D is omitted, inputs data up to the end code indicated in character string C.

*) Echo-back function on host side: Each time 1-byte data is input, this function outputs this data. This function is effective for a unit with a low baud rate (1200 or fewer bits/second).

5. When character string D is not omitted (there are two end codes) and a comma (,) is specified between character strings C and D, inputs data when the end code specified by either character strings C and D is reached.
   When a semicolon (;) is used, inputs data until the end codes given by character strings C and D are input in that order.
   For example, if "A", "B" is specified, data is input until the [A] or [B] code is input; if "A";"B" is specified, data is input until the [A] and [B] codes are input consecutively.

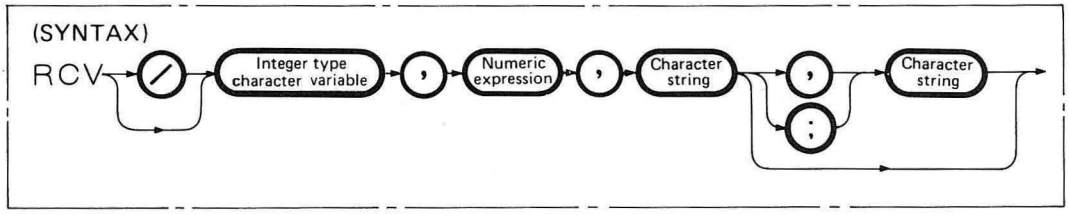
Simple example:

1. `10 RCV A$,0,"@"` ..... Inputs character data until the end code [@] into character variable A$. The waiting time is ∞ .
2. `10 RCV A$,150,"@","A"` ..... Inputs character data until end code [@] or [A] is reached into character variable A$. The waiting time is 150 x 0.1 = 15 seconds.
3. `10 RCV A$,150,"@";"A"` ..... Performs the same processing as Simple example 2 until the [@] and [A] codes are input consecutively.
4. `10 RCV /A$,150,"@";"A"` ..... Performs the same processing as Simple example 2, with the echo-back function on the host side.

● **Function 2:** Inputs data into an integer-type numeric variable.

Format: RCV (/) A%, X, C (, | ; D)

/ : Echo-back function on host side
A% : Integer-type character variable for inputting data
X : Waiting time (0 to 255)
C : End code 1
D : End code 2



Explanation:

1. Input 1-byte numeric data (binary data), for each variable (2 bytes), until the end code indicated in character strings C and D, into integer-type numeric variable A%. In this case, the 1-byte binary data is input to the low order byte of the integer-type numeric variable and &00 is input to the high order byte. Therefore, an integer in the range of 0 to 255 is input to integer-type numeric variable A%. Normally, specify a generalized integer-type numeric array variable (example: C%(*)) in integer-type numeric variable A%.
2. Explanations of slash (/), numeric expression X and so forth follow that of Function 1.

Simple example:

```
10 RCV B%(*),100,"A"
```
..... Inputs 1-byte numeric data (binary data) up to end code "A" (&41) sequentially into generalized array variable B%(*).
(The waiting time for each element is 100 x 0.1 = 10 seconds.)

Note

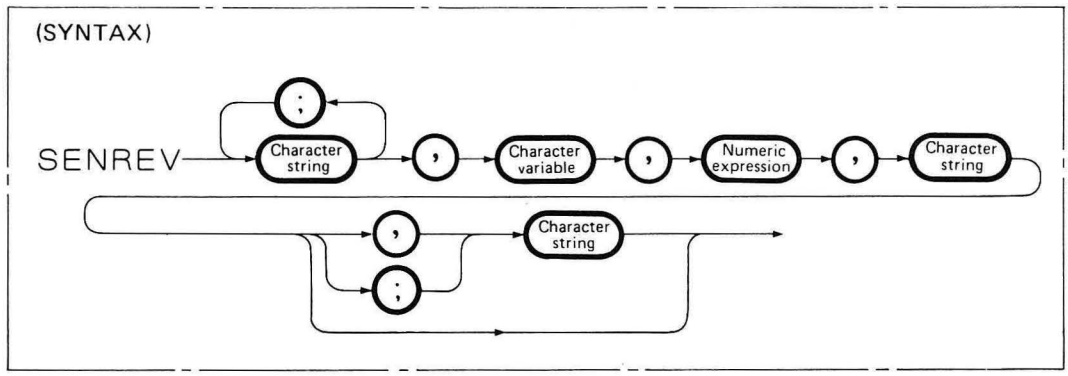
1. External data can be input only in the wait state during execution of this statement or the SENREV statement (explained next).
2. If the end code is set to CHR$ &9F for function 1 in the 7-bit code data system, only 7-bit code data is used for processing regardless of the SI/SO status.
3. Use 8-bit code data for Function 2.

SENREV

Outputs data according to the setting in CHANNEL statement and then immediately inputs data.

● **Function 1:** Outputs character data and then inputs data into character variable.
Format:SENREV 《C》, A$, X, D 〔, | ; E〕

C : Character data
A$: Character variable for inputting data
X : Waiting time (0 – 255)
D : End code 1
E : End code 2



Explanation:

1. Outpus character data indicated in character string C and inputs data up to the end code indicated in character strings D and E into character variable A$ immediately after data output.
2. For other details, see Function 1 of SEND instruction and Function 1 of RCV instruction. The [CR] code (&0D) and [LF] code (&0A) are not output at the end of output data. There is no echo-back function for inputting data.

Simple example:

```
10 SENREV "READY",D$,0,CHR$ &0D
```

..... Outputs "READY" and then immediately inputs data until the [CR] code into variable D$. (Waiting time is ∞ .)

Note

When specifying a generalized array variable in character string C, specify the output of the end code (output side) at the end of the output data. For example, if the [CR] code is the last code on the output side, specify the following:
SENREV A$(*); CHR$ &0D, A$, X, CHR$ & 0A.

● **Function 2:** Outputs character data and then inputs data into an integer-type numeric variable.

Format: SENREV 《C》, A%, X, D 〔,|; E〕

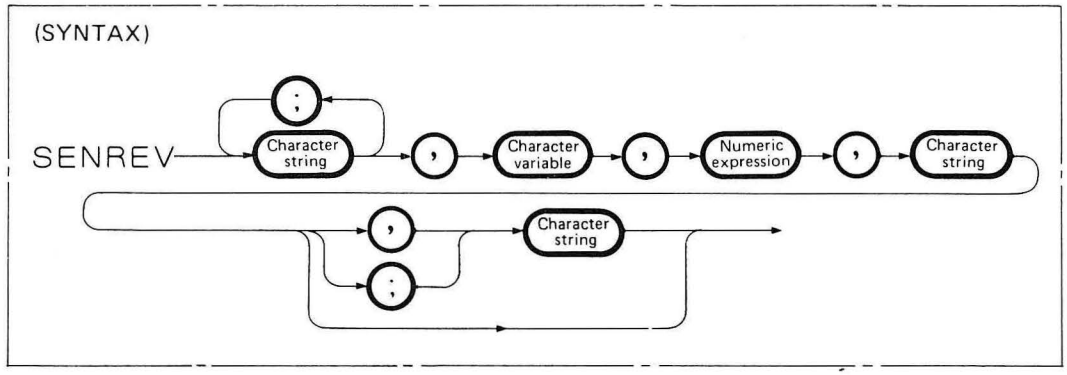
C : Character data

A%: Character variable for inputting data

X : Waiting time (0 – 255)

D : End code 1

E : End code 2



Explanation:

1. Outputs character data indicated in character string C and then immediately inputs 1-byte numeric data (binary data) into integer-type numeric variable A% up to the end code indicated by character strings D and E.
2. For the other details, see Function 1 of SEND instruction, Function 2 of RCV instruction, and Function 1 of this instruction.

Simple example:

```
10 SENREV "ABC",C%(*),100,CHR$ &0D
```

..... Outputs "ABC" and then immediately inputs 1-byte numeric data (binary data) until the end code [CR] (&0D) into collective integer-type numeric array variable C%(*). (The waiting time is 10 seconds for each element.)

OPCHNL

Opens the input buffer memory (1 byte) for the specified channel.

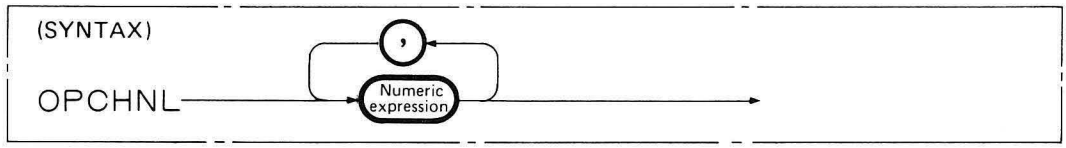
〈Purpose of use〉

For selecting the channel requested by peripheral devices and declaring its use by using this statement in conjunction with a POLLING instruction.

● **Function 1:** Opens the buffer memory for the specified channel.

Format: OPCHNL (X)

X: Channel number



Explanation:

Opens the input buffer memory for the channel indicated in the expression X value and clears the contents. In this case, the other channel statuses are not changed.

This statement is valid for use with a POLLING statement (explained next), it is not related to SEND, RCV, and SENREV statements.

An invididual buffer memory has a size of 1-byte size. If two or more bytes are input, only the last one byte remains in the buffer memory.

The CS signal (pin 5) for the channel opened with this statement goes to ON (high level).

Simple example:

```
10 OPCHNL 0,1,2..... Opens the buffer memories for channel numbers 0, 1,
                     and 2.
```

● **Function 2:** Closes all input buffer memories opened.

Format: OPCHNL



Explanation:

Closes all channels opened.

Note

1. For the 7-bit code data system, the channel opened with this instruction enters the shift-in (SI) status (for input statement).
2. A buffer memory opened is closed also by a SEND, RCV, or SENREV statement.

POLLING

Retrieves the contents of the input buffer memory (one byte) for the channel opened by an OPCHNL statement.

〈Purpose of use〉

To retrieve the channel requested by the peripheral device and declaring its use by using this instruction in conjunction with an OPCHNL instruction.

Format: POLLING C, A

C: Comparison code

A: Numeric variable for assigning the channel number



Explanation:

1. Retrieves the contents of the buffer memory opened with an OPCHNL instruction. If the contents match the comparison code (character code) indicated in character string C, assigns the channel number to numeric variable A and, at the same time, declares the use of this channel.
   (See the CHANNEL statement (page 288).)
   Retrieval is performed sequentially from the channel having the smallest channel number. If a channel having data matching the comparison code is found, checking stops immediately. In the character string, only the first character is valid as the comparison code and the second and subsequent characters are ignored.
2. If there are no matching channels, -1 is assigned to numeric variable A. In this case, the channel for the input/output instructions is not changed.

Simple example:

```
100 POLLING CHR$ &05,A
```

If the buffer memory contents of channel number 1, for the buffer memories opened, match CHR$ &05 (comparison code), this assigns 1 to the numeric variable A and, at the same time, declares the use of channel 1.

Application example:

```
100 CHANNEL 0,300,"7E1"
110 CHANNEL 1,1200,"7E2"
120 CHANNEL 2,110,"8N1"
130 OPCHNL 0,1,2
140 "POL":POLLING CHR$ &05,A
150 IF A<0 THEN "POL"
160 SENREV "READY",D$,0,CHR$ &0D
  :
500 GO TO "POL"
```

In the above program example, three peripheral devices are connected to this system and data is input/output via the related channel by a request (ENQ code (CHR$ &05)) from a peripheral device.

Line number 150 is used for data retrieval, the channel having the ENQ code (CHR$ &05) set, and data is input/output via line number 160. In this program the channel, once set, is not used for the second and subsequent retrieval operations.

Note

The channel whose comparison code is matched is closed automatically.

< Detailed information >

For example, OPCHNL and POLLING instructions can be used in the following case: If a certain code is specified before transferring information between this system and several peripheral devices connected to this system, this system can retrieve the requesting peripheral device (channel) by recognizing this code output by the peripheral device to this system.

RLIST

Outputs a program list to a printer.

〈Purpose of use〉

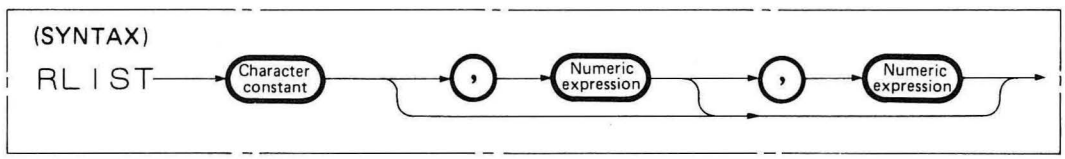
For outputting a program list to a printer with a RS-232C interface.

Format: RLIST S [, X [, Y]]

S : Title to be output

X: First line number

Y: Last line number



Explanation:

Outputs the program list starting with the line number indicated in numeric expression X and ending at the line number indicated in numeric expression Y, along with the outputs of the page number, month/day, hour : minute, and title indicated in character constant S to each page.

For details, see Function 3 of List command.

Simple example:

RLIST "CUBIC EQUATION",1000 [ENTER]

For example, if 20 October 14:30 is specified as the output date and time, the program on the left side is output as shown on the right side.

```
1000 !!EQUA
1010 SUB EQUA
     (X,DX,A(*))
1020*"INI":P=A(1)
1030 FOR I=2 TO 4
1040 Q=Q*X+P
1050 P=P*X+A(I)
1060 NEXT I
1070 DX=-P/Q
1080 DISP
     "DELTA X=";DX
1090 SUBEND
```

```
P001   10/20 14:30   CUBIC EQUATION
            EQUA

 1000 !!EQUA
 1010      SUB EQUA (X,DX,A(*))
 1020 "INI":P=A(1)
 1030      FOR I=2 TO 4
 1040      Q=Q*X+P
 1050      P=P*X+A(I)
 1060      NEXT I
 1070      DX=-P/Q
 1080      DISP "DELTA X=";DX
 1090      SUBEND

*USED BYTES 127*
```

Note

1. The codes of special patterns (CHR$ &80 to CHR$ &9F and CHR$ &E0 to CHR$ &FF) in the program are converted to underlines (CHR$ &5F) for output.
2. For small alphabetic characters in the program, the related codes are output directly. Therefore, a printer not having the character patterns related to these codes does not print these patterns even if these codes are output.
   Some printers may perform special operations.
3. Execution of this statement does not use the new page feed function. Page feed is performed with the CR code (&0D) and LF code (&0A).
4. The output list has maximum 80 columns.

TERM

Sets the data input/output format by using this system as a terminal.

〈Purpose of use〉
To use this system as a terminal.

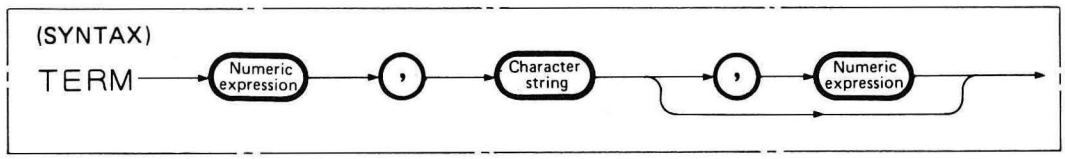
Format: TERM X, C [, Y]

X: baud rate (110, 200, 300, 600, 1200, 2400, 4800, or 9600)

C: Data format

Y: LF (line feed) specification

Expression Y value	Contents
0	The line is not fed when the CR code is entered.
1	The line is fed when the CR code is entered



Explanation:

1. Uses this system as a terminal (I/O unit) and sets the data input/output format as follows:
   Sets the baud rate (bits/second) using the numeric expression X value, sets the data format using the character string C contents, and sets the CR code processing in numeric expression Y.
   For the data format, see the CHANNEL statement (page 288).
2. When numeric expression Y is omitted, 0 is assumed to be set the value of numeric expression Y.

