

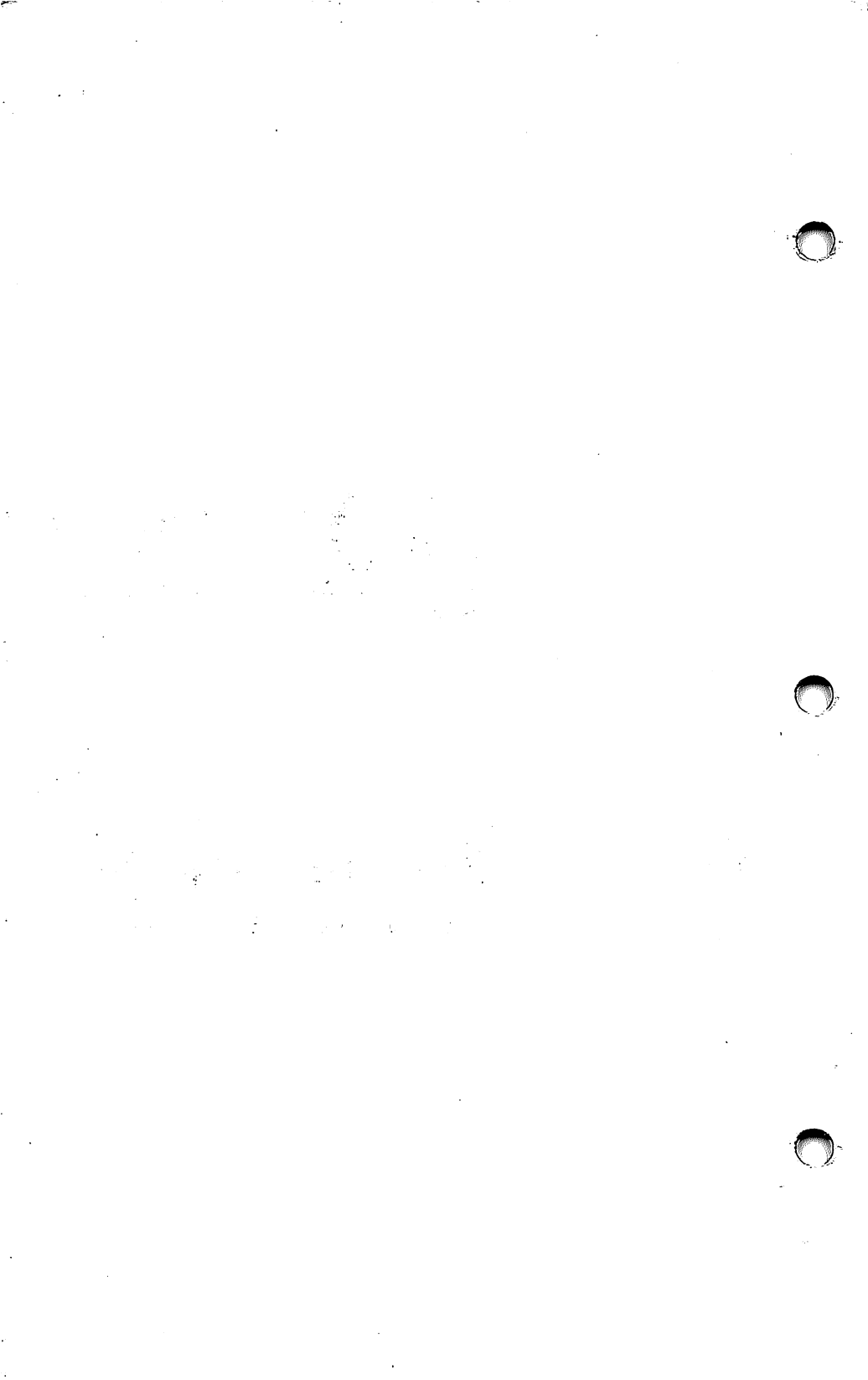
SONY®

Sony Disk BASIC

Part 1

MICRO COMPUTER **SMC-70**

© 1982 by Sony Corporation



NOTICE

TO USE THE SONY SMI-7032 IEEE-488 INTERFACE UNIT WITH SONY DISK BASIC

If you use the IEEE-488 interface unit with Sony disk BASIC, **be sure not to link the "IEEE488.PAC" file from the ROM on the IEEE-488 interface unit.**

To use the interface unit with Sony disk BASIC, load the file named "IEEE488.PAC" on the disk with a LINK command.

Execute the following :

LINK "DSK:IEEE488.PAC"

In this way, you can use the IEEE-488 interface unit with Sony disk BASIC.



Sony Disk BASIC

Part 1. General Information

PREFACE

The Beginner's All-purpose Symbolic Instruction Code (BASIC) is a programming language developed so that even a beginner can use it after very little study. Sony disk BASIC is a version of BASIC developed for use with the Sony SMC-70 and SMC-70G microcomputer. (Here after, we call this series of microcomputer "SMC-70" in this manual.) It has many sophisticated commands for programming, such as high-level data processing, table generation, graphic processing and data file handling.

The Sony disk BASIC programming reference manual is divided into two volumes, Part 1 and Part 2.

This volume, Part 1, explains a general introduction to the Sony disk BASIC. For the explanation of all the commands and functions of Sony disk BASIC, refer to "Part 2. Commands and Functions". Study these manuals carefully in order to be able to take full advantage of the SMC-70.

If you are using BASIC for the first time, you might want to refer to "Sony BASIC: Introductory Manual".

TABLE OF CONTENTS

Chapter 1	Sony disk BASIC outline	5
1-1	Features	6
1-2	General information on Sony disk BASIC	7
Chapter 2	Operation	19
2-1	Start up	20
2-2	Display screen	32
2-3	Debugging the BASIC program	36
2-4	File operation	48
Chapter 3	Program example	59
Appendix		75
1.	Keys to programming	76
2.	Trigonometric and hyperbolic functions	79
3.	Memory configuration	80
4.	FOR to NEXT loop processing and relationship with GOSUB	84
5.	Machine language subroutine	88
6.	Format of the link module	98
7.	I/O port assignment	101
8.	Video RAM address	103
9.	Character/key code table	110
10.	Sony disk BASIC reserved word code assignment	111
11.	Color code table	112
12.	Screen map	115



CHAPTER 1

SONY DISK BASIC

OUTLINE

1-1 FEATURES

Color graphics display

The numerous graphic commands make it possible to easily plot a point, or draw a line or square, or create a variety of original graphics on the graphic display screen.

One of the four resolution mode can be selected for the graphic display screen.

In standard and low resolution modes, sixteen colors can be used, and characters can also be displayed over the graphics. Therefore, it is possible to create a wide variety of graphs, tables, illustrations, etc.

Color character display

The display screen can display 40 or 80 characters (horizontally) and 25 lines (vertically). 8 colors for characters and 11 colors (the complementary colors of the character colors plus black, white and transparent) for the background can be displayed. Furthermore, the character and background colors can be interchanged, and the displayed characters can be blinked.

Enjoyable music function

The music subcommands specify the tones, durations, tempos, rests, and so forth in sequence. Therefore, it is possible easily to compose music and enjoy it while it is played.

Full screen editor

Sony disk BASIC has a Full Screen Editor capability, a character at any position on the display can be corrected at any time by moving the cursor to that position.

The characters are displayed in turquoise until they are entered into the computer by pressing the RETURN key. So that the characters which have not been entered can be recognized.

Extension of commands by LINK command

A LINK command makes it possible to load a machine language file on a device, such as a disk, without destroying the Sony BASIC program located in the memory. By utilizing this LINK command feature, it is possible to extend Sony disk BASIC functions, as required.

1-2 GENERAL INFORMATION ON SONY DISK BASIC

STATEMENTS AND LINE NUMBERS

A Sony disk BASIC program is a collection of lines, each of which consists of a line number and statement(s).

10 PRINT "HELLO"
Line number Statement
 Line

A line can contain up to 249 characters, including the characters for the line number. More than 249 characters can be written continuously, but even if you press the **RETURN** key, a beep is heard and the line is not entered. Clear the characters so that the characters will be less than 249 and press **RETURN**.

If a line number is followed by two or more statements, they are separated with a colon (:). If the **RETURN** key is pressed, the preceding data is regarded as one line and is stored in the memory together with the line number. Data input is then completed.

A long 1-line data (up to **RETURN** key) in the memory may be displayed in two or more lines on the display screen.

Line numbers to be assigned are 0 to 65529. Line numbers may be skipped and lines having a low line number can be entered later. In both cases, line numbers are rearranged in ascending order when executed.

CHARACTERS

Sony disk BASIC can handle the following characters¹⁾:

Alphabetic characters: Small letters (a to z) and capital letters (A to Z)

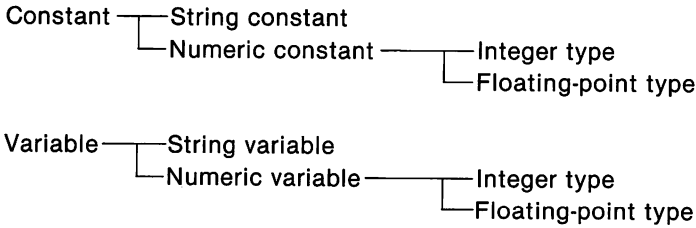
Numeric characters: 0 to 9

Special characters: %, ?, #, \$, [,], space and so forth

¹⁾ See Appendix 9 for a complete listing of characters which can be utilized.

CONSTANTS AND VARIABLES

A constant has a fixed value. A variable is the value stored in the location indicated by the variable name; it can be arbitrarily changed in the program. The constants and variables that can be handled by Sony disk BASIC are classified as follows:



String constant

A string constant is a set of characters (character string) that can be handled by Sony disk BASIC. It is enclosed in quotation marks (""). A string constant can contain up to 255 characters. A character constant containing neither a character nor space is called a null string.

Examples: "MIKE"
"@980"
" " (Null string)

Numeric constant

A numeric constant indicates a positive or negative number or zero. A negative numeric constant must be prefixed with a negative sign (-). For a positive numeric constant, the plus sign (+) may be omitted. Whether a plus sign is used or not when it is entered, it is omitted when output from the computer.

● Integer type constant

All integers from -32768 to +32767

Examples: -1
123

Note

When Sony disk BASIC is used, decimal numbers are entered as they are, and hexadecimal numbers are entered with an &H prefixed. All the input values are converted to binary numbers (0s and 1s) inside the computer.

Decimal notation	Hexadecimal notation	Decimal notation	Hexadecimal notation
0	&H0	9	&H9
1	&H1	10	&HA
2	&H2	11	&HB
3	&H3	12	&HC
4	&H4	13	&HD
5	&H5	14	&HE
6	&H6	15	&HF
7	&H7	16	&H10
8	&H8		

Inside the computer, integers are represented by 16-digit binary numbers. The first digit indicates a sign; 1 indicates a minus sign and 0 indicates a plus sign. Negative numbers are represented by complements.

Decimal notation	Binary notation	Hexadecimal notation
+ 32767	0111111111111111	&H7FFF
+ 32766	0111111111111110	&H7FFE
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
+ 2	0000000000000010	&H0002
+ 1	0000000000000001	&H0001
0	0000000000000000	&H0000
-1	1111111111111111	&HFFFF
-2	1111111111111110	&HFFFE
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
-32766	1000000000000010	&H8002
-32767	1000000000000001	&H8001
-32768	1000000000000000	&H8000

The range of integer type data that can be handled by Sony disk BASIC is as follows :

Decimal notation : -32768 to + 32767

Hexadecimal notation : &H0000 to &HFFFF

Values from &H8000 to &HFFFF are negative values.

When an integer value specifies a memory or video RAM address, a decimal value from 0 to 65535 is represented by a binary number (0s and 1s) (internal representation), and the internal representations of the following decimal numbers are the same :

-1 and 65535

-2 and 65534

. . .
. . .
. . .

-32767 and 32769

-32768 and 32768

- Floating-point type constant

A floating-point type constant is a numeric value represented in an exponential format.

The number of significant digits are 14, and digit 15 is rounded. The range of the exponent part is from -63 to +63. Accordingly, the range of the value that can be handled by Sony disk BASIC is from $-9.999999999999 \times 10^{63}$ to $9.999999999999 \times 10^{63}$.

Examples : 125

-998.562

1.34567E + 10

-5.3333E-8

Sony disk BASIC employs BCD (Binary Coded Decimal) for internal representation of a floating-point type data. So the data displayed on the screen shows the exact value of the data in the computer.

Variable

- Variable name

A variable name is a group of up to 255 alphanumeric characters, digits, period, or underline starting with an alphabetic character.

Note

A variable name must not be a Sony disk BASIC reserved word (command or function). However, if at least one character is different (added, etc.), the reserved word can be used as a variable name. Small and capital letters are not differentiated.

Examples : A

ABC

LISTI

~~LIST~~... Cannot be a variable name

- Type declarator

Whether a variable is the string or numeric type and if it is a numeric type, whether integer or floating-point type must be declared. There are two types of declaration methods.

Type declaration character : Added at the end of a variable name to indicate the type of the variable.

%	Integer type
#	Floating-point type
\$	String type

The same variable names can be used if their types are different.

Examples : A%

A#

A\$

Type declaration statement : A DEF statement is executed to specify a variable type.

DEF INT	Integer type
DEF FLT	Floating-point type
DEF STR	String type

Examples : DEF INT I-K
 DEF FLT A-H

The type declaration character added to a variable name is given priority over the type declaration statement even if the type declaration statement has been executed.

A variable name having neither a type declaration character nor a type declaration statement is regarded as a floating-point type.

- Array variable

An array variable is represented by a variable-name and a subscript. It is used to handle a series of data.

The same variable names are differentiated by their subscripts.

Examples : A(0), A(1), and A(2) are three different variables.

B(0,0,0) and B (0,0,1) are two different variables.

When an array variable is used, its type and size are declared in the DIM statement in advance.

EXPRESSIONS AND OPERATION

An expression consists of constants, variables, functions, and operational signs. There are four kinds of expressions :

- Arithmetic expression
- Character expression
- Relational expression
- Logical expression

Arithmetic expression

An arithmetic expression consists of numeric variables, constants, functions, and arithmetic operators. The operation result is a numeric type.

Arithmetic operators are as follows :

Arithmetic operator	Meaning	Example	Priority
+	Addition	$X + Y$	3
-	Subtraction	$X - Y$	
*	Multiplication	$X * Y$	2
/	Division	X / Y	
\	Division ¹⁾ of integer	$X \setminus Y$	
MOD	Remainder ¹⁾ of division of integer	$X \text{ MOD } Y$	
^	Power	$X \wedge 2$	1

If an expression has two or more arithmetic operators, the operation is performed according to the priority given in the above table. If there are two or more arithmetic operators having the same priority, the operation is performed from left to right. If parentheses are used, the operation in parentheses is given priority. If parentheses contain additional parentheses, the operation in the internal parentheses is performed first.

¹⁾ Division and remainder of integer

\ is used to indicate the quotient of a division of an integer, and MOD is used to indicate the remainder.

Example : $14 \div 4 = 3$ with a remainder of 2, that is, $14 \setminus 4 = 3$, $14 \text{ MOD } 4 = 2$

If the numeric data on the left or right of \ or MOD is not an integer type, the data is rounded to the nearest whole number and then division is performed.

$16.9 \setminus 4.5 = 3$, $16.9 \text{ MOD } 4.5 = 2$

The numeric values that can be used for obtaining quotients and remainders of integer division are in the range from -32768 to 32767.

Character expression

A character expression consists of string variables, constants, functions, and the plus sign. The operation result is a string type.

Example :

```
10 A$="THIS "  
20 B$="IS "  
30 C$="A PEN"  
40 D$="A BOOK"  
50 PRINT A$+B$+C$  
60 PRINT A$+B$+D$  
RUN  
THIS IS A PEN  
THIS IS A BOOK
```

Note

For the character expression, arithmetic operators other than the plus sign cannot be used.

Relational expression

A relational expression is used to compare two data values.

Its format and relational operators are as follows :

<Data> <Relational operator> <Data>

Relational operator	Meaning	Example
=	Equal to	X=Y
<	Less than	X<Y
>	Greater than	X>Y
<>, ><	Not equal to	X<>Y, X><Y
<=, =<	Less than or equal to	X<=Y, X=<Y
>=, =>	Greater than or equal to	X>=Y, X=>Y

The result of a relational expression may be true (–1) or false (0).
 True : The relational expression is formed.
 False : The relational expression is not formed.
 A relational expression can compare two numerical or string data values ; however, a numerical data value cannot be compared with a string data value, or vice versa.

- Comparison of two string data values
 When two string data values are compared, their character codes (see page 274) are compared character-by-character from the first characters.
 If all character codes are the same, the two string data values are equal. If one string data is longer than the other, it is greater. When comparing character strings, blanks are significant.

Example :

Character string to be compared	"A" and "A"
	"A" and "B"
	"AA" and "AB"
	"A" < "B"
	"AA" < "AB"
	"A" < "a"
	"AAA" and "AB"
	"AB" and "ABC"
	"A" < "A "

Logical expression

A logical expression consists of numerical constants, variables, functions, and logical operators such as NOT, AND, OR, and XOR. Before logical operation, the data is rounded to the nearest whole number, and the integer type data is acted upon as if it were a 16-bit binary number. See page 9 for the correspondences between the decimal and binary numbers.
 The range of the values that can be handled for logical operation is from –32768 to 32767. Outside this range, an overflow error occurs. If logical operation is performed for string data, a type mismatch error occurs.

The result of a logical operation (in bit units) is as follows :

NOT (not: Negation)

X	NOT X
1	0
0	1

AND (and: Logical multiplication)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (or: Logical addition)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR (Exclusive or: Exclusive logical addition)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

Examples :

Logical expression	Computation ¹⁾	Result
NOT 1	$\begin{array}{r} 0000000000000001 \\ \hline 1111111111111110 \end{array}$	-2
NOT -3	$\begin{array}{r} 1111111111111101 \\ \hline 0000000000000010 \end{array}$	2
3 AND 7	$\begin{array}{r} 0000000000000011 \\ 0000000000000111 \\ \hline 0000000000000011 \end{array}$	3
-1 AND 2	$\begin{array}{r} 1111111111111111 \\ 0000000000000010 \\ \hline 0000000000000010 \end{array}$	2
1 OR -1	$\begin{array}{r} 0000000000000001 \\ 1111111111111111 \\ \hline 1111111111111111 \end{array}$	-1
5 OR 8	$\begin{array}{r} 0000000000000101 \\ 0000000000001000 \\ \hline 0000000000001101 \end{array}$	13
4 XOR -3	$\begin{array}{r} 0000000000000100 \\ 1111111111111101 \\ \hline 1111111111111001 \end{array}$	-7
9 XOR 2	$\begin{array}{r} 0000000000001001 \\ 0000000000000010 \\ \hline 0000000000001011 \end{array}$	11

● Logical operation of a relational expression

Since the result of a relational expression is 0 (false) or -1 (true), the result of a logical operation is also 0 or -1.

Example : When $X = 3$;

$X > 0$ AND $X < 100$ will be -1 AND -1

The result is -1 (true).

$X > 0$ AND $X < 0$ will be -1 AND 0

The result is 0 (false).

¹⁾ The data under the line is the result in binary numbers.

Priority of operation

A logical operator can be used together with other operators. When they are used simultaneously, the priority of operation is as follows :

- ① Function
- ② $\wedge$ (Exponent)
- ③ $*$, $/$, $\backslash$, MOD
- ④ $+$, $-$
- ⑤ Relational operators ($<$, $>$, $=$, and so forth)
- ⑥ NOT
- ⑦ AND
- ⑧ OR, XOR

(Expressions enclosed in parentheses are executed first, irrespective of this priority.)

Type conversion

If an attempt is made to perform an operation or assignment for numeric data values of different types, one type is automatically converted to the other type.

- When numerical data of a certain type is assigned to a numerical variable of another type, the type of the numerical data is converted to that of the variable.

```
10 A%=45.6
20 PRINT A%
RUN
46
```

```
10 A#=45
20 PRINT A#
RUN
45
```

If an attempt is made to convert floating-point data outside the range of the integer data to the integer type, an overflow error occurs.