Simple example:

1. `TERM 1200,"7E1",1 ENTER` ····· Defines the system as a terminal and sets the baud rate to 1200 bits/second, the data format to 7-bit code data, the parity format to even parity, and the stop bit length to 1. The CR code has the line feed function.
2. `TERM 2400,"8N1" ENTER` ····· Defines the system as a terminal and sets the data input/output format to a baud rate of 2400 bits/second, data format to 8 bit code data with no parity, and a stop bit length of 1. The CR code has no line feed function.

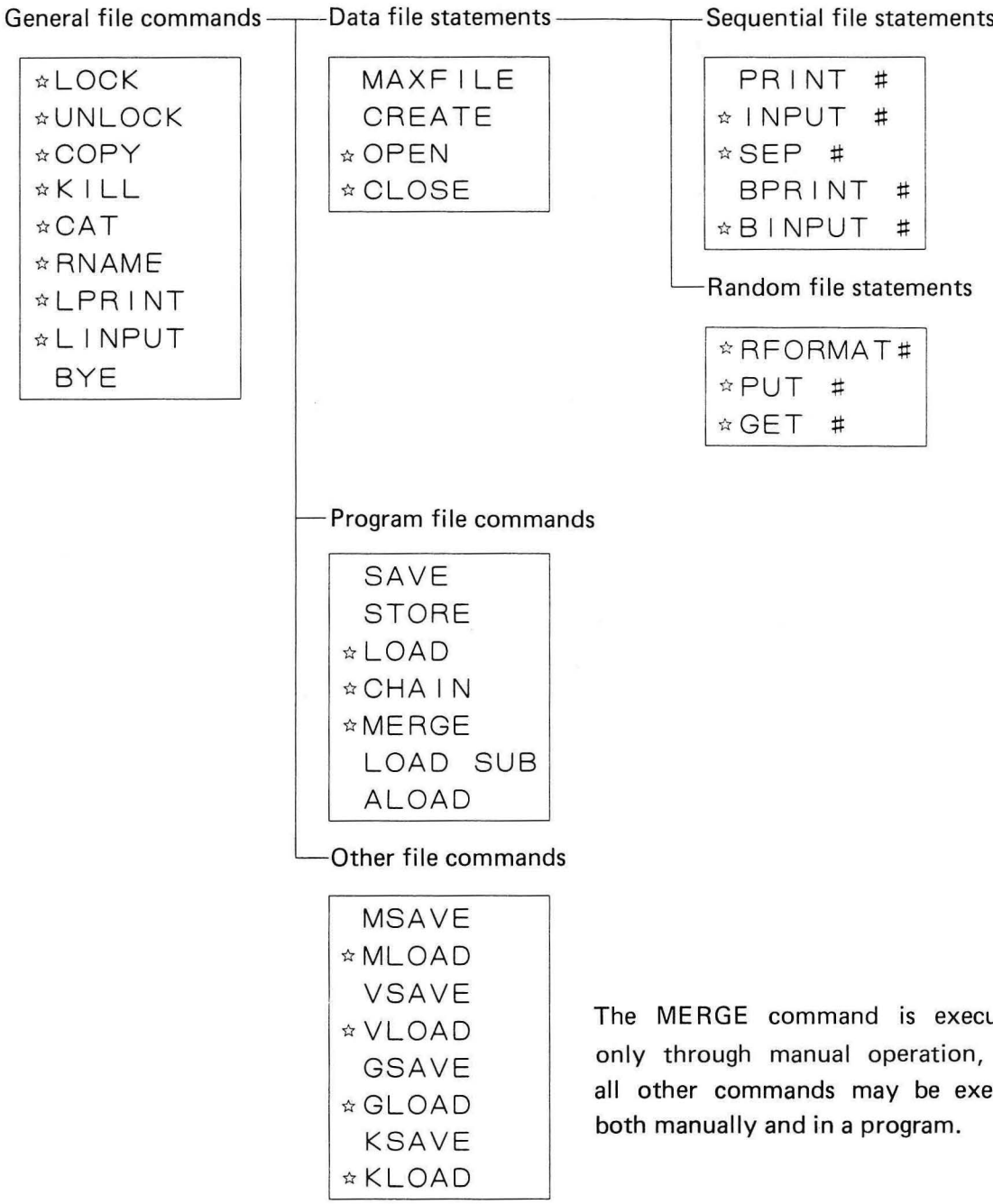
Note

1. The FDOS master disk must be placed in drive A0 during execution of this statement.
2. This statement is valid only for channel 0 (built into the unit).

Ⅳ. DISK BASIC INSTRUCTION WORDS

DISK BASIC is a BASIC language specifically designed to manipulate disk files. This section provides a detailed description of the DISK BASIC instructions concerned with disk files. All files are classified as data files, program files, or other files.
Likewise, the DISK BASIC instructions fall into three groups according to file type. In addition to these instructions, are those common to all files. The common instruction words provide means of controlling files to make efficient utilization of disk space.
Additional file information is available in 1.5 "Files" (page 35) in this manual.

1. INSTRUCTION LIST



The MERGE command is executable only through manual operation, while all other commands may be executed both manually and in a program.

The ☆-prefixed instructions are also applicable to files on disks generated by the Business Computer Model-3000 Series, and applicable only under the following conditions.

- ○ COPY ..... Only the contents of a disk initialized by the Business Computer Model-3000 Series can be copied onto a disk initialized by this unit.
- ○ OPEN ..... Only access specifications "I" and "R" are permissible.

By copying (page 50) or backing up (page 48) files created by the Model-3000 Series onto a disk initialized by this unit, all the instructions can be made usable.

2. GENERAL FILE MANIPULATION COMMANDS

This section defines the commands used to manipulate files:

LOCK
UNLOCK
COPY
KILL
CAT
RNAME
LPRINT
LINPUT
BYE

All these commands are available for use in a program.

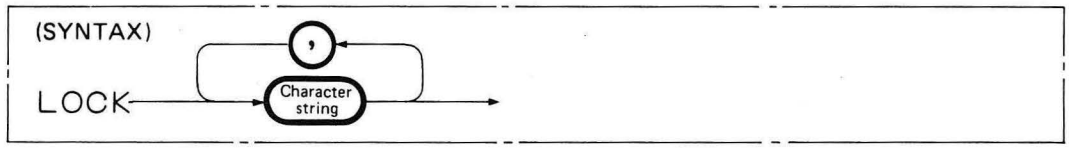
LOCK

Locks a designated file.

〈Purpose of use〉
To prevent writing to, or delection of, a file, or alteration of its file name.

Format: LOCK [C]

C: File specifications



Explanation:

1. Locks the specified file, thereby preventing writing to, or deletion of, final name alteration. More than one file may be specified at the same time.
2. Once a file is locked, an asterisk (*) is output to the right of the corresponding file name in the catalog list output by the CAT command (page 313) to indicate that the file has been locked. (See simple example 1.) The UNLOCK command is used to unlock a locked file.
3. It is recommended that system files and other important files are locked with this command.

Simple example:

1. `LOCK "BASIC/SYS"` [ENTER] ..... Locks the system file, named BASIC/SYS, on the disk in drive A0.

Catalog list sample output (1 copy)

```
BASIC/SYS     *    SYS   21
                ↑
              Asterisk
```

2. `LOCK "SAMPLE1","SAMPLE2"` [ENTER] ..... Locks the two files, named SAMPLE 1 and SAMPLE 2, on the disk in drive A0.

< Detailed information >

The alteration of the contents of a floppy disk can be entirely inhibited by covering the write protect notch in the upper right part of the disk with a write protect seal. (Refer to Owner's Manual, page 27). Thus, any attempted manipulations of the files on the disk, namely, alteration of the volume name, writing to or deletion of the files, locking and unlocking of the files, and alteration of the file names, are prevented.

UNLOCK

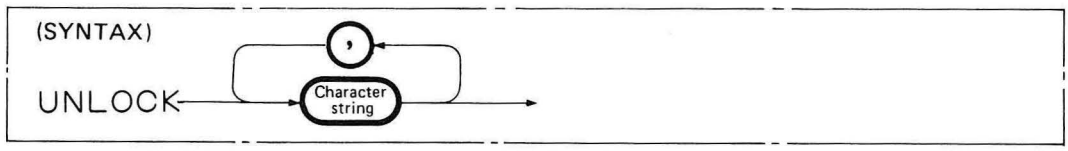
Unlocks designated file.

〈Purpose of use〉

To write data to a locked file, delete the file, or alter the file name.

Format: UNLOCK [C]

C: File specifications



Explanation:

Releases the named file from the locked state, thereby enabling data to be written to the file, the file to be deleted, or the file name to be altered. More than one file can be specified at the same time.

Simple example:

1. `UNLOCK "BASIC/SYS"` ENTER
   ……Unlocks the system file named BASIC/SYS.
2. `UNLOCK "SAMPLE1,SHARP","SAMPLE2:A1"` ENTER
   ……Unlock the two files with the file specifications SAMPLE 1, SHARP and one with SAMPLE 2: A1.

〈Detailed information〉

Once a file is unlocked, the asterisk attached to the right of the corresponding file name in the catalog list output by the CAT command (page 313) is eliminated.

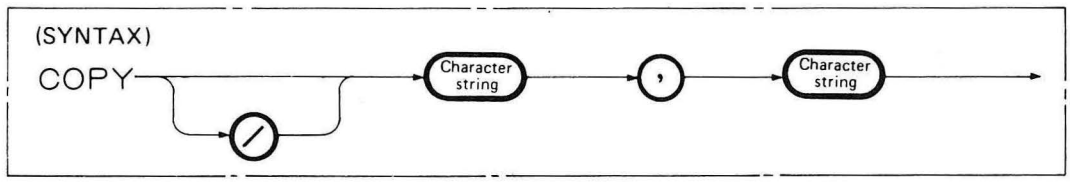
COPY

Copies a file.

〈Purpose of use〉
To copy the contents of a program file or data file for storage.

Format: COPY [/] C, D

/ : Variable hold specification
C: File name (source file)
D: File name (copy file)



Explanation:

Copies a file with file name C to create a file with file name D, without altering the contents of the source file. If no slash (/) is selected before file name C, all the variables in memory are erased.

Alternatively, if a slash is entered before file name C, all variables in memory are retained, in which case an error (ERROR 216) occurs unless the size of the user free area (excluding the data area) up until address &COOO is 84 bytes or more.

Simple example:

1. `COPY "ORIGINAL","COPY:A1"` [ENTER]
   ……Copies the file, named ORIGINAL, on the disk inserted in drive A0 onto the disk in drive A1 with the file name COPY.
2. `COPY "FIRST","SECOND"` [ENTER]
   ……Copies the file, named FIRST, on the disk inserted in drive A0 onto another disk in the same drive with the file name SECOND.

Note

1. It is not possible to create a copy file with the same file name as an existing file on the disk designated by file specifications D. Secure file cannot be copied too.
2. Execution of this command automatically deletes the buffer with buffer number 1 (page 323).
3. The memory map may differ according to the system organization. Further, the size (in bytes) of the user free area may also differ according to the type of variable. If ERROR-216 occurs, delete variables if possible.

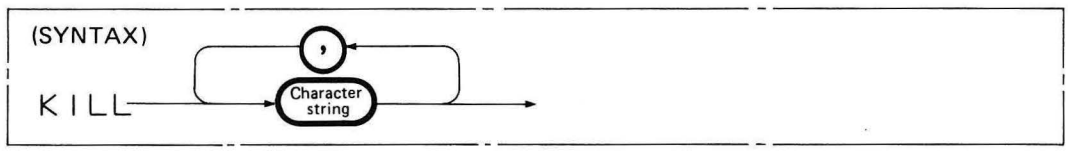
KILL

Deletes a designated file.

〈Purpose of use〉
To delete any file which is no longer required.

Format: K I L L 〔C〕

C: File specifications



Explanation:

Deletes the designated file and thus makes the area on the disk occupied by the file available to another file. More than one file may be specified at the same time.

Simple example:

```
KILL "SAMPLE.SHARP" [ENTER]
```

..... Deletes the file with file name SAMPLE on the disk inserted in drive A0.

Note

It is not possible to delete a file which has been protected (locked) by the LOCK command (page 308). Execution of the KILL command for a locked file will result in error (ERROR 226).

CAT

Outputs a catalog list of files stored on a disk.

〈Purpose of use〉
① To determine what files have been stored on a given disk.
② To identify the specifications of a given disk or the files on it.
③ To see the available area (remaining capacity) of a disk.

– Catalog list –

A catalog list contains information about the volume name of the disk, the description of the auto-load function (see page 363), the files names of the files stored, (the file extensions), whether the files are secure files or not, and the type and size (number of blocks) of each file. For random files, the number of records stored and the record length are also output in the catalog list.

It is also possible to output the creation date of each file as specified. After an entire catalog list is output, the number of unused blocks on the disk is output to indicate the size of the free disk area. Each block is 2K bytes in length.

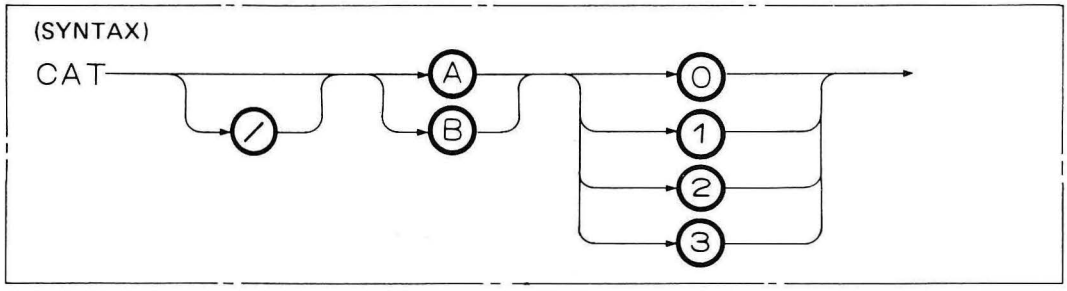
● **Function 1:** Outputs a catalog list on the CRT.

Format: CAT 〔/〕 **CH m**

/ : Date and time output specification

CH: Channel number (A or B)

m : Drive number (0, 1, 2, or 3)



Explanation:

1. Displays on the CRT a catalog list of the files on the disk in the drive with the specified channel drive number.
2. If a slash (/) is selected before the channel number, the date on which each file was written or the date on which it was last rewritten are also displayed in the month/day/hour format. This date corresponds to the information given by the TIME function which was automatically written during execution of the CLOSE, SAVE, STORE, MSAVE, VSAVE, KSAVE, or GSAVE instruction.

3. Locked files are designated by an asterisk (*) attached to the right of the file name. secure files are similarly designated by a character S. File types are represented by a three-character code as follows:

TYPE	Meaning
(Blank)	Data file
BAS	Intermediate-language format program file
TXT	ASCII format program file
OBJ	Machine language file
DSP	Video RAM file
GRA	Graphic video RAM file
KEY	Definable key file
SYS	System file

Simple example:

1. CAT A1 [ENTER] ..... Displays on the CRT a catalog list of the files on the disk inserted in channel drive A1.
2. CAT /A1 [ENTER] ..... In addition to the above display, displays the date on which each file was written or last rewritten.