- When both integer and floating-point numerical data values are acted upon, each data value is handled as floating-point type.
- Before a logical operation, all data values are converted to the integer type.
- If an attempt is made to act upon string and numerical data values, a type unmatched error occurs.

CHAPTER 2

OPERATION

2-1 START UP

TO ACTIVATE SONY DISK BASIC

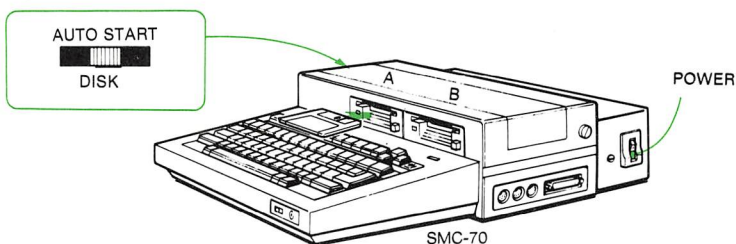
Sony disk BASIC was developed for use with the CP/M® Disk Operating System¹⁾ supplied with the floppydisk™ unit.

First, you must copy the CP/M system onto the Sony BASIC disk, referring to the CP/M manual for the correct procedure.

We recommend that you make a back-up copy of the Sony BASIC disk in case the disk is accidentally erased.

START UP

- 1 Insert the system disk into drive A.
- 2 Set the AUTO START switch on the left side of the SMC-70 to DISK.



- If you do not want to move the AUTO START switch, press **[C]** **RETURN** in the system monitor prompt mode.
- 3 Turn on the power to the system—the SMC-70, the display unit, and the printer—if required.

This will be displayed :

```
SONY SMC-70 -- CP/M Version 2.2
Release 1.0 (C) Copyright 1982, Sony Corporation
CP/M is a registered trademark of Digital Research Inc.

A> █
```

¹⁾ CP/M® is a registered trademark of Digital Research Inc.

To transfer control to Sony disk BASIC, type **B** **A** **S** **I** **C** and **RETURN**.

This will be displayed :



```
SONY Disk-BASIC Version 1.00
Copyright 1982 by SONY Corporation

Ready
█
```

Now the SMC-70 can be operated with Sony disk BASIC commands.

“Ready” is a message from the SMC-70 meaning that the computer is ready for programming. The “█” blinking in the next line is called the cursor, and indicates where the next character will be displayed when the keyboard is pressed.

The unit is now in a mode called the prompt mode and the SMC-70 is waiting for a command (an instruction to be executed) or for a program to be entered. When you press keys in this mode, characters corresponding to these keys are displayed on the screen in turquoise. The color of these characters is changed to white when the **RETURN** key is pressed, indicating that the line has been entered.

- If you want to execute some BASIC commands right after Sony disk BASIC is activated, you can write these commands after you type **B** **A** **S** **I** **C** separated by colons. Up to 127 characters including “BASIC” can be written.

PROGRAM INPUT

There are two ways to enter commands in the prompt mode.

Direct command input

After a command has been typed and the **RETURN** key pressed, Sony disk BASIC executes the command immediately and the prompt mode is restored.

Indirect command input

A line number is assigned, and statements containing commands are entered. When the **RETURN** key is pressed, Sony disk BASIC does not execute the statement, but stores the statement and the line number in memory.

When a RUN or GOTO statement is executed, the statements stored in memory are executed in line number order, then the prompt mode is restored.

Using labels

The GOTO or GOSUB command specifies the line to which the control of the program is to be transferred by a line number. It is possible to label a particular line so that you can recall its function more easily. For example, if you want to label line 20 "DATAINPUT", execute

```
20 *DATAINPUT
```

Now, to transfer control to line 20 by a GOTO statement, just execute

```
GOTO *DATAINPUT
```

You can write multi statements after the label separated each statement by a colon.

```
10 A$=TIME$(3)
20 IF A$="AM" THEN *MORNING
30 IF VAL (LEFT$(TIME$(2),2))<5
   THEN *AFTERNOON ELSE *EVENING
40 *MORNING: PRINT "Good morning!": END
50 *AFTERNOON: PRINT "Good afternoon!": END
60 *EVENING: PRINT "Good evening!": END
```

A label can be named in the same way as a variable name.

The commands in which a label can be used are as follows :

AUTO	LIST
DELETE	LLIST
EDIT	RUN
GOTO	RENUM
GOSUB	RESTORE
	DEBUG E,L,R

You cannot use a label in G command in the DEBUG mode.

You can also use the label in the direct command mode.

To stop program execution

Press the **ESC** key. The line number at which the execution stopped will be displayed on the screen. To restart the program execution, enter the CONT command.

If the program overruns and cannot be stopped by the **ESC** key, use the RESET button. The computer returns to the same mode as when the power was first turned on.

Note

When the RESET button is pressed, the memory contents will be cleared when Sony disk BASIC is activated again.

To stop scrolling on the display screen

While holding down the **CTRL** key, press the **S** key. A small cursor will blink to indicate that the listing has been interrupted.

Pressing the space bar displays the program line by line. To restart the scrolling, keep the space bar pressed or press a key other than the **ESC** key.

If an error occurs during program execution

The computer displays the line number of the erroneous line and an error message. The program execution stops at that point. See Chapter 3 in Part 2 for a list of all the error messages.

PROGRAM CORRECTIONS

If you notice an error in a program list on the display screen, you can correct it.

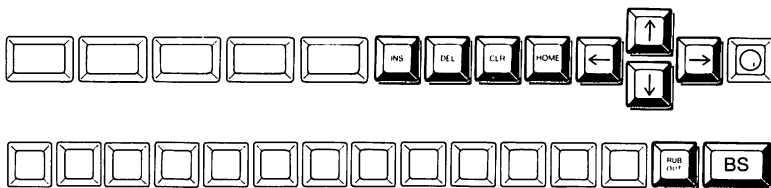
Sony disk BASIC has a Full Screen Editor capability, that is a character at any position on the display can be corrected at any time by moving the cursor to that position.

Using the Full Screen Editor capability, you can correct the errors in a program by listing it on the display. You can also execute a statement in the direct command mode, check the result and add that statement to a program by simply inserting a line number before the statement. At the same time, Sony disk BASIC has an edit mode in which a program can be displayed and corrected line by line.

It is very helpful to correct a long program whole of which cannot be listed on screen at a time.

The Full Screen Editor function allows you to correct any error on the display even in the edit mode.

Make corrections using these edit keys.



The edit keys are used as follows :

Key	Pressed by itself	Pressed together with SHIFT	Pressed together with CTRL
RUB OUT BS	The character to the left of the cursor is deleted. The character at the cursor position and following characters are moved one space to the left.		
	The cursor moves one space to the right.	The cursor moves to the end of the line on the display.	Invalid (Y is displayed.)
	The cursor moves one line up.	Invalid ¹⁾	Invalid (W is displayed.)
	The cursor moves one line down.	Invalid ¹⁾	Invalid (N is displayed.)
	The cursor moves one space to the left.	The cursor moves to the beginning of the line on the display.	Invalid (V is displayed.)
HOME	The cursor moves to the beginning of the program line.	The cursor moves to the end of the program line.	The cursor moves to the upper-left corner of the display.
CLR	The characters between the cursor and the end of the program line are deleted.	All characters preceding the cursor are deleted. The character at the cursor position and the following characters are moved to the beginning of the program line.	All characters on the display are deleted and the cursor moves to the upper-left corner of the display.
DEL	The character at the cursor position is deleted. The following characters are moved one space to the left.	Invalid	The line on which the cursor is located is deleted. The following lines move up one line.
INS	The character at the cursor position and the following characters are moved one space to the right and a space is inserted at the cursor position.	The cursor becomes a quarter of its height. Any number of characters can be inserted at the cursor position. By pressing SHIFT + INS or RETURN the normal mode is re-entered.	The line on which the cursor is located is moved down one line and a blank line is inserted.

¹⁾ In the edit mode, the corrected line is entered and the preceding or succeeding line is displayed by pressing **SHIFT** + either  or . For details, see the EDIT command in Part 2.

OUTPUT TO PRINTER

To print a program list

Enter

```
LLIST ^1)
```

The program list will not be displayed on the screen, but it will be printed out by the printer.

To print out lines when they are entered

Press the **P** key while holding down the **CTRL** key. Programs, data or commands entered from the keyboard will be printed out by the printer each time the **RETURN** key is pressed. The results of program execution will also be printed out. In addition, the information entered and the program results will be displayed on the screen.

To deactivate the printer, press the **P** and **CTRL** keys again.

Note

Graphic data will not be printed out.

To print out the result

To print out the result by the printer, the **PRINT @1** statement is used. For example, to print "Hello!" on the printer, enter

```
PRINT @1, "Hello!"^1)
```

In the following program, line 30 is used to print out the results.

```
10 FOR I=1 TO 10
20  A=A+1
30  PRINT @1,I,A
40 NEXT I
```

The results will be printed out by the printer instead of being displayed on the screen.

¹⁾ ^ mark in this manual indicates of the pressing of **RETURN** key.

FUNCTION KEYS

Keys **F1** to **F5**, which are located at the upper-left of the keyboard, are called function keys. When Sony disk BASIC is activated, these keys are defined as follows :

Function key number	Definition
F1 (F1)	List
F2 (F2)	Edit
F3 (F3)	Auto
F4 (F4)	Debug
F5 (F5)	Run
F6 (SHIFT + F1)	Wipe' ↵
F7 (SHIFT + F2)	Console
F8 (SHIFT + F3)	Gclear
F9 (SHIFT + F4)	Renum
F10 (SHIFT + F5)	Keylist' ↵

By using the function keys, it is possible to enter the indicated definition quickly.

The definitions of the keys can be displayed on the screen by using the KEY LIST command.

```
KEYLIST
 1      List
 2      Edit
 3      Auto
 4      Debug
 5      Run
 6      Wipe' M
 7      Console
 8      Gclear
 9      Renum
10      Keylist' M
```

The mark **M** means that the **RETURN** key function is included in the definition.

To change function key definitions

Function key definitions can be changed by using a DEF KEY command. You can define up to 16 characters, including the control codes such as CHR\$(13). A control code is counted as one character. The definition remains valid until the key is redefined or reset (by switching the power OFF and then ON or Sony disk BASIC is re-activated). When reset, the definition of each key returns to the initial definition.

MACHINE LANGUAGE SUBROUTINES

Machine language subroutines in memory can be called from Sony disk BASIC with a CALL statement.

Available memory area

The CP/M and the Sony disk BASIC interpreter occupy large areas in the main memory. Therefore, when generating machine language subroutines, be careful not to overwrite these areas.

If machine language subroutines are written in the BASIC work area, the subroutines may be overwritten by Sony disk BASIC programs or variable data and vice versa. To avoid this, it is better to limit the Sony disk BASIC memory area, saving some space for machine language subroutines to be generated. This is done by using the MCLEAR command.

For example, when

```
MCLEAR &HA500 , &HBFFF
```

is entered, Sony BASIC programs are limited to addresses &HA500 to &HBFFF, and machine language programs can be written outside of the area.

Note

If Sony BASIC programs are already stored in memory, the programs will all be erased when the MCLEAR command is executed.

For details about the memory configuration, See Appendix 3.

How to call machine language subroutines

The machine language subroutines can be called from Sony BASIC programs using the CALL statement. To call a subroutine, set the subroutine starting address to a variable, then specify the variable with the CALL statement. For example, when the following is entered,

```
100 SUB=&HC000  
110 CALL SUB
```

address &HC000 is called.

Note

When calling a machine language subroutine, the values of the variables used in the program to that point can be passed to the subroutine as they are, or the subroutine processing results can be used in the BASIC program. For these purposes, in Sony disk BASIC the variable to be passed can be added to the CALL statement, and the subroutine processing results can be returned to the main program by the ANN function. (Refer to Appendix 5 for details.)

How to save programs which include machine language subroutines

In Sony disk BASIC, machine language subroutines can be saved in the same file as the Sony disk BASIC main program by using the command SAVE or CSAVE.

For example, to save a program with subroutines under the name of "TEST" on a disk, enter the following :

```

      starting address of the subroutines
      _____
SAVE  "TEST" , &HC000 , &HC300 ↵
                        _____
                        end address of the subroutines
```

The machine language subroutines are recorded on the disk at the same time as the Sony disk BASIC program. To load the Sony BASIC program and the machine language subroutine, enter the following :

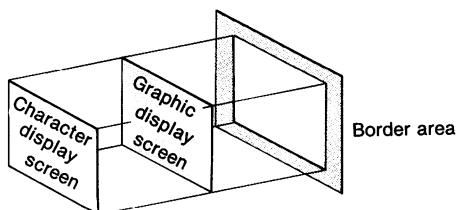
```
LOAD "TEST" ↵
```

Note

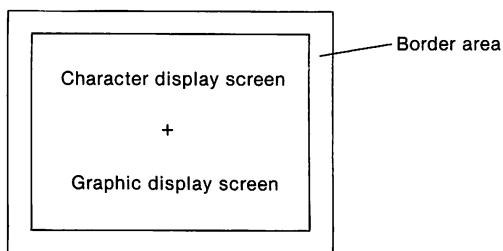
When you save a program with machine language subroutines, low and high memory addresses of the Sony BASIC area are saved at the same time. Thus, the memory area will automatically be limited if you load such a program.

2-2 DISPLAY SCREEN

The SMC-70 display screen configuration is as follows :



As viewed from the front :

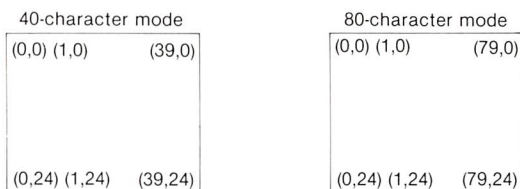


The character display screen has priority over the graphic display screen, and a graphic display can be seen only through the transparent portions of the character display screen.

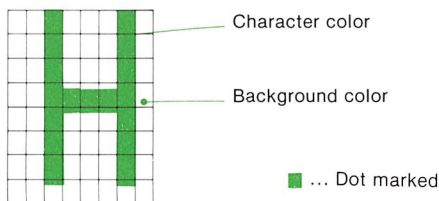
The border area is always seen ; however, no data or graphics can be displayed on it.

Character display

The character plane can display 80 or 40 characters per line and 25 lines (vertically). A location on the character plane is specified by horizontal and vertical coordinates.



A character is displayed with 8×8 dots on the character display screen. For example, "H" is displayed as follows :



The character display mode is preset as follows :

- Number of characters/line : 80
- Number of lines/screen : 25
- Character color : Turquoise (characters not entered)
White (characters entered)
- Background color : Transparent
- Character blinking : No
- Screen scrolling¹⁾ : Yes

Note

The SMC-70 contains a programmable character generator (PCG) which makes it possible to redefine a character form (font) with a DEF FONT statement.

In Sony disk BASIC, the CCOLOR statement can specify character and background colors and whether it blinks or not.

These are called a character's attributes.

¹⁾ Scrolling means the movement of the entire screen up one line so that a new line can be entered at the bottom of the screen even after the last line displayed on the screen is filled.

Graphic display

This screen is used to display graphics by plotting dots in the desired positions on the screen. Colors can be specified for each dot.

The graphic display screen has four modes which are classified depending on the resolution level.

- Low-resolution mode: 160×100 dots, 16 colors, 4 pages
- Standard mode: 320×200 dots, 16 colors (Initial setting)
- High-resolution mode: 640×200 dots, 4 colors, (2 types)
- Super high-resolution mode:
 640×400 dots, monochrome

The mode is selected by a GMODE command.

There are two possible color combinations in the high-resolution mode: red/green/blue/black or red/green/white/black. The GMODE command can also specify the color combination to be displayed.

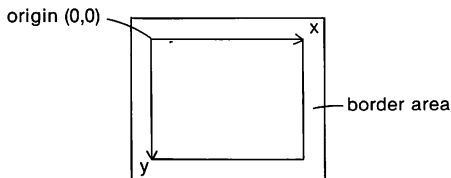
The low resolution mode has four pages which are displayed separately. A GPLANE command designates which page is to be displayed and on which page graphic data is to be written.

Notes

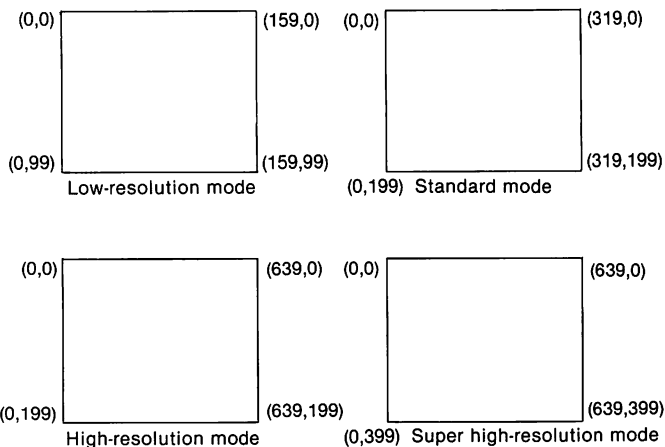
- When a monochrome monitor is used, 16 and 4-color displays are in shades of gray.
- The super high-resolution mode employs interlace scanning. Because of this, when a normal CRT display is used, the display may flicker. It is best to use a CRT display using a fluorescent substance with a long afterglow, such as the Sony CPD-120 character display.

A location on the graphic display screen is specified by horizontal and vertical coordinates.

Initially, the x and y axes of the coordinate system on the graphic display screen are set as follows :

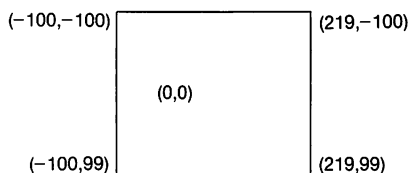


The initial coordinates of each mode are as follows :



If a position outside of the screen is specified in a graphic command, an error does not occur, but the graphic data cannot be seen on the screen.

The origin of the coordinate system can be moved to a position between -16384 to $+16383$ in any direction along the x and y axes of the initial coordinate system using a GLOCATE command. For example, if GLOCATE (100, 100) is specified in the standard mode, the coordinates will be.



A graphic command specifies the display location, display color, background color and the mode. When a graphic command is executed, logical operation will be performed for the current and specified display color codes.

The color indicated by this logical operation result is displayed. See Appendix 11 for the color and mode codes.

2-3 DEBUGGING THE BASIC PROGRAM

Debugging is a procedure to find an error in a program. The debug mode for BASIC is one of the greatest features of the Sony disk BASIC. The DEBUG command is not included in the Sony disk BASIC interpreter. The program for debugging is filed with the name of "DEBUGGER.PAC", which can be moved into the main memory with a LINK command at any time.

The debugger may be linked before or after the program to be debugged is entered. The program previously stored in the main memory is retained even if the LINK command is executed. If a program error is detected with the commands of the debugger, correct the program with an E (Edit) command in the ordinary way.

LOADING THE DEBUGGER

In the prompt mode, enter the following :

```
LINK "DEBUGGER.PAC"
```

Thus the debug mode is enabled.

Note

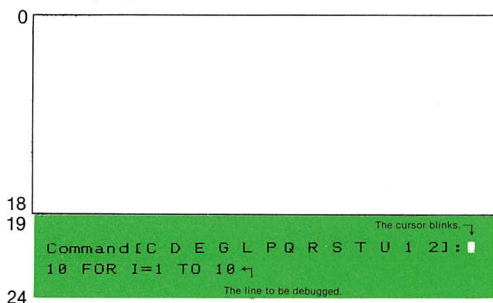
Use a LINK command when loading a debugger. If a LOAD command is entered by mistake, an error occurs and the debugger is not loaded.

HOW TO DEBUG

To start debugging, enter the following :

```
DEBUG line number ↵
```

The line number indicates from which line the debugging is started. The display will be :



The lowest 6 lines are reserved for the debug operation.

If a line number is not specified, the line where program execution was interrupted by an error or STOP, or which has been displayed last with LIST, EDIT or AUTO, is assumed to have been specified.

The last 6 lines on the screen are used as a debugger display area ; ordinary data is displayed in the first 19 lines. The debugger command menu is displayed in the debugger display area.