Application example:

CAT /A0 [ENTER] ..... Command entry

CRT display

```
CAT   A0                                   V1.0A
VOLUME NAME-SHARP
ALOAD       -TEST/SR:A0,  100 <-- Description of auto-load function.
-------------------------------------------------
FILE NAME        TYPE SIZE |    DATE
                           |
SUBCPU/SYS    *   SYS   4  |                <-- The display in this area
DOS/SYS       *   SYS   8  |                    is suppressed if the slash
BASIC/SYS     *   SYS  21  |                    (/) is omitted.
DEMO1             OBJ   1  |  05.21.10
DEMO2          S  BAS   2  |  05.22.09
TEST/SR       *S  TXT   5  |  06.03.15
DATA.0                 18  |  06.11.14
  REC= 1100 LEN=   28      |
DATA.1        *        3  |  05.27.11
DATA.2                 8  |  06.16.13
K10-20            KEY   1  |  07.18.09
CRT.12            DSP   2  |  06.25.14
-------------------------------------------------
FREE     79
```

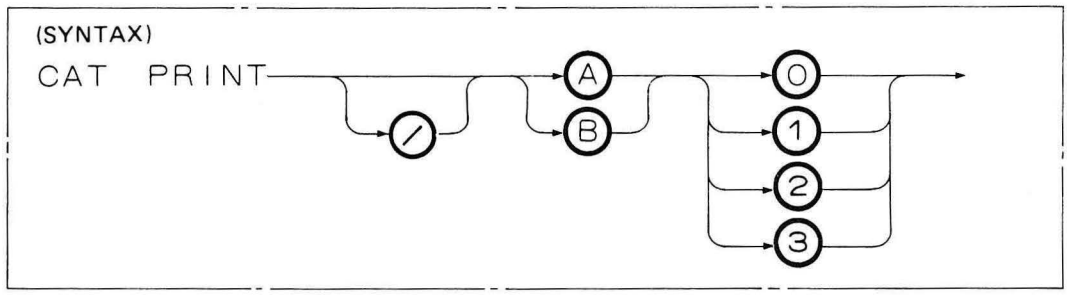
● **Function 2:** Outputs a catalog list to a printer.

Format: CAT PRINT [/] CH m

/ : Date and time output specification

CH: Channel number (A or B)

m : Drive number (0, 1, 2, or 3)



Explanation:

Outputs to a printer a catalog list of the files on the disk in the drive with the specified channel drive number. Similar to function 1 in all other details.

Simple example:

CAT PRINT /A1 [ENTER] ..... Outputs to a printer a catalog list of the files on the disk in channel drive A1, together with date and time.

Note

The listing operation may be interrupted by pressing either the HALT key or the space bar. Pressing the HALT key interrupts the execution of this command, as well as the listing operation itself, thereby passing control to the next command. If the listing operation has been interrupted with the space bar, the operation may be resumed by pressing the space bar again. If the [ENTER] key is pressed with the listing operation being interrupted with the space bar, the number of unused blocks is output before execution of this command is terminated and control is passed to the next command.

After the listing operation has been interrupted with the space bar, all keys other than the space and the [ENTER] key are invalid. During the listing operation all keys other than the HALT and the space bar are invalid.

RNAME

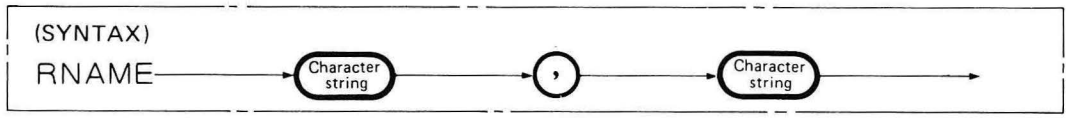
Changes a file name.

〈Purpose of use〉
To change the file name for easier recognition or reorganization.

Format: RNAME C, D

C: Current file name (file specifications)

D: New file name



Explanation:

1. Changes the file name of a file from C to D.
2. If a file name changes, the old file name is erased on disk, and the new file name is displayed in its place in the catalog list output by the CAT command (page 313).

Simple example:

1. RNAME "SAMPLE:A1","SAMPLE2" ENTER
   ..... Changes the file name of the file SAMPLE on the disk in drive A1 to SAMPLE2.
2. RNAME "FIRST","SECOND" ENTER
   ..... Changes the file name of the file FIRST on the disk in drive A0 to SECOND.

Note

1. Execution of this command for a locked file (page 308) results in error (ERROR 226).
   An error (ERROR 210) also occurs when the new file name specified already exists on the same disk.
2. The volume name and the channel drive number need not be specified in character string D. If specified, they will be ignored.
   The volume name and the channel drive number specified in file specification C are effective.

LPRINT

Changes a volume name.

〈Purpose of use〉
To change the volume name which is given at initialization (page 45).

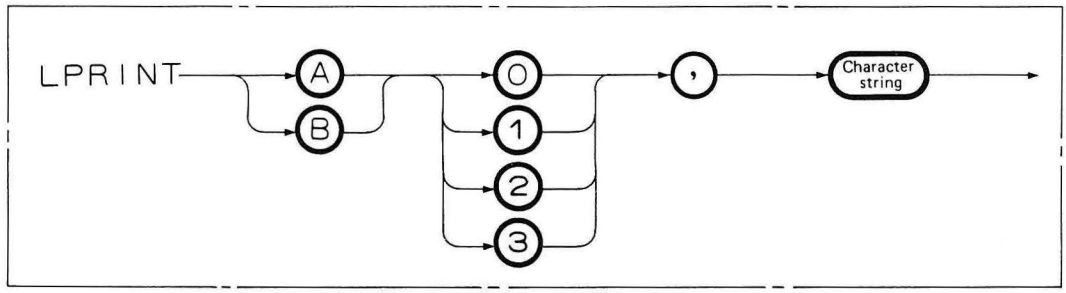
● **Function 1:** Specifies the channel and the drive number with constants.

Format:LPRINT CH m, C

CH: Channel number (A or B)

m : Drive number (0, 1, 2, or 3)

C : New volume name



Explanation:

Changes the volume name of the disk with a specified channel number and drive number to the new volume name C. The new volume name consists of six or fewer uppercase letters, and/or digits, exclusive of spaces.

Simple example:

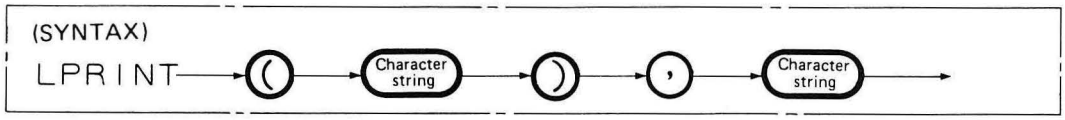
LPRINT A0,"AUGUST" ENTER .....Changes the volume name of the disk in channel drive A0 to AUGUST.

● **Function 2:** Specifies the channel drive number with a character string.

Format:LPRINT (D), C

D: Channel drive number (character string)

C: New volume name



Explanation:

Specifies the channel drive number with character string D. Similar to function 1 in all other details.

Simple example:

LPRINT (C$),"SAMPLE" ENTER ..... Changes the volume name of the disk in the channel drive with the channel drive number given by the contents of character variable C$, SAMPLE.

< Detailed information >

Changes in the volume names effected by this command can be ascertained by looking at the catalog list output by the CAT command (page 313).

LINPUT

Assigns a volume name to specified variable.

〈Purpose of use〉

To determine if the proper disk has been mounted in a specified drive.

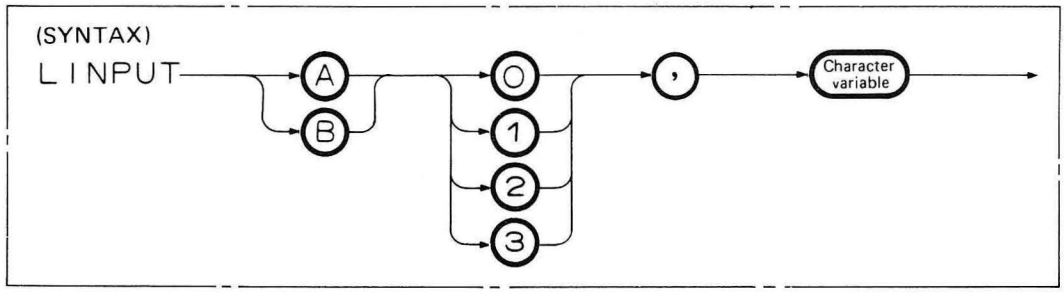
● **Function 1:** Specifies the channel and the drive number with constants.

Format:LINPUT CH m, A$

CH: Channel code (A or B)

m : Drive number (0, 1, 2, or 3)

A$: Character variable to which the volume name is assigned



Explanation:

Assigns the volume name of the disk with the specified channel and the drive number to character variable A$.

Simple example:

```
10 LINPUT A1,X$
```
..... Assigns the volume name of the disk with the channel drive number A1 to character variable X$.

Application example: Check the mounted disk in the drive by the volume name.

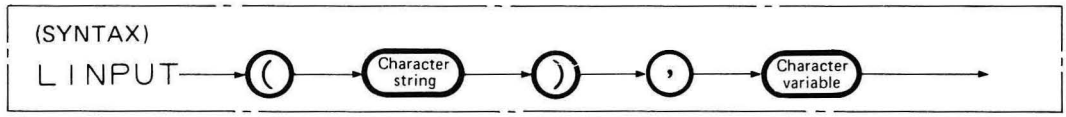
```
10 LINPUT A0,X$
```
..... Assigns the volume name of the disk with the channel drive number A0 to character variable X$.

```
20 IF X$<>"SHARP" DISP "DISK ERROR!":END
```
..... Displays DISK ERROR! on the CRT and ends program execution if the volume name assigned to X$ is not SHARP.

● **Function 2:** Specifies the channel code drive number with a character string.
Format: L I NPUT (C), A$

C : Channel code drive number (character string)

A$: Character variable to which the volume name is assigned



Explanation:

Specifies the channel code drive number with character C. Similar to function 1 in all other details.

Simple example:

```
20 LINPUT (C$),X$
```
..... Assigns the volume name of the disk with the channel drive number defined by the contents of character variable C$ to character variable X$.

Application example: Check the disk mounted in the drive by the volume name.

```
100 READ B$
110 LINPUT (B$),X$
120 IF B$="A1" THEN 150
130 IF X$<>"VOL1" DISP "DISK ERROR!":END
140 GO TO 100
150 IF X$<>"VOL2" DISP "DISK ERROR!":END
160 DATA "A0","A1"
```

Assigns the volume name of the disk with the channel code drive number specified by character string B$ to character variable X$.

Assigns channel code drive numbers via the DATA statement on line 160, or ends program execution if the volume names A0 and A1 are not VOL1 and VOL2, respectively.

BYE

Passes control from DISK BASIC to FDOS.

〈Purpose of use〉

To execute FDOS operations (page 40).

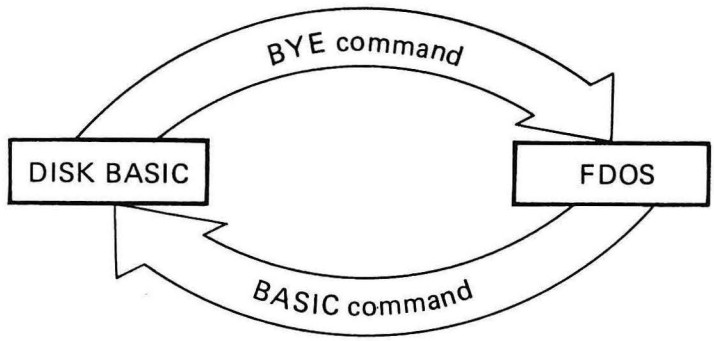
Format: BYE

(SYNTAX)

BYE

Explanation:

The BYE command passes control from BASIC to FDOS. Its purpose is to enable a FDOS operation (FDOS command) to be executed. To return control from FDOS to DISK BASIC, the BASIC command (page 44) is used. (The BASIC command is included in FDOS, and is used to enable DISK BASIC execution.) The relationship between the BYE command and the BASIC command is as illustrated below.



Note

Execution of this command will close all data files (page 329).

3. DATA FILE MANIPULATION STATEMENTS

This section explains the statements which are applicable to data files in general:

MAXFILE
CREATE
OPEN
CLOSE

All these statements are also executable in manual operation.

MAXFILE

Sets the number of buffers, or assigns the set number to buffers to a variable.

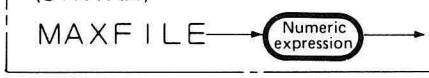
〈Purpose of use〉
① To handle data files.
② To check the number of buffers that has been set.

● **Function 1:** Sets the number of buffers.

Format:MAXFILE X

X : Number of buffers (1 to 7)

(SYNTAX)



Explanation:

1. Sets the number of buffers equivalent to the value of numeric expression X. This setting also limits the maximum number of files that can be handled concurrently to the value of numeric expression X. Up to seven buffers may be set. Each buffer takes up 64 bytes.
2. All variables are erased when the number of buffers is set by this statement.
3. The number of buffers is automatically set to 3 at power ON.
4. The number of buffers, once set, remains effective until it is reset.

Simple example:

```
10 MAXFILE 4
```
..... Sets the number of buffers to 4 and erases all variables at the same time. This statement assigns 256 (64 x 4) bytes of memory.

Note

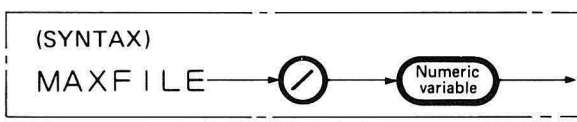
Since all the variables are erased when the number of buffers is set, this setting should be done in the initial part of the program.

● **Function 2:** Assigns the set number of buffers to a specified variable.

Format:MAXFILE /N

N : Numeric variable.

(SYNTAX)



Explanation:

Assigns the current number of buffers to numeric variable N. This means that the value of variable N indicates the current number of buffers set.

Variable N assigned 3 after power ON before setting the number of buffers. (See function 1).

Simple example:

```
10 MAXFILE #A ..... Assigns the current number of buffers to numeric
                    variable A.
```

CREATE

Reserves a recording area for a data file on a disk.

〈Purpose of use〉

To create a data file (in particular a random file).

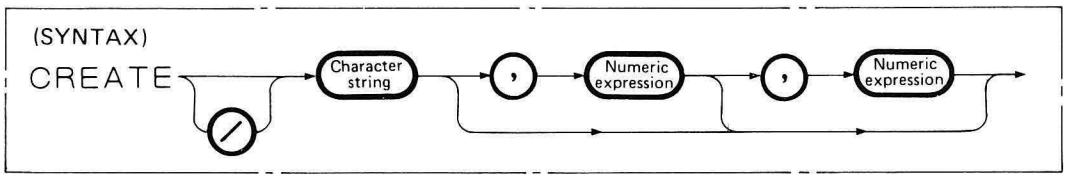
Format: CREATE [/] C [, X [, Y]]

/ : Area clearing specification

C : File specifications

X : Number of records (1 to 32767)

Y : Record length (2 to 65535)



Explanation:

1. Reserves a recording area equivalent in size to the number of records X times record length Y (bytes) on a disk for the data file with file specifications C. The number of records is in the range 1 to 32767, and the record length is in the range 2 to 65535 (bytes).
2. If record length Y is not specified, it is automatically set to 256 bytes.
3. If both the number of records X and record length Y are omitted, they are automatically set to 8 and 256 bytes, respectively, so that a recording area covering a single block (8 x 256 byes = 2K bytes) is reserved.
4. When a slash (/) is written preceding the file specifications, a null code alone is written in the reserved area. (The area is cleared.)

Simple example:

1. `20 CREATE "SAMPLE:A0",100,20`
   ..... Reserves a recording area equivalent in size to 100 records times a record length of 20 bytes, or covering 2000 bytes, on the disk for the data file with file specifications SAMPLE : A0. (Actually, a one-block area is reserved because 2000 ≦ 2048 (= 2K).)
2. `20 CREATE /"SAMPLE:A0",50`
   ..... Reserves a recording area equivalent in size to 50 records times a record length of 256 bytes on the disk for the data file with file specification SAMPLE : A0, and clears that area. (Actually, a two-block area is reserved because 3840 ≦ 4096 (= 4K).)
3. `20 CREATE /"SAMPLE:A0"`
   ..... Reserves a recording area equivalent in size to eight records times a record length of 256 bytes on the disk, and clears that area.

Note

1. A recording area must be reserved for a random file with this statement whenever the file is initially created.
2. This statement cannot be executed more than once with identical file names on the same disk.

< Detailed information >

In creating a sequential file, there is no need to reserve a recording area for it with this statement, yet this statement may be used to ascertain the availability of the required recording area on the disk.

OPEN

Declares the start of use of a data file.

〈Purpose of use〉

To allow access a data file for reading or writing.

Format: OPEN C, X, D

C : Access specification

"I" . . . . . Sequential file read (input)

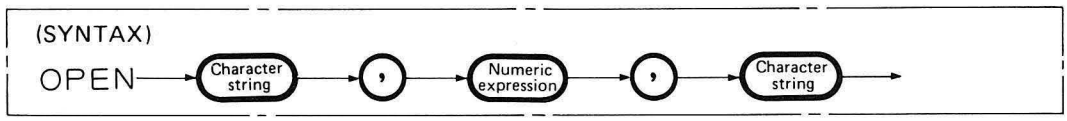
"O" . . . . . Sequential file write (output)

"A" . . . . . Sequential file update (appending)

"R" . . . . . Random file read/write (input/output)

X : Buffer number

D : File specifications



Explanation:

Declares the start of use of the data file with file specifications D via buffer X according to access specification C. The buffer number is assigned in the range of the number of buffers as set by the MAXFILE statement (page 323). The CLOSE statement, which is discussed next, is used to declare the end of this statement.

Simple example:

```
1. 10 OPEN "I",1,"ABCD:A1"
```

..... Declares the start of use of the data file with file specification ABCD : A1 for sequential file reading specification, and assigns buffer number 1 at the same time. Writing to the file in this condition is disabled.

```
2. 10 OPEN "A",2,"XYZ,SHARP:A0"
```

..... Declares the start of use of the data file with file specifications XYZ, SHARP : A0 for appending to a sequential file, and assigns buffer 2 at the same time. Execution of this statement enables appending to the sequential file with specifications XYZ, SHARP : A0.

Note

1. When the use of a data file opened by this statement is finished, the end of its use must be declared by using the CLOSE statement (page 329).
2. Before declaring the start of use of a random file, a recording area must be assigned to the file using the CREATE statement (page 325).
3. Execution of a statement in conflict with the access specifications results in error (ERROR 206), e. g., the PRINT # statement (page 335) is executen for a file with access specification " I ".
4. When a buffer has previously been assigned to a file opened by this statement, the buffer cannot be reassigned to another file with this statement unless the end of use of the file is declared; otherwise, an error (ERROR 202) occurs.
   Once the start of use of a data file is declared, the same file can be opened again by assigning another buffer to it (when the sequential file is specified for read).
5. Data files, once opened, cannot be deleted (page 312) unit properly closed.

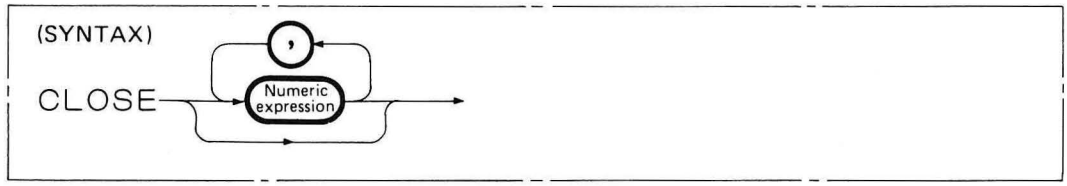
CLOSE

Declares the end of use of a data file.

〈Purpose of use〉
To close a file after it has been accessed for reading or writing.

Format: CLOSE [[X]]

X : Buffer number



Explanation:

1. Declares the end of use of the file to which buffer X has been assigned, and frees the buffer. More than one buffer may be handled at the same time.
2. If buffer number X is not specified, the end of use of all open files is declared.

Simple example:

1. `100 CLOSE 2,3` ..... Declares the end of use of the data files to which buffer Nos. 2 and 3 have been assigned, and free these buffers.
2. `200 CLOSE` ..... Declares the end of use of all open data files and frees all the buffers assigned to them.

Note

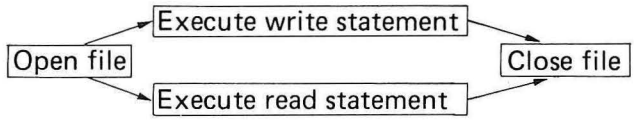
1. In cases where the disk contains any data file which has been opened with the sequential file write specification or random file read/write specification , the disk must not be taken out of the drive or the unit powered off until the end of use of the file is declared. Otherwise, residual data, if any, in the buffer assigned to the file might not be written completely to the file.
2. Execution of this statement overrides the function of theSEP # statement (page 340) and the RFORMAT # statement (page 343).
3. This statement cannot be used to free buffers which have not been assigned by the OPEN statement.
4. Execution of the LOAD command (page 352), the RUN command (page 78), a variation of the CHAIN command (page 354), the MERGE command (page 356), and the NEW command used in machine language causes a forced declaration of the end of use of all data files.

3-1. SEQUENTIAL FILE MANIPULATION STATEMENTS

This section describes the statements which are directly used to read or write sequential files:

BPRINT #
BINPUT #
PRINT #
INPUT #
SEP #

All these statements are also executable in manual operation. The sequence of procedural steps to read or write sequential files is:



A simple program example is shown next; follow the overall flow and understand each statement then go over again to make sure you understand it.

i) Writing to a sequential file

```
 10 MAXFILE 1 ..... Setting of number of buffers
 20 OPEN "O",1,"TEL:A1" ..... Opening of file
 30 INPUT "NUMBER OF PERSONS?";N
 40 PRINT #1,N; ..... Execution of write statement
 50 FOR I=1 TO N
 60 INPUT "NAME?  ";N$
 70 INPUT "TEL?   ";T$
 80 PRINT #1,N$;CHR$ &0D;T$ ..... Execution of write statement
 90 NEXT
100 CLOSE 1 ..... Closing of file
110 END
```

In this example, all data is written in ASCII code format.
The BINPUT # statement (page 334) is used to write binary data format.

ii) Reading from a sequential file

```
 10 MAXFILE 1 ..... Setting of number of buffers
 20 OPEN "I",1,"TEL:A1" ..... Opening of file
 30 INPUT #1,N ..... Execution of write statement
 40 DISP :DISP "NUMBER OF PERSONS";N
 50 DISP "NAME","TEL"
 60 FOR I=1 TO N
 70 INPUT #1,N$,T$ ..... Execution of read statement
 80 DISP N$,T$
 90 NEXT
100 CLOSE 1 ..... Closing of file
110 END
```

In this example, all data to be read must have been stored in ASCII code format. The BINPUT # statement (page 334) is used to read binary data format.

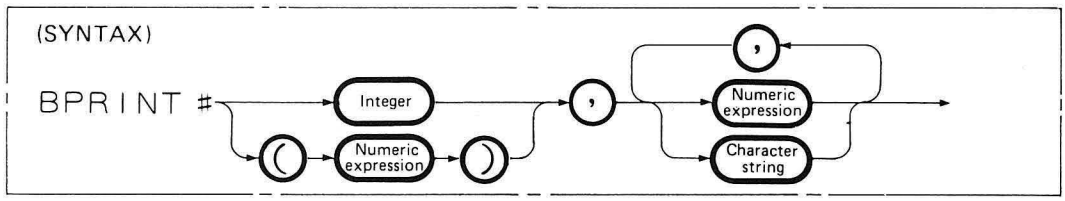
BPRINT

Writes data to a sequential file in binary code format.

〈Purpose of use〉
① To create a sequential file in binary code format.
② To rewrite data to a sequential file in binary code format.

Format: BPRINT # n | (X), {Y | C}

n : Buffer number
X : Buffer number (specified by the value of numeric expression X)
Y : Numeric data
C : Character data



Explanation:

1. Writes the data represented by numeric expression Y or character string C via the buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions truncated) as a sequential file in binary code format onto disk.
2. If the sequential file write specification ("O") has been selected in the OPEN statement (page 327), data is written in the file from the beginning of the data file onwards. If the append specification ("A") has been selected in the OPEN statement, data is added to the existing data file.
3. When data is in binary code format, constants or variables representing the data require the sizes specified in the table below.

Data	Required size for writing
Integer type numeric constants and variables (e.g., 3%, A%)	2 bytes
Real type numeric constants and variables (e.g., 4.5, X)	8 bytes
Character constants (e.g., "ABC")	(Character length + 2) bytes
$-type character simple variables (e.g., M$)	18 bytes
@-type character simple variables (e.g., E@)	(Character length specified by DIM statement (or 16) + 4) bytes
$-type character array variables (e.g., A$ (*))	(Character length specified by DIM statement x Number of elements + 2) bytes
@-type character simple variables (e.g., G@ (*))	((Character length specified by DIM statement + 2) x Number of elements +2) bytes
Numeric expressions (e.g., 5 + C1)	2 or 8 bytes
Character strings (e.g., P$ + Q$)	(Character length + 2) bytes

Simple example:

```
100 BPRINT #1,A$,B
```
..... Writes the contents of character variable A$ and numeric variable B via buffer with buffer number 1 onto a disk in this order. (The size of the recording area required for such writing is (18 + 8) bytes.)

Note

1. The start of use of a sequential file must be declared before data is written to the file (page 327).
2. No provision for separators is required in writing data in binary code.

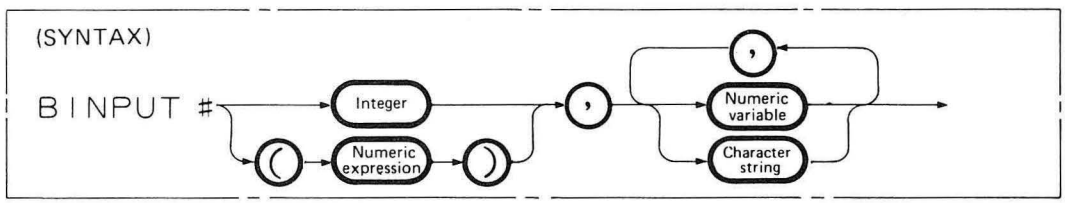
BINPUT

Reads sequential file data in binary format.

〈Purpose of use〉
① To use sequential file data from a sequential file in binary code format in a program.
② To alter data in a sequential file written in binary code format.

Format: BINPUT # n | (X), {N | A$}

n : Buffer number
X : Buffer number (numeric expression)
N : Numeric variable to which data is assigned
A$: Character variable to which data is assigned



Explanation:

Reads data from a sequential file via the buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions truncated) and assigns the data to numeric variable N or character variable A$. The file from which data is to be read must be in binary code.

Simple example:

```
100 BINPUT #2,X$,Y
```

Reads data from disk via buffer with buffer number 2, and assigns it to character variable X$ and numeric variable Y.

Note

1. The start of use of a sequential file must be initially declared before data is read from the file (page 327).
2. Data written in the form of an generalized array variable (e.g., AP (*)) is read into a similar variable.

PRINT

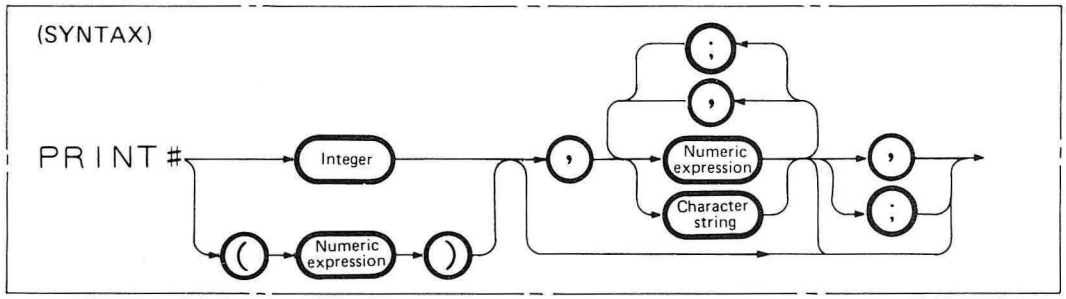
Writes data to a sequential file in ASCII format.

〈Purpose of use〉
① To create a sequential file in ASCII code.
② To rewrite data to a sequential file in ASCII code.

● **Function 1:** Writes data to a sequential file in ASCII format.

Format: PRINT # n | (X) [, {{Y | C}} [, | ;]]

n : Buffer number
X : Buffer number (numeric expression)
Y : Numeric data
C : Character data



Explanation:

1. Writes the data represented by numeric expression Y or character string C via the buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions truncated) as a sequential file on disk in ASCII code in the format specified by the PRINT statement (page 110).
2. If the sequential file write specification (O) has been selected in the OPEN statement (page 327), data is written in the file starting from the beginning of the data file afterwards.
   If the appendage specification (A) has been selected in the OPEN statement, data is added to the existing data file.

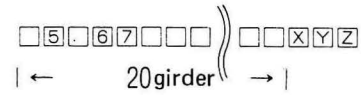
Simple example:

1. `20 PRINT #1,12.34;"ABC"` ..... Data is written onto disk via buffer with buffer number 1 in the following format:

*1) *2)
□1 2 . 3 4 □ A B C CR

*1) □ : Denotes a space character.
*2) CR : Denotes the carriage return code (CD) (&0D).

2. `20 PRINT #(2+3),5.67,"XYZ";` ····· Data is written onto disk via buffer with buffer number 5 in the following format:

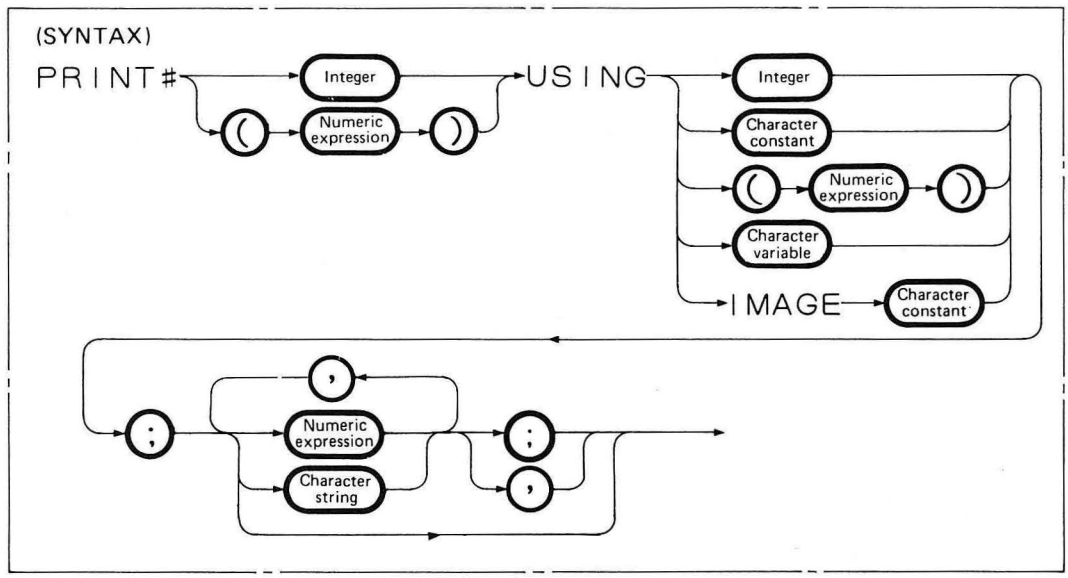


(Note the functions of the comma (,) and the semicolon (;))

● **Function 2:** Writes data with the aid of the USING statement (page 108).