The meaning of each command is as follows :

C: Clear	R: Run
D: Debug	S: Step
E: Edit	T: Trace
G: Goto	U: Undefined variable
L: List	1: First print items
P: Print	2: Second print items
Q: Quit debugger	

If the command is input from the keyboard, the meaning and input format of the command will be displayed. <CR> indicates to press the **RETURN** key. Enter the command according to this format. If "C" is input, for example, the following is displayed.

```
Clear <CR>
Clear █
  10 FOR I=1 TO 10
```

If the space bar is pressed instead of the first character of a command when the command menu is displayed, the menu disappears and the cursor blinks at the leftmost column (direct mode). In this mode, a line of the Sony disk BASIC commands can be entered as usual.

```
(Direct Mode)
```

```
█ ← The cursor blinks.
```

DEBUGGER COMMAND LIST

Command name	Entering format (Specification in [] can be omitted.)	Meaning
C	Clear ↵	Clears all the variables.
D	Debug [line number] ↵	Initializes the debugger.
E	Edit [line number] ↵	Corrects the line.
G	Goto [start line No.] [/break line No.] [:statement number] [(number of repetitions)] ↵	Executes a program between specified lines or up to the specified statement
L	List [start line number] [-end line number] ↵	Displays the program list.
P	Print [@ device number,] expression separator... ↵	Outputs the specified constant or variable to the screen.
Q	Quit $\begin{Bmatrix} Y \\ N \end{Bmatrix}$ ↵	Terminates the debug mode when "Y" is entered.
R	Run [start line number] ↵	Executes the program.
S	Step [number of statements] ↵	Executes a specified number of statements of the program.
T	Trace [number of lines] ↵	Executes a specified number of lines of the program.
U	Undefined variable detection $\begin{Bmatrix} Y \\ N \end{Bmatrix}$ ↵	Sets or resets the undefined variable detection which causes an error when a variable which is not assigned to any value appears.
1	PRINT items to be printed ↵	Executes the specified PRINT statement each time a statement in the program is executed.
2	PRINT items to be printed ↵	

Of these commands, the C, E, P, and R commands can be used exactly in the same way as the ordinary Sony disk BASIC commands.

With the L command, the same result as the LIST command will be obtained; however, a screen editing function is not operative on the program listed by the L command.

Editing must be effectuated with the E (Edit) command.

TO EXECUTE PART OF A PROGRAM

In the debug mode, a program can be executed from the middle of the program or only part of it can be executed.

To execute a whole program

- 1 Press the **[R]** key when the debugger command menu is displayed.

```
Run [<line#>]
Run █ ← The cursor blinks.
```

- 2 Press **[RETURN]**.
The program will be executed.

To execute a program from the middle

Specify the line number to start execution. There are two commands to start execution:

1) R (Run) command

- 1 Press the **[R]** key when the debugger command menu is displayed.

```
Run [<line#>]
Run █ ← The cursor blinks.
```

- 2 When the above display is made, key in the start line number in the cursor position, then press the **[RETURN]** key. To start from line 50, enter the followings

Run 50 ↵

On pressing the **[RETURN]** key, execution starts. With the R (Run) command, the program is executed after all the variables are cleared.

2) G (Goto) command

- 1 Press the **[G]** key when the command menu is displayed.

```
Goto [<line#>][</>]<line#>[:n][<c>],... ]
Goto █
```

- ② Input the execution start line number at the blinking cursor position, then press the **RETURN** key.

The variable values before the G command is executed are retained.

If the specification of the line number is omitted, the execution will start from the next line of the line specified in the last DEBUG command, where the execution has been interrupted with G command or which has been executed with S or T command.

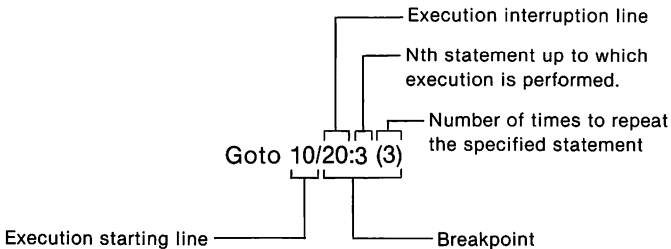
The G (Goto) command can be used in the same way as the R (Run) command. The differences between the G and R commands are as follows :

- A breakpoint can be specified for the G command.
- Variables are not cleared for the G command.

To execute a program up to the breakpoint

When a G (Goto) command is used, a program portion between specified lines can be executed. If a line includes several statements, you can specify a particular statement for the breakpoint where the program execution is to be stopped. When a line is repeatedly executed in a loop, the number of times to be repeated can also be specified.

The values specified with the G (Goto) statement have the meanings as follows :



For example, if "Goto 10/20:3(3) ↵" is entered for the following program :

```
10 FOR I=1 TO 10
20 PRINT "I=";I,:A=I^2:B=I^3:PRINT A,B
30 NEXT
```

The program starts and execution is interrupted when the following is displayed.

```
I=1      1      1
I=2      4      8
I=3
```

In this example, the program is interrupted when the third statement ($B = I^3$) in line number 20 has been executed three times.

With the G (Goto) command, up to three breakpoints can be specified and the program execution stops when the first breakpoint is encountered.

Note

When the specification of the nth statement is omitted, 0 (up to the end of the preceding line) is assumed to have been specified. If the specification of the number of times to repeat the specified statement is omitted, 1 is assumed to have been specified.

TO PERFORM STEP-BY-STEP OPERATION IN LINE OR STATEMENT UNITS

In the debug mode, part of a program can be executed by specifying the number of lines or statements to be executed in one operation. Use a T (Trace) command to execute a program line by line (press the **T** key in the command menu state). Use a S (Step) command to execute a program statement by statement (press the **S** key in the command menu state).

T (Trace) command

```
Trace [nn]  
Trace █ ← The cursor blinks.  
10 FOR I=1 TO 10
```

Enter the number of lines to be executed in the cursor position, then press the **RETURN** key.

S (Step) command

```
Step [nn]  
Step █ ← The cursor blinks.  
10 FOR I=1 TO 10
```

Enter the number of statements to be executed in the cursor position, then press the **RETURN** key.

In both cases, the program will restarts from the line where program execution was interrupted immediately before or which has been specified in DEBUG. A program can be traced by repeatedly executing a T (Trace) or S (Step) command; however, these commands can be executed only immediately after execution has been interrupted at the breakpoint (specified by the G (Goto) command) or by a T (Trace) command, S (Step) command, or **ESC** key. Otherwise, "Error (139): Can't CONTINUE" will be displayed.

Notes

- If the number of lines/statements is not specified, 1 is assumed to be specified.
- The execution speed is lowered when a T or S command is used.

TO CHECK THE VALUE OF A VARIABLE ON THE WAY OF PROGRAM EXECUTION

To check when program execution is interrupted

Use a P (Print) command. Press the **P** key in the command menu state.

```
Print [<expr>]  
Print █
```

Input a variable or a constant, or an expression including these to be displayed from the cursor position and press the **RETURN** key. The P command will immediately be executed just as PRINT in the direct command mode; the result will be displayed between the top and the 19th lines on the screen.

To check while the program is being executed

Use a 1 (1st print item) or 2 (2nd print item) command to continuously check the changes of a variable. The check result is displayed in the 24th line for the 1 command; the 25th line for the 2 command.

If an expression is specified in 1 or 2 command, the specified expression is displayed every time a statement is executed. Using a 1 or 2 command, you can check the value of variables while program is being executed.

Suppose the following program as an example.

```
10 DIM A(6)  
20 FOR I=0 TO 5:READ A(I):NEXT I  
30 DATA 5,103,-6,82,0,-38
```

If the PRINT statement is entered as follows ;

```
2nd print items
```

```
Print "A(";I;")=";A(I)
```

The display on the last line on the screen changes as follows as the program is executed.

Line of program to be executed	Display in last two lines
DIM A (6)	A (0)=0
FOR I=0 TO 5	A (0)=0
READ A (I)	A (0)=5
NEXT I	A (1)=0
READ A (I)	A (1)=103
NEXT I	A (2)=0
READ A (I)	A (2)=-6
NEXT I	A (3)=0
READ A (I)	A (3)=82
NEXT I	A (4)=0
READ A (I)	A (4)=0
NEXT I	A (5)=0
READ A (I)	A (5)=-38
NEXT I	A (6)=0
DATA 5, 103, -6, 82, 0, -38	A (6)=0

To invalidate the statement specified in a 1 (1st print items) or 2 (2nd print items) command, input a 1 (1st print items) or 2 (2nd print items) command, then delete the statement in the lower part of the screen by pressing the **CLR** key.

For example, the statement specified in a 1 (1st print item) statement is deleted as follows :

```
Command [C D E G L P Q R S T U 1 2]:
```

```
1st print items
```

```
PRINT "A(";I;")"
```



CLR



```
1st print items
```



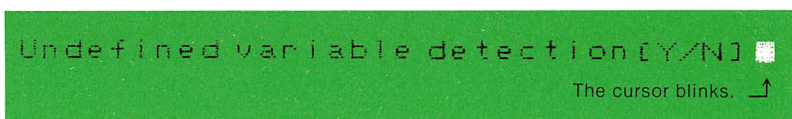
Deleted.

Note

If you want to specify a statement other than a PRINT statement as 1 (1st print items) or 2 (2nd print items) command, after entering 1 or 2 command, delete "PRINT" using **BS** key, etc. and enter the desired statement. Thus, the specified statement will be executed.

TO CHECK AN UNDEFINED VARIABLE

Normally the variable to which no-value has been assigned is handled as 0 if it is of the numeric type; a null string if it is of the character type. A debugger command can display an error indication when a variable to which no value has been assigned appears on the right side of an expression in a program. Press the **[U]** key in the command menu state.



Press the **[N]** key to make an error indication and the display returns to the command menu. If there is an undefined variable when a program is executed by a G command, the following error message is displayed and the program execution stops.

ERROR (131) in 200: Undefined variable



line number where the error occurs.

To terminate the undefined variable detection

To terminate the undefined variable detection, press **[U]** key in the command menu state, then press the **[N]** and **RETURN** keys.

TO MODIFY THE PROGRAM

When errors in commands, statements and expressions of the program are detected, correct them with the E (Edit) command. Rules for a use of the E command are the same as those for the EDIT command under the normal mode.