Format: **PRINT # n | (X) USING m | S | (Z) | A$ | IMAGE T ; { {Y | C} {; | , }}**

n : Buffer number
X : Buffer number (numeric expression)
m : Line number
S : Line label name
Z : Line number (numeric expression)
A$: Image symbol (page 113)
T : Image symbol
Y : Numeric data
C : Character data



Explanation:

Writes the contents of numeric expression Y or character string C via the buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions truncated) onto disk in the format represented by the image symbol. The write format conforms to the PRINT statement as mentioned in Function 2. See the description of the IMAGE statement (page 113).

Simple example:

1. `20 PRINT #1 USING 300:Q$` ..... The contents of character variable Q$ are written onto disk via buffer with buffer number 1 in the format represented by the image symbol on line number 300.

2. `20 PRINT #(3+4) USING P$:Q$,K` ..... The contents of character variable Q$ and numeric variable K are written onto disk via buffer with buffer number 7 in the format represented by the image symbol specified by character variable P$.

Note

1. The start of use of a sequential file must be declared before data can be written to the file (page 327).
2. Because the writing with this statement is effected in ASCII code format, provision must be made for separators (delimiters) during reading. Detailed information on separators will appear later under the INPUT # statement(page 338).

Application example:

[Program A]

```
 10 MAXFILE 1
 20 OPEN "O",1,"AZ:A0"
 30 A$="ABC"
 40 PRINT #1 USING 500:A$
500 IMAGE "5A"
600 END
```

Data is written via buffer with buffer number 1 as a file with file specifications "AZ : A0" on disk in the following format:

A B C □ □ CR

[Program B]

```
 10 MAXFILE 2
 20 OPEN "O",2,"AZ:A1"
 30 A$="5A 2X $#,###"
 40 B$="ABC":K=1320
 50 PRINT #2 USING A$;B$,K
100 END
```

Data is written via buffer with buffer number 2 as a file with file specifications "AZ : A1" on disk in the following format:

A B C □ □ □ □ $ 1 . 3 2 0 CR

Sample programs are also available on page 329.

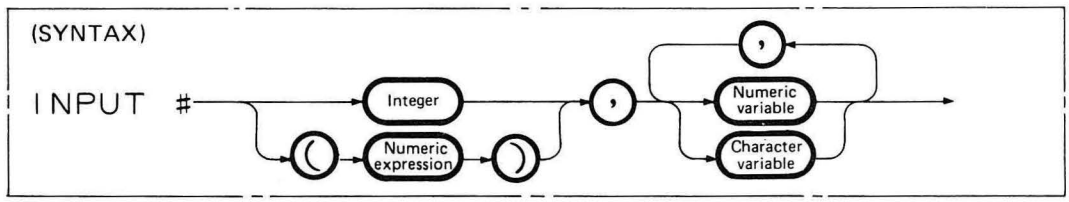
INPUT

Reads data from a sequential file in ASCII format.

〈Purpose of use〉
① To use, in a program, sequential file data in ASCII code.
② To after data in a sequential file in ASCII code.

Format: INPUT # n | (X), {N | A$}

n : Buffer number
X : Buffer number (numeric expression)
N : Numeric variable to which data is assigned
A$: Character variable to which data is assigned



Explanation:

1. Reads data from a sequential file via buffer with the buffer number specified by integer n or numeric expression x (with decimal fractions truncated) and assigns the data to numeic variable N or character varibale A$. The sequential file form which data is to be read must be in ASCII code.
2. A separation between two data fields is assumed whenever CR LF *1), or EOD *2) is encountered while data is being read with this statement. These are called separators (delimiters). In the case of numeric constants, the space character is also regarded as a separator.

   In writing the PRINT # statement (page 335), therefore, care should be exercised to ensure the proper use of separators.

Simple example:

```
10 INPUT #1,A$,B
```

When the following format of data is read via buffer with buffer number 1, SHARP and −78.9 are assigned to character variable A$ and numeric variable B respectively.

SHARP CR −78.□9□ CR

↑ Separator ↑ Separator (space)

*1) LF : Indicates LF (carriage return and line feed) code (&0A)
*2) EOD : Indicates the end of data
An error (ERROR 209) occurs when EOD is read. To use EOD as a separator, release the error by the ON ERROR statement and test if ERN is 209.

Note

1. The start of use of a sequential file must be initially declared before data is read from the file (page 326).
2. Normally [CR] , [LF] and EOD are used as separators for character constants, and [ ], [CR], [LF], and EOD are used as separators for numeric constants. Separators for character constants may also be set through the execution of the SEP# statement (page 340).

   Application programs are available on page 330).

SEP

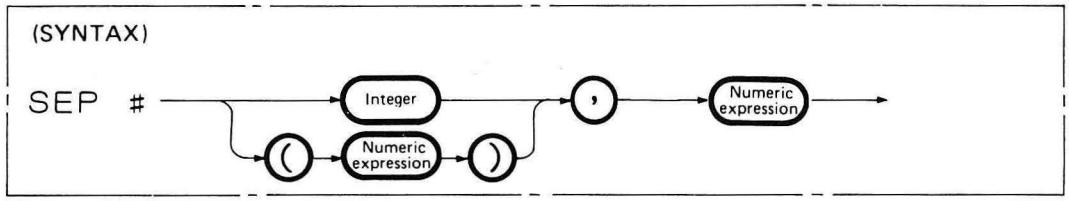
Sets separators for reading a sequential file in ASCII format.

Format: SEP # n | (X), Y

n : Buffer number

X : Buffer number (numeric expression)

Y : ASCII code character



Explanation:

1. Sets the separators to the ASCII code character represented by numeric expression Y and EOD, for reading character constants in the sequential file for reading character constrants in the sequential file to which the buffer with buffer number specified by integer n or numeric expression X (decimal fraction truncated) is assigned. For ASCII code, refer to ASCII Code table in Owner's Manual, Appendix 1.
2. A list of the separators is given below.

Data	Separators	
Numeric constants	□, [CR], [LF], and EOD (regardless of SEP statement)	
Character constants	If not specified by SEP statement	[CR], [LF], EOD
	If specified by SEP statement	Specified charactor, EOD

Simple example:

1. `10 SEP #1,&53` ..... Sets the separator for reading character data from the sequential file to which the buffer with buffer number 1 has been assigned, as character S in ASCII code 53 (hexadecimal) and EOD.
2. `10 SEP #1,83` ..... Sets the separator for reading character data from the sequential file to which the buffer with buffer number 1 has been assigned, as character S in ASCII code 83 (decimal) and EOD.

Note

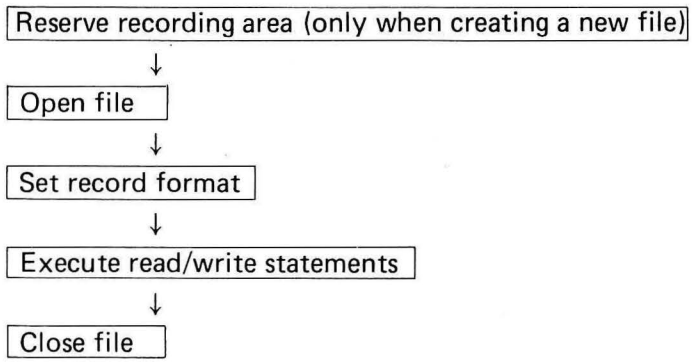
A label name cannot be attached to this statement. Further, a multi-statement cannot be set before this statement.

3-2. RANDOM FILE MANIPULATION STATEMENTS

This section describes the statements which are used to read or write random files:

RFORMAT #
PUT #
GET #

All these statements are also executable in manual operation. The sequence of steps to follow to read random files is:



The reservation of a recording area to a file on disk is required only when the file is created by the CREATE statement (page 325). It is not required for existing files.
Sample programs to give the reader a basic understanding of this sequence are given below.

i) Writing to a random file

```
 10 MAXFILE 1 ..... Setting of number of buffers
 20 INPUT "HOW MANY ITEMS ? ";N
 30 CREATE "STOCK:A1",N,40 ..... Reserves recording area
 40 OPEN "R",1,"STOCK:A1" ..... Opens file
 50 RFORMAT #1,C%,H$,T,Z ..... Sets record format
 60 DISP CLEAR
 70 DISP "CODE NO.          :"
 80 DISP "ITEM NAME         :"
 90 DISP "UNIT PRICE        :"
100 DISP "STOCK QUANTITY    :"
110 CURSOR 17,0:INPUT N
120 IF N=0 THEN "CLOSE"
130 CURSOR 17,1:INPUT H$
140 CURSOR 17,2:INPUT T
150 CURSOR 17,3:INPUT Z
160 CURSOR 17,5:INPUT "OK (Y/N) ? ";A$
170 IF A$="Y" THEN "PUT"
180 IF A$="N" THEN 130:ELSE 160
190*"PUT":C%=N:PUT #1,C% ..... Executes write statement
200 GO TO 60
210*"CLOSE":CLOSE 1 ..... Closes file
220 DISP CLEAR :DISP "WRITE END":END
```

ii) Reading a random file

```
 10 MAXFILE 1 ..... Set number of buffers
 20 OPEN "R",1,"STOCK:A1" ..... Opens file
 30 RFORMAT #1,C%,H$,T,Z ..... Set record format
 40 DISP CLEAR
 50 DISP "CODE NO.          :"
 60 DISP "ITEM NAME         :"
 70 DISP "UNIT PRICE        :"
 80 DISP "STOCK QUANTITY    :"
 90 CURSOR 17,0:INPUT N
100 IF N=0 THEN "CLOSE"
110 GET #1,N ..... Execute write statement
120 CURSOR 17,1:DISP H$
130 CURSOR 17,2:DISP T
140 CURSOR 17,3:DISP Z
150 CURSOR 17,5:INPUT "NEXT (Y/N) ? ";A$
160 IF A$="Y" THEN 90
170 IF A$="N" THEN "CLOSE":ELSE 150
180*"CLOSE":CLOSE 1 ..... Closes file
190 CURSOR 0,6:END
```

iii) Writing to and reading a random file concurrently

```
 10 MAXFILE 1 ..... Set number of buffers
 20 DIM J$(1)*40
 30 OPEN "R",1,"LIST:A1" ..... Opens file
 40 RFORMAT #1,C%,N$,T$,J$(1) .... Reserves recording area
 50 DISP CLEAR
 60 DISP "CODE NO.:"
 70 DISP "NAME     :"
 80 DISP "TEL      :"
 90 DISP "ADDRESS  :"
100 CURSOR 10,0:INPUT N
110 IF N=0 THEN 260
120 GET #1,N ..... Executes read statement
130 IF C%=0 THEN 200
140 CURSOR 10,1:DISP N$
150 CURSOR 10,2:DISP T$
160 CURSOR 10,3:DISP J$(1)
170 CURSOR 10,5:INPUT "OK (Y/N) ? ";A$
180 IF A$="Y" THEN 240
190 IF A$="N":ELSE 170
200 CURSOR 10,1:INPUT N$
210 CURSOR 10,2:INPUT T$
220 CURSOR 10,3:INPUT J$(1)
230 GO TO 170
240 C%=N:PUT #1,C% ..... Executes write statement
250 GO TO 50
260 CLOSE 1 ..... Closes file
270 CURSOR 0,6:END
```

In this sample program, modification and addition to stored data is executed concurrently.

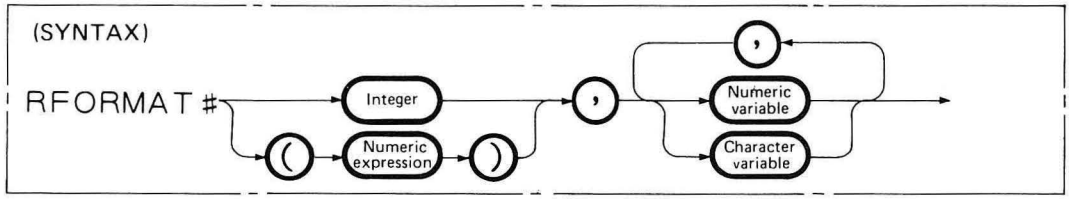
RFORMAT

Assigns variables for reading or writing a random file.

〈Purpose of use〉
To read or write a random file.

Format: RFORMAT # n | (X), {N | A$}

n : Buffer number
X : Buffer number (numeric expression)
N : Numeric variable
A$: Character variable



Explanation:

Assigns to numeric variable N or character variable A$, the contents of the data items to be read from or written to the random file to which the buffer with the buffer number specified by integer n or numeric expression X (with dicimal fractions truncated) has been allocated. In reading data from the random file, therefore, the data is assinged to numeric variable N or character variable A$ specified in this statement. In writing data to the random file, the contents of numeric variable N or character variable A$ are written.

Simple example:

1. `100 RFORMAT #3,C%,N$,A$(1),B` ..... Assigns to variables C%, N$, A$(1), and B, the contents of the data items in the random file to which the buffer with buffer number 3 has been allocated.
2. `200 RFORMAT #1,R(*)` ..... Assigns, to generalized array variable R(*) the contents of the data items in the random file to which the buffer with buffer number 1 has been allocated.

Note

1. This statement must be executed prior to the random file read/write statements, GET # (page 345) and PUT #.
2. A generalized array variable (e.g., X(*) and Y$(*)) is assigned to read/write the entire contents of an array variable.
3. A label name cannot be attached to this statement. Further, a multi-statement cannot be set before and after this statement.

Application example:

An application example is given on page 341.

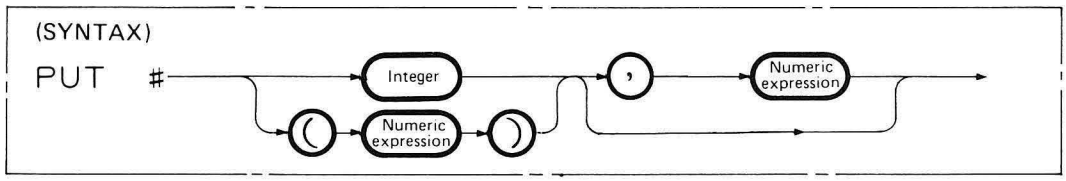
PUT

Assigns variables for writing a random file.

〈Purpose of use〉
① To create a random file.
② To modify and add data to a random file.

Format:PUT # n | (X) [,Y]

n : Buffer number
X : Buffer number (numeric expression)
Y : Record number



Explanation:

1. Writes data to record number Y in the random file whose variable contents are defined by the RFORMAT # statement (page 343) using the buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions turncated).
2. If Y is omitted, the record number is set to 1 immediately after the opening of the file. Thereafter, the record number is incremented by one every time this statement or the GET # statement (page 346) is executed.

Simple example:

1. `100 PUT #3,5` ..... Writes data to record number 5 in the format defined by the RFORMAT # statement via buffer with buffer numer 3.
2. Writes data to record numbers 5 and 6 with the contents of variable A%, B$, and C(1) using the buffer with buffer number 3 in this order.

```
    :
100 RFORMAT #3,A%,B$,C(1)
110 PUT #3,5
120 PUT #3
    :
```

Note

The assignment of a recording area (page 325) and the opening of a random file (page 327) are required before any data is written to the file.

Application example:

An application example is given on page 341.

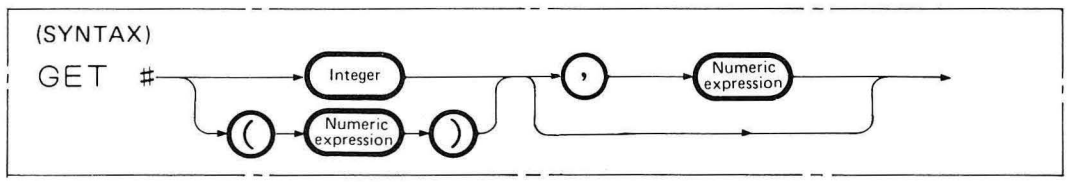
GET

Reads data from a random file in ASCII code.

〈Purpose of use〉
To input data from a random file.

Format: GET # n | (X) [,Y]

n : Buffer number
X : Buffer number (numeric expression)
Y : Record number



Explanation:

1. Reads data from the recording area with record number Y for a random file via buffer with the buffer number specified by integer n or numeric expression X (with decimal fractions truncated) and assigns the data to the variable defined by RFORMAT # statement (page 343).
2. If Y is not specified, the record number is set to 1 immediately after the opening of the file. Thereafter, the record number is incremented by one every time this statement or the PUT # statement (page 345) is executed.

Simple example:

1. `100 GET #3,5` ..... Reads data from the recording area with record number 5 using buffer number 3 and assigns it to the variable defined by the RFORMAT # statement via buffer 3.
2. Reads data from the recording area with record number 6 via buffer with buffer number 3 and assigns it to variables A%, B$, and C(1).

```
    ⋮
100 RFORMAT #3,A%,B$,C(1)
110 PUT #3,5
120 GET #3
    ⋮
```

Note

The assignment of a recording area to a random file (page 326) is required before any data is written to the file.

Application example:

An application example is given on page 341.

4. PROGRAM FILE MANIPULATION COMMANDS:

This section describes the statements which are used to read or write program files:

SAVE ※
STORE ※
LOAD ※
CHAIN ※
MERGE
LOAD SUB ※
ALOAD ※

The statements marked with ※ can also be used in manual operation.

SAVE

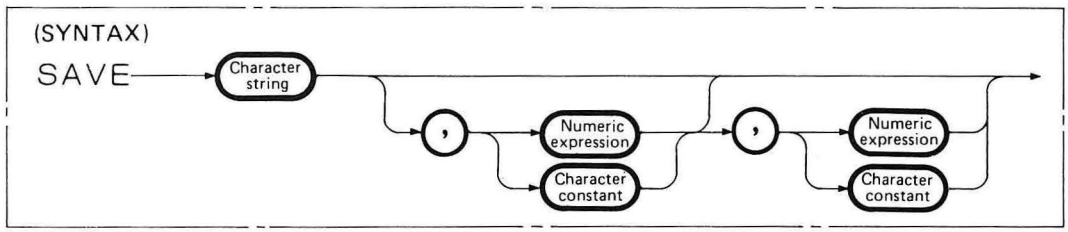
Writes a program onto disk in intermediate language format.

〈Purpose of use〉
To save a stored program in memory onto disk in intermediate language format

● **Function 1:** Writes a program as a file.

Format:SAVE C [, X | S [, Y | T]]

C : File specifications
X : Write beginning line number
S : Write beginning line label name
Y : Write ending line number
T : Write ending line label name



Explanation:

1. Writes that part of the program in memory between line number X or line label name S and line number Y or line label name T onto disk in intermediate language format with file specifications C.
2. If the ending line number or label name [, Y | T] is omitted, the contents of all the program following the line specified by either X or S is written onto disk.
3. If both the beginning line and the ending line specifications are omitted, the entire program in memory is written onto a disk.

Simple example:

1. SAVE "RADIO" ENTER
   ..... Writes the program in memory with the file name RADIO to the disk in drive A0, in intermediate language format.
2. SAVE "SAMPLE,SHARP" ENTER
   ..... Writes the program in memory with the file name SAMPLE to the disk which has volume name SAHRP, in intermediate language format.
3. SAVE "MEMORY:A1",100,200 ENTER
   ..... Writes that part of the program in memory, beginning with line 100, as far as line 200, with the file name MEMORY, onto disk, in intermediate language format.

● **Function 2:** Writes a program as a secure file.

Format: SAVE @ C [, X | S [, Y | T]] [, /D]

@ : Secure file indication

C : File specifications

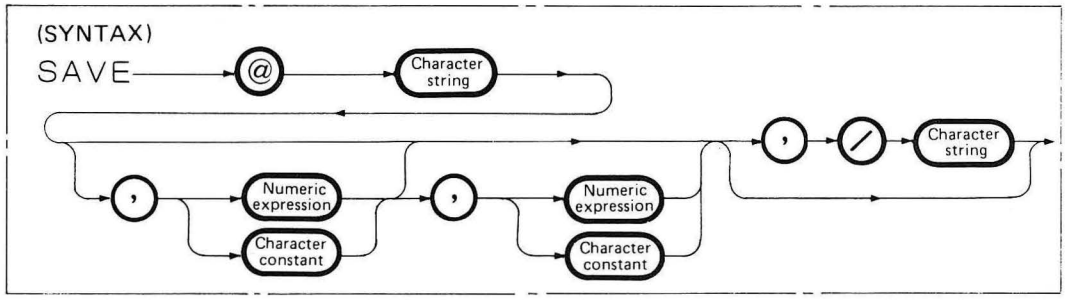
X : Write beginning line number

S : Write beginning line label name

Y : Write ending line number

T : Write ending line label name

D : Password (until six characters)



Explanation:

1. Use of @ before file specifications C causes the program to be written as a secure program file.
   Similar to function 1 in all other details.
2. A secure program file, once saved on disk, cannot be listed, modified, debugged, or written to. The PEEK function (page 29) and the USR command (page 284) cannot be used either. This secret state cannot be released.
3. By having a user-determined password specified in a secure program file, program listing only is possible.
   For listing instructions, see the description of the LIST command (page 85).

Simple example:

1. SAVE @"START:A0","STEP" ENTER
   ..... Writes that part of the program in memory beginning the line label name STEP onwards onto the disk in drive A0 as a secure program with the file name START.
2. SAVE @"TEST",100,/"KEY" ENTER
   ..... Writes that part of the program in memory beginning with the line number 100 onwards onto the disk in drive A0 as a secure program with the file name TEST. A password KEY is given at the same time. (Program listing is made possible by entering the password KEY.)

<Helpful hint>
The CAT command (page 313) may be used to determine if a particular file has been written onto disk as a secure file.

STORE

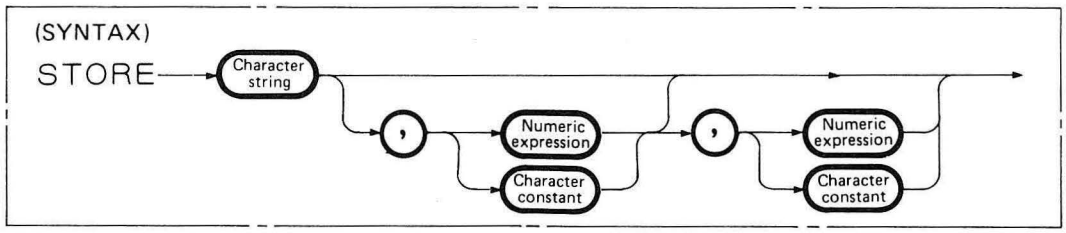
Writes a program onto disk in ASCII format.

〈Purpose of use〉
To store a program in memory onto disk in ASCII code.

● **Function 1:** Writes a program as a file.

Format:STORE C [, X | S [, Y | T]]

C : File specifications
X : Write beginning line number
S : Write beginning line label name
Y : Write ending line number
T : Write ending line label name



Explanation:

1. Writes that part of the program in memory, between line number X or line label name S and line number Y or line with label name T onto disk in ASCII code with file specifications C.
2. If the end line number or label name [, Y | T] is omitted, the contents of all the program following the line specified by either X or S is written onto disk.
3. If both the start line number or label name X|S and the end line number or label name X|Y are omitted, the entire program in memory is written onto disk.

Simple example:

1. STORE "ASCII,SHARP:A1" ENTER
   ..... Writes the program in memory to the disk with the volume name SHARP in drive A1 with the file name ASCII, in ASCII code.
2. STORE "ASCII",10,500 ENTER
   ..... Writes that part of the program in memory between line numbers 10 and 500 onto the disk in drive A0 with the file name ASCII, in ASCII code.

● **Function 2:** Writes a program as a secure file.

Format:STORE @ C [, X | S [, Y | T]] [, /D]

@ : Secure file indication

C : File specifications

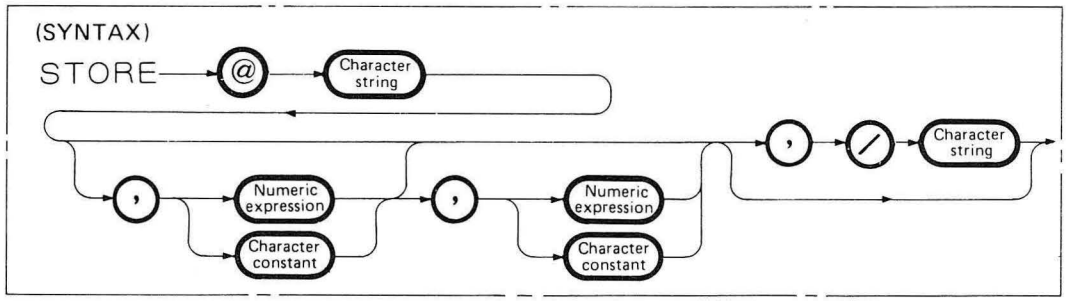
X : Write beginning line number

S : Write beginning line label name

Y : Write ending line number

T : Write ending line label name

D : Password



Explanation:

1. Entry of @ before file specifications C causes the program to be written as a secure program file in ASCII code.
   Similar to function 1 in all other details.
2. For security, see SAVE command, function 2 (page 349).

Simple example:

1. STORE @"START",10,200 [ENTER]
   ..... Writes onto the disk in drive A0, as a secret file with file name START the part of the program in memory from line 10 to 200.
2. STORE @"START:A1",/"J2" [ENTER]
   ..... Writes onto the disk in drive A1, as a secret file with file name START, the program in memory. A password "J2" is set at the same time. (Program listing is made possible by entering the password "J2" thereafter.)

<Helpful hint>
The CAT command (page 313) may be used to determine if a particular file has been written onto disk as a secure file.

LOAD

Loads (Reads) a program from a disk.

〈Purpose of use〉

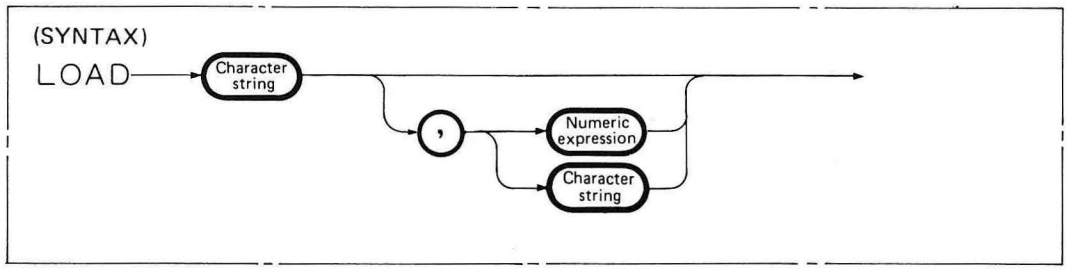
To load a program from disk into memory for execution, modification, addition, or other manipulations.

Format: LOAD C 〔, X | S〕

C : File specifications

X : Execution beginning line number

S : Execution ending line number



Explanation:

1. Loads a program (in intermediate language or ASCII code) from disk into memory, and executes it beginning with the program line with line number X, or with the line label name S.
2. If the execution 〔, X | S 〕 is omitted, but in the program is only loaded in manual operation, but in program mode is loaded and executed from the lowest line number.
3. ASCII-codes programs are converted into intermediate language format then load into memory.

Simple example:

1. `LOAD "FILE"` ENTER
   ..... Loads the program with the file name "FILE" from the disk in drive A0, into memory.
   The loaded program is not executed.
2. `LOAD "START:A0","STEP"` ENTER
   ..... Loads the program with the file name "START" from the disk in drive A0, into memory, and executes it starting with the program line with the line label name "STEP."
3. `100 LOAD "SUBPROG"`
   ..... Loads the program with the file name SUBPROG from the disk in drive A0, into memory, and executes it starting with the program line with the lowest line number. (Used in the program)

Note

1. Execution of this statement erases the entire program and all variables which are in the memory. Also, execution of this statement in a program will close all data files automatically (page 329).
2. If a unsecured file has been loaded following a secure file, the unsecured file will be treated as a secure file. To avoid this, the NEW statement (page 281) must be executed before the next file is loaded.

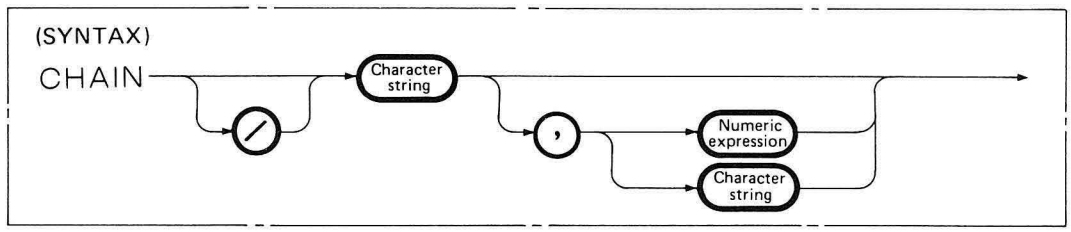
CHAIN

Reads a program from disk and chains it to the program in memory to execute it.

〈Purpose of use〉
To execute a program stored on disk by chaining it to the program which is in memory as a sub program.

Format: CHAIN [/] C [, X | S]

/ : File close option
C: File specifications
X: Execution beginning line number
S : Execution ending line number



Explanation:

1. Reads the program with file specifications C stored on disk (in intermediate language or ASCII code) and executes it starting with the program line with line number X or the line label name S.
2. Execution of this statement will erase the contents of the program in memory beginning from the lowest line number upwards of the loaded program.
   (The contents of any overlapped line number will be changed to the contents of the overlapping program.)
   All variables, as well as the contents of the program lines with line numbers lower than the lowest line number of the loaded program will be retained.
3. If the [/] is omitted before file specification C all data files will be closed (page 329) automatically.
4. IF the execution [, X | S] is omitted, the program is only loaded in manual operation, but in program mode is loaded and executed from the lowest line number.

Simple example:

```
500 CHAIN "SUBPRO",10
```

.....Reads the program with the file name"SUBPRO"from disk, and executes it from line number 10.
That part of the existing program in memory which is overlapped by the read program is replaced by the contents of the overlapping program.

Application example:

Let the exising program in memory be A.

```
10 DISP TAB 10,"*************************************"
20 DISP TAB 10,"* COMPUTE CIRCUMFERENCE AND AREA *"
30 DISP TAB 10,"*************************************"
40 DISP
50 A=20
60 B=100
70 A$="AREA"
80 END
```

```
50 INPUT "           RADIUS",R
60 DISP:IF R=0 THEN 110
70 DISP "    CIRCUMFERENCE=";2*PI*R
80 DISP "             AREA=";PI*R^2
90 DISP
100 GO TO 50
110*"END":DISP "COMPUTATION END":END
```

Program file with the file name B on the disk in drive A1.