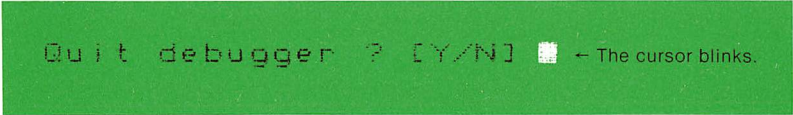
Verify if the modification is correctly effectuated with the L (List) command. Note that the Full Screen Editor function is not available under the debug mode.

TO INITIALIZE THE DEBUGGER

Use a D (Debug) command to initialize and re-enter the debugger. In the command menu, press the **[D]** key and input a line number to specify the line to be debugged, then press the **RETURN** key. Thus, the initial debug mode state is restored.

TO RETURN TO THE NORMAL MODE

Use a Q (Quit debugger) command to terminate the debug mode and to return to the normal mode. Press the **[Q]** key in the command menu mode.



```
Quit debugger ? [Y/N]  ← The cursor blinks.
```

To terminate the debug mode, press the **[Y]** and **RETURN** keys. The debugger display area disappears from the screen and the normal mode is re-entered ; however, since the debugger program is maintained in the main memory, it can be re-called at any time.

2-4 FILE OPERATION

Using the Sony disk BASIC, programs and data can be stored (saved) and loaded again into memory any time.

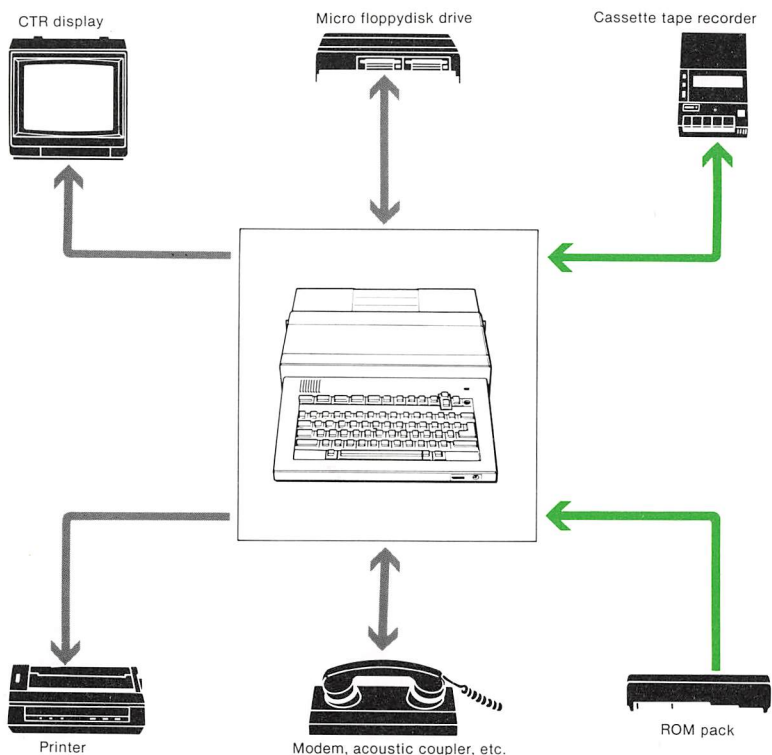
We call the unit of a BASIC program or data a "file". BASIC accesses the file and writes or reads it.

A file of a BASIC program is called a "program file" and a file of data is called a "data file".

FILE DEVICES

Sony disk BASIC saves a program onto a disk and loads the program from the disk. A device which BASIC accesses for the file write/read operation is called a "file device".

The following are the file devices that Sony disk BASIC can handle.



The file device to be used is specified by a “device name” in the BASIC commands. The file devices and their names are as follows :

File device	Device name	Accessibility	
		Write	Read
Floppydisk drive	DSK :	○	○
Cassette tape recorder	CTR :	○	○
ROM pack	PAC :	×	○
Console device (CRT, keyboard)	CON :	○	○
List device (printer connected to the PRINTER, PARALLEL)	LST :	○	×
Channel device (modem connected to RS- 232C connector)	CHN :	○	○

○ : yes

× : no

NAMING OF FILES

Sony disk BASIC handles a program or data using a file name and a type specification when saving or loading files.

The file name can be defined with eight or fewer alphanumeric characters beginning with an alphabetic character, and the type specification with up to three characters.

The file name and type specification are separated by a “.”. The type specification is available so that the category of file (for example, whether it is a program file or a data file, or whether it has been saved in the ASCII format) can be referred to later.

The type specification can be omitted if desired.

PROGRAM FILE

To save or load BASIC programs, use the SAVE, LOAD, CSAVE or CLOAD command.

The device name and the file name are specified after the command. If the device name is omitted, the cassette tape recorder will be specified with the CSAVE or CLOAD command and the floppydisk drive will be specified with the SAVE or LOAD command.

To save a program, execute a SAVE or CSAVE statement specifying a file name. The program currently in the memory will be saved onto the specified device with the file name.

If a file of the same file name as that specified in a SAVE command exists on the disk, the old file will be erased and the new file saved.

To load the program, execute a LOAD or CLOAD command specifying the file name and type which were used when the program was saved.

You can also use the RUN command to load a program and run it, the MERGE command to merge a program on a device and the program in memory and the CHAIN command to run a program on a device after the execution of the program in memory.

DATA FILE

There are two types of data files ; sequential access files and random access files. A random access file can be handled only on a disk drive and a ROM pack.

Sequential access file

Data are written sequentially into a sequential access file. The data of the file can only be read in the same sequence as they have been written.

To write data,

- ① Open a file.
- ② Write data.
- ③ Close a file.

Let us now explain each of these operations in detail.

❶ Open a file.

Execute

OPEN/O # file number, "device name: file name", buffer length
A file with the specified file name is opened on the specified device. Suppose that the computer has 128 boxes (called "buffers" in many versions of BASIC). "File number", then, specifies the number of the box into which data is to be put and "buffer length" specifies how many bytes of data (up to XXXX) can be put into that box. The buffer length is measured in a multiple of 128 bytes. Any bytes over XXXX will cause the XXXX bytes in the box to be written into the file specified by "file name" in the device specified by "device name" and the excess bytes to be put into the box.

❷ Write data.

Execute

PRINT #file number, variable separator variable separator...
The values of the variables in the PRINT # statement will be written into the specified file. The file number in the PRINT # statement must be the same as that specified in the OPEN/O statement.
The data will be written in the same format as that of the display generated by the PRINT statement. When the data are separated by semicolons ; data are written without any separator inserted between them. When a comma is used, a tab code (&H09) is added after a datum and when no separator is inserted, return code (&H0D) and line feed code(&H0A) are added after a datum. When the output format is specified with USING, the data are written into a file in the same format as that displayed on the screen by the PRINT USING statement.

❸ Close the file.

Execute

CLOSE # file number
The computer clears the relation between the file and the file number.

To read data,

- ❶ Open a file.
- ❷ Read data.
- ❸ Close the file.

Now let us explain these operations in detail.

❶ Open a file.

Execute

OPEN/I # file number, "device name : file name", buffer length

The computer searches for the specified file on the specified device.

When the file is found, the computer prepares the file to be read.

Again, suppose the computer has 128 boxes numbered 0 to 127.

The number of bytes specified in "buffer length" are taken into the box specified by "file number" and the data are read sequentially from the first datum.

When the data in the box have been all read, the next length of bytes are taken into the box and read.

❷ Read data.

Execute

INPUT # file number, variable, variable, ...

The INPUT # statement reads data in the same way as an INPUT statement which reads data from the keyboard. When several data are read by an INPUT #, commas must be written between data to be read.

An INPUT # statement regards a return code as a separator as well as a comma.

When the read data are assigned to the variables, they are converted to the type of the variables. When a string data are to be assigned to a numeric variable, an error will occur.

INPUT/L # can also be used to read string data from the file.

Execute

INPUT/L # file number, variable, variable, ...

The variable in this statement must be of the string type.

All characters up to the first return code are assumed to be a single string datum.

A comma is assumed to be one character of a datum.

❸ Close the file.

CLOSE # file number

The computer clears the relation between the file and the file number.

Example

```
10 OPEN /O #1,"NAME"  
20 FOR I=1 TO 5  
30 READ A$  
40 PRINT #1,A$;" , ";  
50 NEXT I  
60 CLOSE #1  
70 DATA NEW YORK,LOS ANGELES,  
    SAN FRANCISCO, CHICAGO, DALLAS
```

When this program is executed, a file named "NAME" is opened and a box numbered "1" is prepared. The data written by the PRINT # statement are taken into the box. That is, the string data, "NEW YORK", ",", "LOS ANGELES", ",", are taken into the box. When 128 bytes of data have been taken into the box or when a CLOSE statement has been executed, the data are written into the file named "NAME" on the disk.

The data are written on the disk as follows :

```
NEW YORK, LOS ANGELES, SAN FRANCISCO, CHICAGO,  
DALLAS,
```

The following program reads data from the file "NAME".

```
10 DIM A$(4)  
20 OPEN /I #2, "NAME"  
30 FOR I=0 TO 4  
40 INPUT #2,A$(I)  
50 PRINT A$(I)  
60 NEXT I  
70 CLOSE #2
```

When this program is executed, the file "NAME" is opened and the data read are assigned to the variables. As a comma or a carriage return code is assumed to be the separator by the INPUT # statement, the string data "NEW YORK", "LOS ANGELES", "SAN FRANCISCO", "CHICAGO" and "DALLAS" are assigned to the variables A\$(0) to A\$(4).

As an INPUT/L # does not assume a comma as a separator, a return code must have been written to the end of a datum. Insert a line

```
55 PRINT #1
```

to the first program on the previous page. A return code will be added after the last comma and the string "NEW YORK, LOS ANGELES, SAN FRANCISCO, CHICAGO, DALLAS," will be read as a datum.

```
10 OPEN /I #2, "NAME"  
20 INPUT /L #2,A$  
30 PRINT A$  
40 CLOSE #2
```

Notes

- A file number specified in an OPEN/I(or/O) statement can not be specified in another OPEN/I(or/O) statement at the same time.

```
OPEN /O #1,"TEST1.DAT"
```

```
OPEN /I #1,"TEST2.DAT"
```

```
Error(153) : File already open
```

- The OPEN/S and OPEN/R commands also open a sequential file. OPEN/O and OPEN/I commands open a sequential file and data are written/read in the ASCII format by a PRINT # statement, and INPUT # or INPUT/L # statement. On the other hand, OPEN/S and OPEN/R commands open a sequential file and data are written/read in the Sony BASIC internal format¹⁾ by SEND and RECEIVE statements. If you attempt to read a file whose data have been written by PRINT # statements by executing a RECEIVE statement, invalid data will be read.