Suppose the above two programs have been stored in memory and on the disk in drive A1, respectively.

CHAIN MERGE"B:A1" [ENTER] ..... Loads the file with the file name B from the disk in drive A1 into memory.

The existing program in memory is then updated as follows:

```
10 DISP TAB 10,"*************************************"
20 DISP TAB 10,"* COMPUTE CIRCUMFERENCE AND AREA *"
30 DISP TAB 10,"*************************************"
40 DISP
50 INPUT "           RADIUS",R
60 DISP:IF R=0 THEN 110
70 DISP "    CIRCUMFERENCE=";2*PI*R
80 DISP "             AREA=";PI*R^2
90 DISP
100 GO TO 50
110*"END":DISP "COMPUTATION END":END
```

As shown, program B is chained at line 50 follwoing the lines of program A existing in memory prior to execution.

<Helpful hint>

If this statement is executed in a subroutine, the routine saves the return address and control returns to the return address after execution.

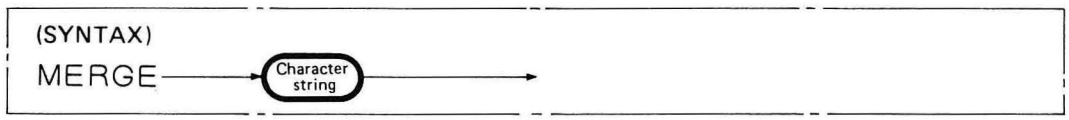
MERGE

Merges a program read from a disk with the existing program in memory.

〈Purpose of use〉
To merge two different programs, one stored in memory and the other on disk to produce one program.

Format: MERGE C

C: File specifications



Explanation:

1. Reads the program with file specifications C stored on a disk (in intermediate language or ASCII code) and merges it with the existing program in memory. The line numbers remain unchanged but the contents of overlapped lines are changed to the contents of the overlapping program.
2. Execution of this command erases all the variables in memory.

Simple example:

```
MERGE "CALCU" [ENTER]
```

..... Reads the program file with the file name "CALCU" from disk and merges it with the current existing program which is in memory.

Note

1. This statement cannot be used during a program.
2. If this statement is executed while there is no program in memory, the designated program is only loaded into memory.

Application example:

Let the existing program in memory be A.

```
 50 DISP TAB 10," 1    TEST"
 55 DISP
 60 DISP TAB 10," 2    MUSIC"
 65 DISP
 70 DISP TAB 10," 3    GAME"
 75 DISP
 80 INPUT A
 90 IF A=1 THEN 200
100 END
```

Application example:

Program with the file name ADDITION on the disk in drive A0.

```
 80 DISP TAB 10,"(SELECT NUMBER!)"
 90 INPUT A
100 IF A=1 THEN 400
110 IF A=2 THEN 1000
120 IF A=3 THEN 1500
130 DISP "          ERROR !!!":DISP TAB 10,
   "SELECT CORRECT NUMBER!"
135 GO TO 90
140 END
```

Suppose the above two programs have been stored, in memory and on the disk in drive A0 respectively.

MERGE "ADDITION" [ENTER]

..... Loads the file with the file name ADDITION from the disk in drive A0, and merges it with the existing program in memory (in this example, A).

(Execution results)

```
 50 DISP TAB 10," 1    TEST"
 55 DISP
 60 DISP TAB 10," 2    MUSIC"
 65 DISP
 70 DISP TAB 10," 3    GAME"
 75 DISP
 80 DISP TAB 10,"(SELECT NUMBER!)"
 90 INPUT A
100 IF A=1 THEN 400
110 IF A=2 THEN 1000
120 IF A=3 THEN 1500
130 DISP "          ERROR!!!":DISP TAB 10,
   "SELECT CORRECT NUMBER!"
135 GO TO 90
140 END
```

As shown, the program read from the disk is merged with the existing program in memory. Note that the contents of lines 80, 90 and 100 (overlapped) of the program A previously existing in memory are replaced with the contents of the overlapping program.

<Helpful hint>

The problem of overlapping line numbers may be avoided by using larger line numbers in the creation of the overlapping program. To merge two programs each in complete form, the line numbers of each program should be resequenced with REN command (page 98) before this statement is executed.

LOAD SUB

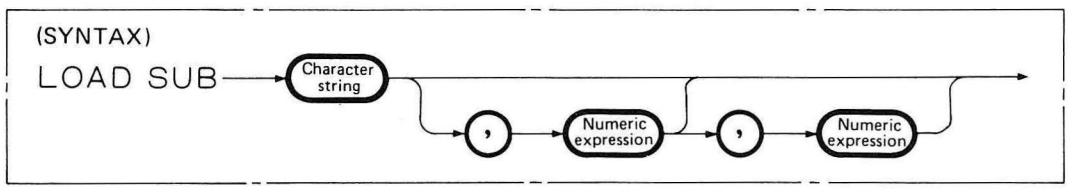
Loads (Reads) a program from disk.

〈Purpose of use〉
To load a program from a disk and append it to behind the existing program in memory to execute it.

● **Function 1:** Appends a program to the existing program in memory.

Format:LOAD SUB C [, X [, Y]]

C: File specifications
X: Line number
Y: Increment



Explanation:

1. Appends the program with file specifications C stored on disk (intermediate language format) to behind the current existing program in memory, and also renumbers the lines of the appended program with new line numbers sequenced in increments given by the value of Y, beginning with the value specified by X.
2. If starting line number X and increment Y are omitted, the lines of the program loaded from disk are renumbered with new line numbers sequenced in increments by 10, beginning with the value of the largest line number of the existing program in memory with increment by 10.
3. Line number X must be greater than the largest line number of the existing program in memory.

Simple example:

```
LOAD SUB "STEP,SHARP",500,20 ENTER
```

..... Loads the program with the file name"STEP"from the disk with volume name"SHARP"and appends it to the existing program in memory, renumbering the lines of the appended program with line numbers sequenced in increments by 20 beginning with line 500.

● **Function 2:** Appends a program to the existing program by erasing specified lines of the existing program in memory.

Format:LOAD SUB C; Z | S [, X [, Y]]

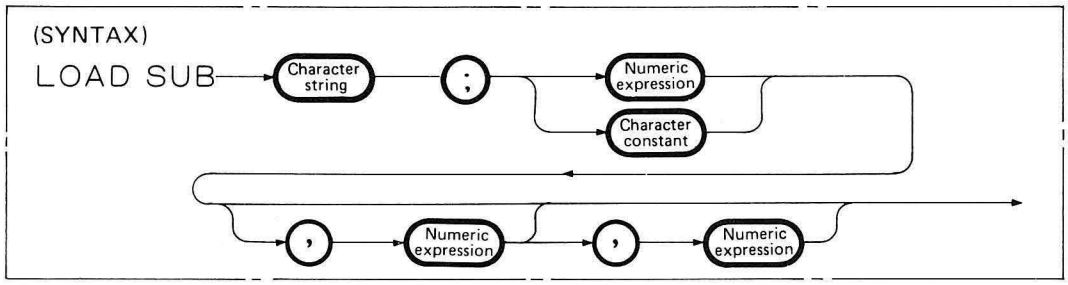
C: File specifications

Z: Starting line number of lines to be erased

S: Starting line label name of lines to be erased

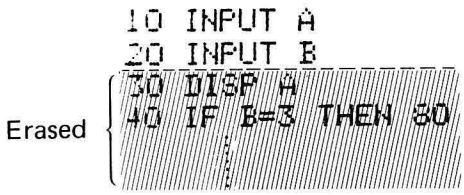
X: Updated line number of the appended program

Y: Increment



Explanation:

1. Erases the contents of the following lines beginning with line number Z or with the line label name S from the existing program in memory, and renumbers the lines of the program with file specifications C on a disk, with line numbers sequenced in increments by the value of numeric expression Y, starting with line number X before appending it to the existing program in memory.
2. This statement performs a function similar to function 1 if X and Y are omitted.
3. The updated line number X of the program to be loaded from disk must be greater than the line number of the line immediately preceding the line designated by Z or S. For example, if the following program exists in memory:



If 30 is specified as Z in the format, the contents of number 30 and all following lines are erased. The beginning line number of the program to be appended from disk must be greater than line number 20.

4. In executing this statement in a program, the starting line to be erased, as designated by Z or S, must be greater than the line number of this statement.

Simple example:

```
LOAD SUB "STEP";100 ENTER
```

..... Erases the contents of the following lines beginning with line number 100 of the existing program in memory. Loads the file with the file name "STEP" from the disk in drive A0, and appends it to the line immediately preceding line number 100 of the program in memory by renumbering the lines of the appended program with line numbers sequenced in increments by 10.

Note

1. This statement cannot be used in a subprogram.
2. This statement may not be used to load a program in ASCII code.

Application example:

[Program A]

```
100 DIM A$(4),B$(3,4)
110 DISP CLEAR
120 TABLE /,6,15,11,8,15;5,3,3,3,3,3
130 A$="TEL. DIRECTORY"
140 DISP A$
150 LOAD SUB "SUB1"
160 END
```

Let the program name of the program in memory be A. This command is shown on line 150.

```
10 CURSOR 11,6:DISP "NAME"
20 CURSOR 25,6:DISP "AREA"
30 CURSOR 34,6:DISP "TEL1"
40 CURSOR 46,6:DISP "TEL2"
50 SUBEND
```

Program stored on the disk in drive A0 with file name SUB1.

RUN [ENTER] ..... Two programs are appended by the LOAD SUB command specified on line 150 of program A in memory.

(Execution results)

```
100 DIM A$(4),B$(3,4)
110 DISP CLEAR :CHANGE DISP "G"
120 TABLE /,6,15,11,8,15;5,3,3,3,3,3
130 A$="TEL. DIRECTORY"
140 DISP A$
150 LOAD SUB "SUB1"
160 END
170 CURSOR 11,6:DISP "NAME"
180 CURSOR 25,6:DISP "AREA"
190 CURSOR 34,6:DISP "TEL1"
200 CURSOR 46,6:DISP "TEL2"
210 SUBEND
```

..... The line numbers of the program with the file name SUB1 are updated, and this program is appended to the program with the file name A.

[Program B] Execute different programs continuously.

Program in memory

```
10 A$="A":B$=":A1"
20 FOR I=1 TO 2
30 J=48+I
40 C$=CHR$ (J)
50 D$=A$+C$+B$
60 LOAD SUB D$;100,100,20
70 GO SUB 90
80 NEXT :END
90 PRINT "START"
100 GO TO 10
```

The file designated by character string D$ is loaded and appended to the existing program in memory with the line numbers sequenced in increments by 20, starting with line 100.

Program on disk

File name "A1"

```
10 PRINT "H O P"
20 LIST PRINT
30 RETURN
```

File name "A2"

```
10 PRINT "S T E P"
20 LIST PRINT
30 RETURN
```

(Execution results) The following list is output to the printer:

```
START
H O P
   10 A$="A":B$=":A1"
   20 FOR I=1 TO 2
   30 J=48+I
   40 C$=CHR$ (J)
   50 D$=A$+C$+B$
   60 LOAD SUB D$:100,100,20
   70 GO SUB 90
   80 NEXT :END
   90 PRINT "START"
  100 PRINT "H O P"
  120 LIST PRINT
  140 RETURN
START
S T E P
   10 A$="A":B$=":A1"
   20 FOR I=1 TO 2
   30 J=48+I
   40 C$=CHR$ (J)
   50 D$=A$+C$+B$
   60 LOAD SUB D$:100,100,20
   70 GO SUB 90
   80 NEXT :END
   90 PRINT "START"
  100 PRINT "S T E P"
  120 LIST PRINT
  140 RETURN
```

The file with the file name A1 is appended and executed.

The file with the file name A2 is appended and executed.

In this way, multiple programs can be appended to the existing program in memory and executed by altering line number 20.

ALOAD

Sets or resets the program auto-load function after power ON.

〈Purpose of use〉
To execute a specified program automatically after power ON.

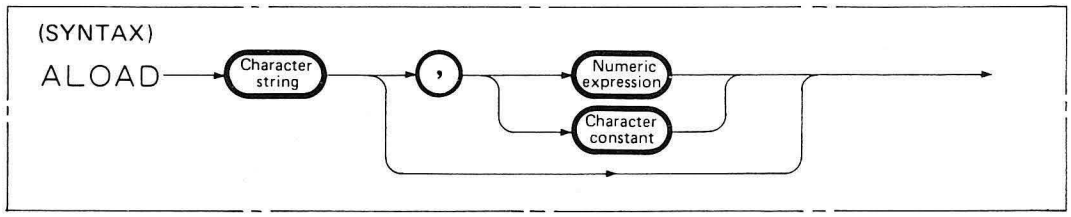
● **Function 1:** Sets the auto-load function.

Format: ALOAD C [, X | S]

C: File specifications

X: Line number

S: Line label name



Explanation:

1. Automatically loads the program with file specifications C (in intermediate language or ASCII code) into memory. (auto-load function)
2. If either line number X or line label name S is specified, program executes beginning with the program line specified. If neither line number X nor line label name S is specified, the program is only auto loaded.

Simple example:

1. ALOAD "MENU","GAME" [ENTER]
   ..... Sets the auto-load function for the program with the file name" MENU". After the program is auto-loaded, it is executed beginning with the program line with the line label GAME.
2. ALOAD "MENU",10 [ENTER]
   ..... Sets the auto-load function for the program with the file name MENU. After the program is auto-loaded, it is executed starting with program line 10.

● **Function 2:** Resets the auto-load function.

Format: ALOAD

(SYNTAX)

ALOAD ⟶

Explanation:

Resets the auto-load function.

Simple example:

ALOAD [ENTER] ..... Resets the auto-load function. After a power-on sequence is completed, the system waits for the entry of the FDOS command with the keyboard.

Note

1. The auto-load function can be used only with an FDOS master disk set in channel drive A0.
2. The auto-load function can be set for only one program stored on a single disk. (It cannot be set for two or more programs.)

< Detailed information >

Whether or not the auto-load function has been set for a particular program can be determined by viewing a catalog list of the programs on the disk (page 313).

Catalog list example (part)

```
CAT   A0                                  V1.0A
VOLUME NAME-SHARP
ALOAD       -TEST/SR:A0,  100
```

↑ Program file for which the auto-load function has been set

↑ Execution starting line number (label name)

5. OTHER FILE MANIPULATION COMMANDS

This section defines the commands which are applicable to machine language files, video RAM files, and definable key files:

MSAVE
MLOAD
VSAVE
VLOAD
GSAVE
GLOAD
KSAVE
KLOAD

All these commands are also executable in a program.

MSAVE

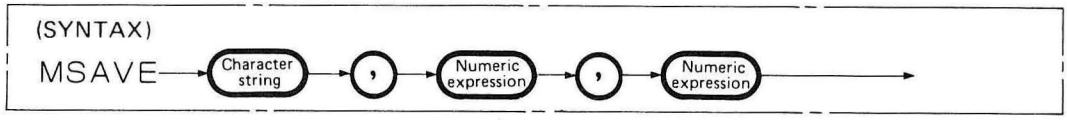
Saves (Writes) a machine language file.

Format: MSAVE C, X, Y

C: File specifications

X: Starting address

Y: Number of bytes



Explanation:

Saves the contents, on disk, in machine language, of the number of bytes specified by numeric expression Y from the starting address X in memory with file specifications C.

Simple example:

```
MSAVE "MACHINE,SHARP",&A000,1024 ENTER
```

..... Saves on the disk with volume name SHARP the contents of 1024 bytes in machine language starting with the address &A000 in memory which has the file name MACHINE.