A numeric datum allocates fewer bytes when it is written or read by executing SEND or RECEIVE command. These commands are useful for transferring data between two SMC-70s or when using the RS-232C standard interface.

¹⁾ See Appendix 3 for the internal format of a datum.

Random access file

The data are randomly written into or read from a random access file.

- ① Open a file.
- ② Allocate one record.
- ③ Write/read data.
- ④ Close the file.

Now let's explain these operations in more detail.

- ① Open a file.

Execute

OPEN # file number, "device name ; file name", record length
If the specified file exists on the specified device, it is opened for the data write/read operation. If the specified file does not exist, a new file is opened.

Suppose the computer has boxes numbered 0 to 127. The "file number" specifies which box is to be used and the "record length" specifies how many bytes are to be written or read at one time.

- ② Allocate one record.

Execute

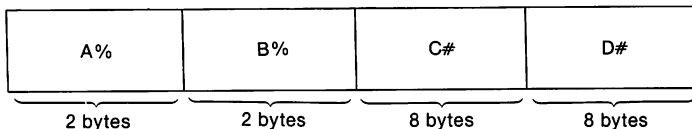
RECORD record allocation

This statement declares the variables to be used and allocates a number of bytes to each variable. For numeric variables, an integer value uses 2 bytes and a floating point value uses 8 bytes. For a string value, you must specify how many bytes are to be used.

For example, if integer variables **A%** and **B%** and floating point variables **C#** and **D#** are to be used, execute

RECORD #1 ,A% ,B% ,C# ,D#

The record will be allocated as follow :



In the above case, the record length must have been specified as at least 20 bytes in the **OPEN** statement.

For string data, the number of bytes and whether the data are to be written with right-justification or left-justification must be specified. For example, execution of

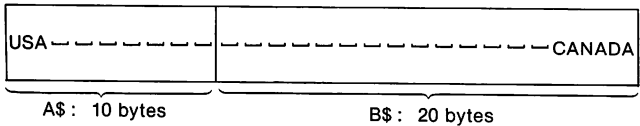
RECORD #1 ,LJUST 10 AS A\$,RJUST 20 AS B\$

allocates 10 bytes for A\$ and 20 bytes for B\$. The value of A\$ will be written with left justification and the value of B\$ is with right-justification. If

A\$=" USA "

B\$=" CANADA "

are executed, the values will be written in box "1" as follows ;

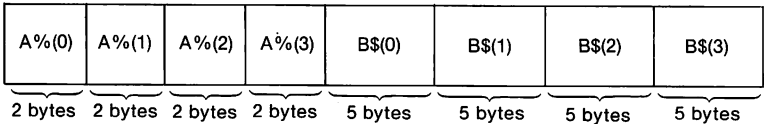


If no value has been assigned to a variable, 0 will be written if the variable is of the numeric type and a null string will be written if it is of the string type.

If an array variable is written in a RECORD statement, the array is declared. For example, execution of

RECORD #1 ,A%(3),LJUST 5 AS B\$(3)

allocates the record as ;



③ Write/read data.

Execute

PUT # file number, record number

The data currently in the specified box will be written into the file with the specified record number. If the record number is omitted, the record number is specified as 0,1,2 as PUT statements are executed. When

GET # file number, record number

is executed, the data of the record specified by the record number are taken into the box.

④ Close the file.

Execute

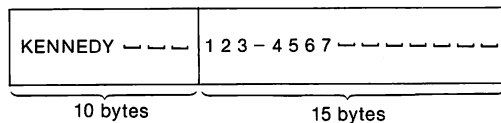
CLOSE # file number

The computer clears the relation between the file and the file number.

Example

```
10 OPEN #1, "TELE.DAT"
20 RECORD #1, LJUST 10 AS NAME$,LJUST 15 AS NO$
30 FOR I=1 TO 5
40 READ NAME$, NO$
50 PUT #1,I
60 NEXT I
70 CLOSE #1
80 DATA KENNEDY, 123-4567
90 DATA TAYLOR, 345-6789
100 DATA JONES, 333-2345
110 DATA PETERS, 567-8901
120 DATA HORTON, 789-1234
```

When this program is executed, a file named "TELE.DAT" is opened. One record is specified as 128 bytes, and 10 bytes are allocated for the value of NAME\$ and 15 bytes are allocated for the value of NO\$. In line 40, the data "KENNEDY" is assigned to the value of NAME\$ and "123-4567" is assigned to the value of NO\$.



In line 50, the PUT statement writes these data into the file "TELE.DAT" with the record number 1. The control returns to the line 30. Again, the next data of the name and the telephone number are assigned to the variables and these data are written into the file with the record number 2. The sequence is repeated and the name and the telephone number for five people are written into the file with the record numbers 1 to 5.

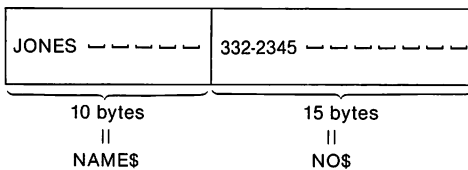
To read the name and the telephone number of the third and fifth person in this file, execute the following program.

```

10 OPEN #1,"TELE.DAT"
20 RECORD #1, LJUST 10 AS NAME$,LJUST 15 AS NO$
30 INPUT "Record number?",N$
40 IF N$="" THEN CLOSE #1:END
50 GET #1, VAL (N$)
60 CCOLOR 3:PRINT NAME$,NO$:CCOLOR 7
70 GOTO 30

```

In line 10, the file "TELE.DAT" is found on the disk and is opened. One record is allocated as 128 bytes, and 10 bytes are for the value of NAME\$ and 15 bytes are for the value of NO\$. In line 30, the computer displays "Record number?" At that point enter the record number of the data you desire. To get the data of the third person, press . The data with the record number 3 will be put into box 1 and assigned to the variables.



In line 60, the read data are displayed. Again, the computer displays "Record number?", so enter the desired record number.

If you do not need any more data, simply press . The opened file will be closed.

In order to change the contents of a file, execute a PUT statement specifying the record number after the value of the variables have been changed.

CHAPTER 3

PROGRAM

EXAMPLES

PROGRAM TO GENERATE A BAR GRAPH

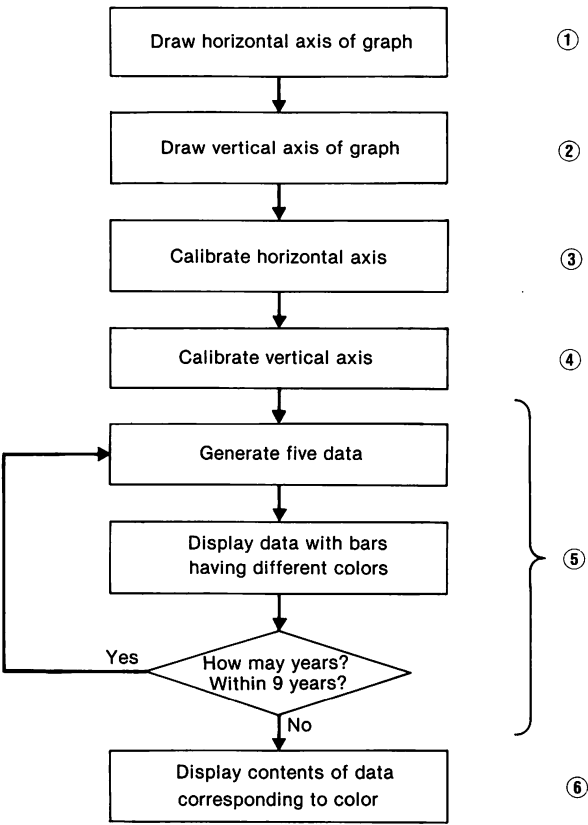
```

10  CONSOLE 80:WIFE:CURSOR OFF
20  CCOLOR 3:PRINT "(m.#)";
30  CCOLOR 5:PRINT TAB(25);"SALES FORECAST
    1981-1989"
40  LINE (20,0)-(20,180),7
50  FOR Y=180 TO 0 STEP -24
60      LINE(20,Y)-(319,Y),7
70  NEXT Y
80  FOR X=1 TO 319 STEP 2
90      LINE(X,0)-(X,170),0
100 NEXT X
110 CCOLOR 2
120 FOR I=1 TO 9
130     LOCATE(I*8+1,23):PRINT 8;I
140 NEXT I
150 LOCATE(73,24):PRINT "(YEAR)";
160 LOCATE(0,1):CCOLOR 3
170 FOR I=7 TO 1 STEP -1
180     PRINT I*100:PRINT :PRINT
190 NEXT I
200 PRINT "    0"
210 FOR I=1 TO 9
220     S=0
230     FOR K=1 TO 5
240         A(K)=A(K)+RND*5
250         Y=180-S-A(K)
260         IF Y>180 THEN Y=180
270         BOXF(I*32,180-S)-(I*32+13,Y),6-K
280         S=S+A(K)
290     NEXT K
300 NEXT I
310 LINE(70,36)-(125,36),0
320 CCOLOR 7
330 FOR I=1 TO 5
340     BOXF(30,I*8+8)-(70,I*8+14),I
350     LOCATE(20,I+1):PRINT "PRODUCT ";I
360 NEXT I

```

Flowchart

(Number in program example)