< Detailed information >

The catalog list (page 313) can be used to check if the file was written.

MLOAD

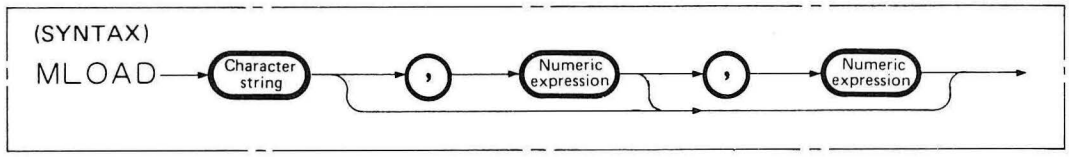
Loads a machine language file.

Format: MLOAD C [, X [, Y]]

C: File specifications

X: Starting address

Y: Number of bytes



Explanation:

1. Loads from disk the number of bytes specified by numeric expression Y from starting address X in memory, with the machine language file with file specifications C.
2. If either starting address X or number of bytes Y is omitted, the starting address or the number of bytes as specified in the MSAVE command is assumed.

Simple example:

```
MLOAD "MACHINE",&A000,1024 ENTER
```

..... Loads from disk the contents of 1024 bytes starting address &A000 in memory with the machine language file with file specifications MACHINE .

VSAVE

Saves a text video RAM file.

〈Purpose of use〉

To save character information from the CRT onto a disk.

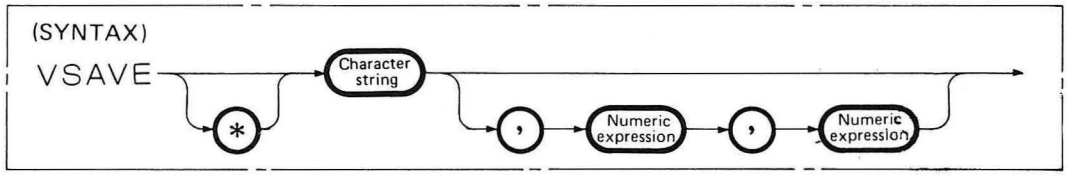
Format: VSAVE [*] C [, X, Y]

*: Screen information designation

C: File specifications

X: Save starting line number

Y: Save ending line number



Explanation:

1. Saves the character information between the (X + 1)th line and the (Y + 1)th line of the CRT screen onto disk, with file specifications C. Ruler lines, colors, and character modes (blinking or reversing) are not included.
2. If the asterisk is specified before file specifications C only, information concerning ruler lines, colors, and character modes (blinking or reversing) can be saved.
3. If X and Y are omitted, the character information from the entire CRT screen is saved onto disk.

Simple example:

1. `VSAVE "VIDEO",0,9` [ENTER]

   ..... Saves the text screen information between line 1 and line 10 on the CRT onto disk with file specifications VIDEO.

2. `VSAVE *"VIDEO2"` [ENTER]

   ..... Saves only the information concerning ruler lines, colors, and character modes (blinking or reversing) on the CRT onto disk with file specifications VIDEO2.

< Detailed information >

When character information and graphic information are intermixed on the CRT and both need to be saved onto disk, this command is used in conjunction with the GSAVE command (page 370). The result of processing will be the same whichever command is used first. To reproduce the contents of video RAM, the VLOAD and the GLOAD command (page 372) are used. (These are described later.)

VLOAD

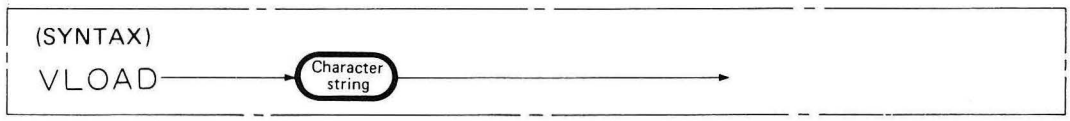
Loads text video RAM file.

〈Purpose of use〉

To load text display information from disk for display on the CRT.

Format: VLOAD C

C: File specifications



Explanation:

Loads the text video RAM file (character information, or ruler line, colors, and character modes—blinking or reverse) with file specifications C from disk into memory (text video RAM) and displays it on the CRT. The contents of the lines as saved with the VSAVE command (page 368) are displayed on the CRT. Any screen information being displayed on the CRT outside the range of the lines loaded is retained.

Simple example:

```
VLOAD "VIDEO SHARP" ENTER
```

..... Loads into memory the text display information from the file"VIDEO"stored on the disk with the volume name SHARP, and displays it on the CRT, beginning with the starting line whose contents had been saved onto the disk with the VSAVE command.

< Detailed information >

To save and reproduce blinking characters on the CRT screen:

1. First, save the characters and the character mode into separate files using the VSAVE statement.
2. Load the two files with the VLOAD statement. (The result of processing will be the same whichever file is read first.)

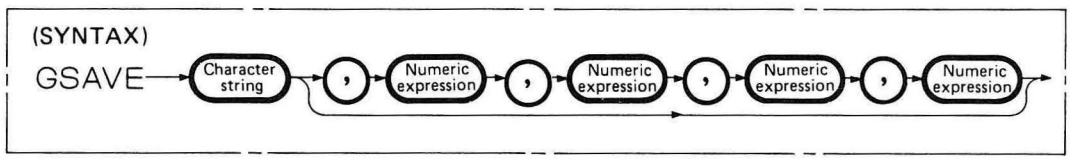
GSAVE

Saves a graphic video RAM file.

〈Purpose of use〉
To save information from the graphic screen (plotting screen) onto disk.

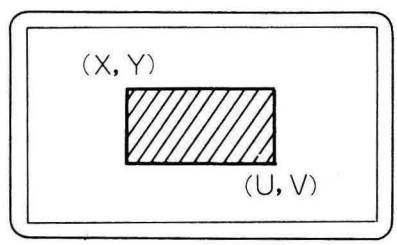
Format: GSAVE C [, X, Y, U, V]

C : File specifications
X : X-coordinate of the upper left corner
Y : Y-coordinate of the upper left corner
U : X-coordinate of the lower right corner
V : Y-coordinate of the lower right corner



Explanation:

1. Saves the graphic display information enclosed in the area (the shaded section in the figure below) designated by the upper-left graphic coordinates (X, Y) and the lower-right graphic coordinates (U, V) on the CRT onto disk with file specifications C. The contents of the entire graphic screen (plotting screen) set by the SDISP statement may be saved.
   The graphic display information that can be saved does not include text information (such as characters).



[Graphic screen]

2. If the upper-left coordinates (X, Y) and the lower-right coordinates (U, V) in the above format are omitted, the graphic information on the entire graphic screen is saved onto disk.

Simple example:

GSAVE "GRAPH,SHARP",10,10,100,100 ENTER

..... Saves the graphic display information enclosed in the area designated by the upper-left coordinates (10, 10) and the lower-right coordinates (100, 100) on the CRT onto disk with file specifications GRAPH, SHARP.

Note

Graphic display information is stored in graphic video RAM. For information about video RAM, refer to the Owner's Manual (page 3).

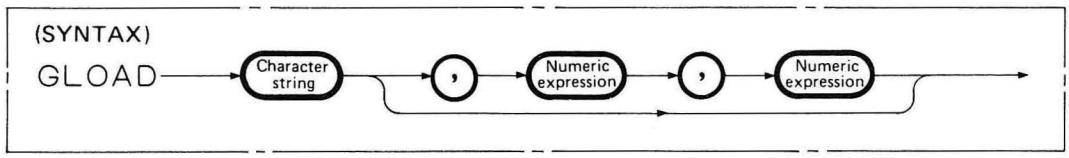
GLOAD

Loads a graphic video RAM file.

〈Purpose of use〉
To load graphic display information stored on disk for display on the CRT.

Format: GLOAD C [, X, Y]

C : File specifications
X : X-coordinate of the starting point
Y : Y-coordinate of the starting point



Explanation:

1. Loads the graphic video RAM file with file specifications C stored on disk into memory (graphic video RAM), and reproduces the graphic display information contained in the file on the graphic screen (plotting screen).
   The starting point of the display area where the graphic display information is reproduced is designated by the graphic coordinates (X, Y).
2. If the starting point X and Y are omitted, graphic display information is reproduced beginning at the coordinates designated by GSAVE command.

Simple example:

```
GLOAD "GRAPH,SHARP",10,10 [ENTER]
```

.....Loads the file designated by the volume name"SHARP" and the file name"GRAPH"into graphic video memory and reproduces the graphic display information on the CRT beginning at the graphic coordinates (10, 10).

Note:

1. Any excess portion of the graphic display information overflowing the graphic screen is not reproduced.
2. Only the screen 1 and screen 2 information is loaded if two plotting screens are set when a file where information of three screens has been saved by a GSAVE command is loaded by this command. (The number of the plotting screens set for saving must be the same as that of the loaded screens.)
3. Information is loaded on the first screen, and the second screen is cleared if two plotting screens are set when a file containing the information of one screen is loaded.
   (If the number of the screens to be loaded is less than that of the plotting screens set, the contents of the plotting screen to which nothing was loaded, is cleared.)
4. The screen number used when saving need not be the same as that used when loading.

KSAVE

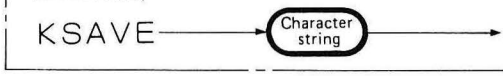
Saves a definable key file.

〈Purpose of use〉
To save the definitions of definable keys onto disk.

Format: KSAVE C

C : File specifications

(SYNTAX)



Explanation:

Saves the definitions of all definable keys (DEF 1 — DEF 20) onto disk with file specifications C.

Simple example:

```
KSAVE "KEY1,SHARP" ENTER
```

..... Saves the current definitions of all the definable keys with the file name "KEY 1" onto the disk which has the volume name "SHARP".

KLOAD

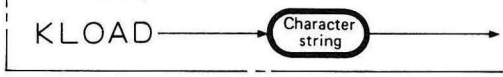
Load a definable key file.

〈Purpose of use〉
To define the contents of the definable keys stored on disk

Format: KLOAD C

C : File specifications

(SYNTAX)



Explanation:

Loads the definable key file with file specifications C from disk into memory, and redefines its contents. The definitions of all the definable keys (DEF 1 – DEF 20) are replaced by the contents thus loaded.

Simple example:

```
KLOAD "KEY1,SHARP" ENTER
```

..... Loads the contents of all the definable keys stored in the file with file specifications "KEY1, SHARP", and redefines these contents.

INDEX

	Appendix
Abbreviated command statements	46
Arguments	158
Arithmetic operators	3
Array elements	20
Array variables	19, 128
ASCII format	38
Background color	34, 242
Catalog list	47, 313
Channel drive No.	41
Character constants	10
Character strings	15
Character variables	12
Color code	237
Command	2
CRT display	31
Cursor	33, 222
Data item	36
Debug	71
DISK BASIC	305
Error code	Appendix 51
File	35
File name	35
File specifications	41
Fixed point format	17
Floating point format	18
FDOS	40
Function	23
Function name	198
Graphic character	276
Graphic coordinate	33
Graphic pattern	242
Graphic screens	32
Hexadecimal notation	10, 24
Integer-type numeric variables	11
Intermediate language format	38, 348
Line debug	71
Line number	61
Lock	308
Logical operators	5
Loop	149
Machine language	1, 280
Main program	158, 161
Manual operation	2
Memory map	Appendix 50
Numeric constants	9
Numeric expression	14
Numeric variables	12
Output screen	31, 239
Output formats of numeric values	16
Pointer	110
Plotting screen	32, 241
Program	62
Programming	62
Program file	38, 347
Program line	61
Pseudo graphic symbol	231
Random file	38, 341
Random numbers	28, 211
Real type numeric variables	12
Record	36
Record length	37
Relational operator	3
Sequential file	37, 330
Stack	7
Standard format	16
Statement	2
Subprogram	157, 160
Subroutine	134
Text coordinate	33
Text screen	31
Trace debug	73
Variable declaration	13
Volume name	35

COMMANDS, STATEMENTS

ABS 26
ACS 27
ALOAD 363
AND 5
ASC 23
ASN 27
ATN 27
AUTO 96
BACKUP 48
BASIC 44
BINPUT # 334
BL 227
BPRINT # 332
BYE 321
CALL 157
CAT 43, 313
CDEG 29
CDMS 29
CHAIN 354
CHANGE DISP 235
CHANGE PRINT 236
CHANNEL 288
CHR$ 23
CIRCLE 255
CLOSE 329
CLS 244
COLOR 237
CONT 82
COPY 50, 311
COS 27
CREATE 325
CURSOR 222
DATA 120
DEBUG 55
DEFFN 198
DEFINT 205
DEFREAL 206
DEFSTR 207
DEG 209
DELETE 93
DFK 58
DIM 128
DISP 107
DISP CLEAR 214
EDIT 90
END 127
ENTER 220
ERASE 95, 280
ERL 14
ERN 14
ERROR MUSIC 212
EXP 28
FIXED 17
FLOAT 18
FN 198
FOR/NEXT 149
GCHR 276
GCOLOR 242
GCURSOR 271
GDISP 268
GENTER 265
GET # 346
GINPUT 273
GLOAD 372
GO SUB 134
GO TO 131
GPRINT 270
GSAVE 370
IF---:ELSE 144
IF---THEN 141
IMAGE 113
INIT 45
INPUT 117
INPUT # 338
INT 26
KEY 163
KEYIN 165
KILLALL 54
KILL 312
KLOAD 374
KSAVE 373
LEFT$ 25
LET 104
LEN 24
LINE 226, 251
LINPUT 319
LIST 85
LN 28
LOAD 352
LOAD SUB 358
LOCK 308
LOG 28
LPRINT 317

MAXFILE 323
MAXS 190
MERGE 356
MID$ 25
MINS 192
MLOAD 367
MOD 30
MODIFY 29
MOVE 187
MSAVE 366
MUSIC 201
MVCUR 224
NEW 92, 281
NOT 6
NORMAL 16
ODISP 241
OFF ERROR 175
OFF KEY 180
OFF TIME 184
ON ERROR 172
ON ..... GO TO 138
ON ..... GO SUB 139
ON KEY 176
ON TIME 181
OPCHNL 299
OPEN 327
OR 5
PAINT 260
PEEK 29, 283
PI 9
PRESET 250
PRINT 110
PRINT # 335
POINT 30
POKE 283
POLLING 300
POP 171
POS 219
PSET 248
PUT # 345
RAD 210
RANDOM 211
RCV 294
RE 229
READ 121
REM 125
REN 98
RESET 218
RESTORE 123
RESUME 185
RETURN 136
RFORMAT # 343
RIGHT$ 25
RLIST 302
RNAME 316
RND 28
RUN 78
RV 228
SAVE 348
SCALE 246
SCOPY 52
SCROL 233
SDISP 239
SEARCH 194
SEND 290
SENREV 297
SEP # 240
SET 216
SGN 26
SIN 26
SPA 25
SQR 26
STATUS 29
STOP 126
STORE 350
STR$ 24
STRING$ 24
SUB 160
SUBEND 162
TAB 107, 110
TABLE 231
TAN 27
TERM 304
TIME 13
TR 100
TR STOP 102
UNLOCK 310
USR 284
VAL 24
VLOAD 369
VRUN 80
VSAVE 368
WAIT 203

* * * **MEMO** * * *

* * * **MEMO** * * *

* * * **MEMO** * * *

SHARP CORPORATION
OSAKA, JAPAN

Printed in Japan
Gedruckt in Japan
Imprimé au Japon
3G 0404-K
TINSE1028ACZZ