PROGRAM TO GENERATE 3-DIMENSIONAL BAR GRAPH

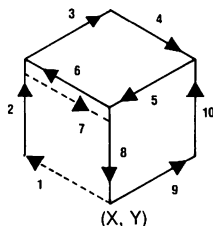
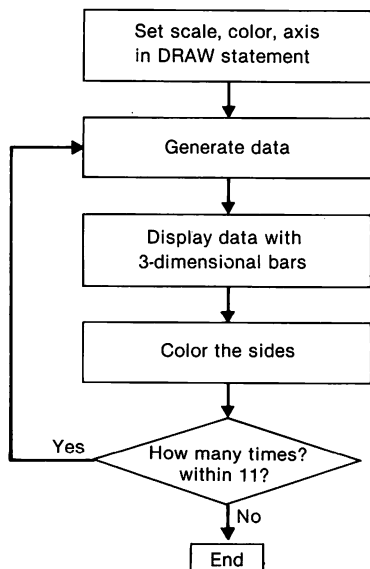
```

10  CONSOLE 80
20  RANDOMIZE
30  WIPE:CURSOR OFF
40  Q1$="W 20,-5"
50  Q2$="W-20, 5"
60  Q3$="W 20, 5"
70  Q4$="W-20,-5"
80  DRAW "Z0S1C3"
90  FOR X=40 TO 240 STEP 20
100   H=INT(RND*13+3)
110   H=H+HM:IF H>120 THEN H=120
120   Y=190-.25*X
130   IF X=40 THEN DRAW "m(X),(Y)[Q4$]"
140   DRAW "m(X),(Y)w-20,-5U(H)[Q1$][Q3$]"
      [Q2$][Q4$][Q3$]D(H)[Q1$]U(H)"
150   PAINT (X+3,Y-3),15,3
160   PAINT (X,Y-H-5),3,3
170   PAINT (X-5,Y-HM-2),13,3
180   HM=H
190 NEXT X

```

Flowchart

(Number in the program)



PROGRAM TO DRAW A PI CHART

```

10 GCOLOR 0,2,0:CONSOLE 40
20 CCLEAR 0:WIPE:CURSOR OFF
30 CENTER$="m160,100"
40 R=80
50 DRAW "Z1S1A0"
60 DRAW "[CENTER$]O(R)"
70 READ MAX
80 FOR I=1 TO MAX
90   READ DT(I)
100  SUM=SUM+DT(I)
110 NEXT I
120 J=0
130 DRAW "A0U(R)"
140 FOR I=1 TO MAX
150   J=J+DT(I)
160   A=-J/SUM*360
170   C=I+8
180   DRAW "[CENTER$]A(A)U(R)w-3,3P(C),7"
190   BOXF(11,I*8+1)-(39,I*8+5),C
200   PRINT (6,I);"ITEM";I
210 NEXT I
220 EVAL INKEY$(1)
230 DATA 7
240 DATA 2518,1352,1026,415,403,286,252
    
```

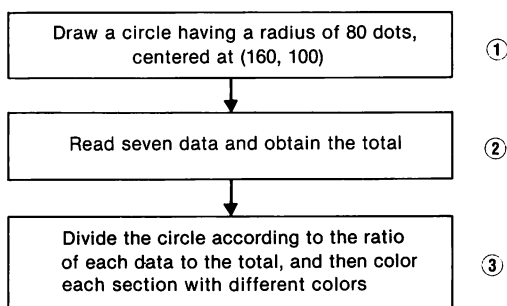
①

②

③

Flowchart

(Number in program example)



PROGRAM TO DRAW A PICTURE FROM THE KEYBOARD

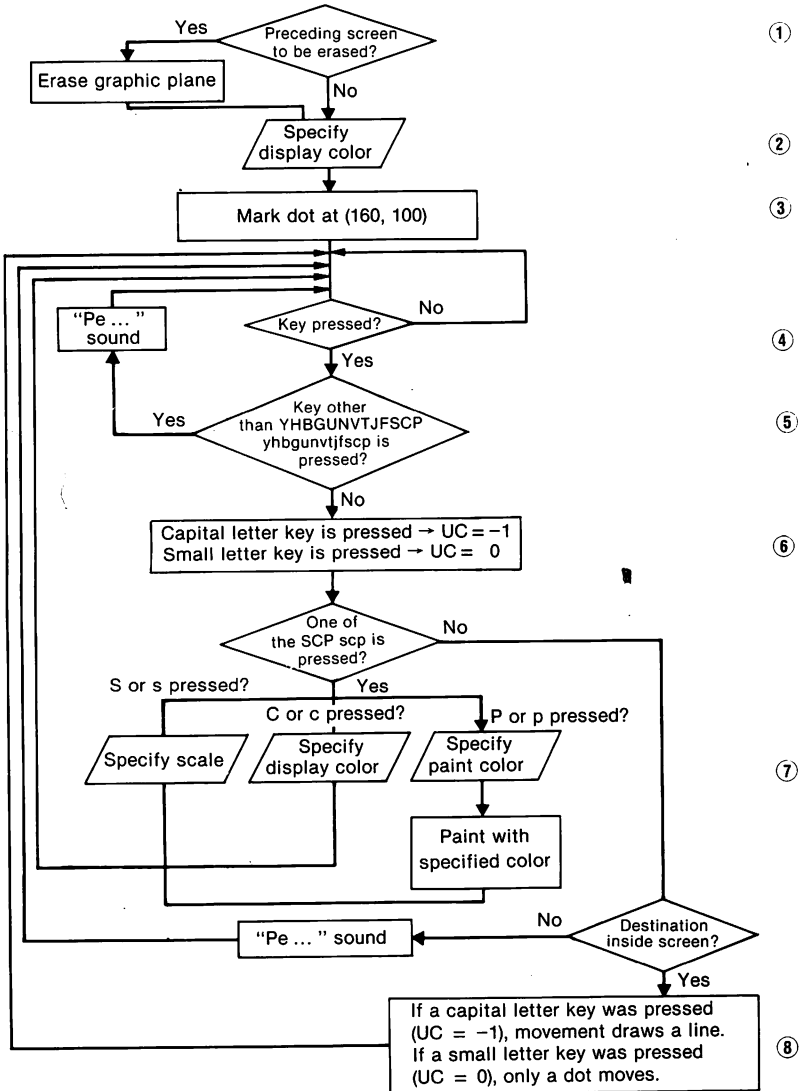
```

10 CCLEAR
20 CONSOLE 80,22,3,1
30 PRINT (0,0);"Y H B G U N V T J F S
C P";(0,22)
40 INPUT "Erase Picture (Y/N)";A$
50 IF A$="y" OR A$="Y" THEN GCLEAR
60 CCLEAR 22,3
70 DIM D$(10)
80 X$="YHBGUNVTJFSCP"
90 FOR I=1 TO 10
100 READ A$
110 D$(I)=A$
120 NEXT
130 GOSUB 280
140 X=160:Y=100:S=1
150 DRAW "C(C)S(S)m(X),(Y)W0,0"
160 /
170 Y$=INKEY$(1):IF Y$="" GOTO 170
180 A=INSTR(X$,Y$):IF A=0 THEN
BEEP:GOTO 170
190 UC=A<=13:IF NOT UC THEN A=A-13
200 IF A>10 THEN ON A-10 GOSUB 270,280,
290:GOTO 170
210 IF UC THEN A$="C(C)W"+D$(A) ELSE
A$="C0W0,0w"+D$(A)+"C(C)W0,0"
220 DX=VAL(LEFT$(D$(A),2)):X=X+DX*S
230 DY=VAL(RIGHT$(D$(A),2)):Y=Y+DY*S
240 IF X>=0 AND X<320 AND Y>=0 AND
Y<200 THEN DRAW A$:GOTO 170
250 X=X-DX*S:Y=Y-DY*S:BEEP:GOTO 170
260 /
270 PRM$="scale(1- )":GOSUB 310:S=V:
DRAW "S(S)":RETURN
280 PRM$="color(0-15)":GOSUB 310:C=V:
RETURN
290 PRM$="color(0-15)":GOSUB 310:DRAW
"C0W0,0P(V),(C)C(C)":RETURN
300 /
310 LOCATE (0,24):CURSOR ON :PRINT PRM$:
320 INPUT V:CURSOR OFF
330 RETURN
340 DATA " 0,-1"," 1, 0"," 0, 1","-1,
0"," 1,-1"," 1, 1","-1, 1","-1,-1"," 1,
0","-1, 0"

```

Flowchart

(Number in the program)



PROGRAM TO GENERATE AN ADDRESS LIST

```

100 /
110 / Address writer
120 /
130 DEF INT A-Z
140 DEF FNSKIPSP$(A$)=COND(LEFT$(A$,5)
="      ",FNSKIPSP$(RIGHT$(A$,LEN(A$)-5))
,COND(LEFT$(A$,1)=" ",FNSKIPSP$(RIGHT$(A
$,LEN(A$)-1)),A$))
150 DEF FNDELSP$(A$)=COND(RIGHT$(A$,5)
="      ",FNDELSP$(LEFT$(A$,LEN(A$)-5)),C
OND(RIGHT$(A$,1)=" ",FNDELSP$(LEFT$(A$,L
EN(A$)-1)),A$))
160 ON ERROR GOTO *ERRORCOR
170 ERROR OFF
180 *TIMEENT
190 WIPE
200 CONSOLE 40
210 CCOLOR 2
220 PRINT "      ";DATE$(2);" ";DAY$;" ";
TIME$(1);" OK(Y/N):";
230 EVAL INKEY$
240 IF INSTR("Nn",INKEY$(1))=0 THEN PR
INT "Yes";PAUSE 5;GOTO *OPENFILE
250 PRINT "No"
260 *REDEF
270 CCOLOR 6
280 PRINT "Input the date as follows
:":PRINT "YY/MM/DD DAY HH:MM:SS"
290 CCOLOR 7
300 INPUT ,T$:T$=FNSKIPSP$(T$)
310 IF T$="" GOTO *TIMEENT
320 D1$=MID$(T$,4,2)+MID$(T$,7,2)+LE
FT$(T$,2)
330 D2$=MID$(T$,14,2)+MID$(T$,17,2)+
RIGHT$(T$,2)
340 D3$=MID$(T$,10,3)
350 ERROR ON
360 SET DATE D1$,D2$,D3$
370 ERROR OFF
380 /
390 *OPENFILE
400 OPEN #1,"address.dat"

```

```

410 RECORD #1,LJUST 30 AS NAME$,LJUST
20 AS TEL$,LJUST 75 AS ADR$,RJUST 3 AS A
GE$
420 *SELECT
430 WIPE
440 CONSOLE 40,1,24
450 CCOLOR 7
460 X=CSRLCM:Y=CSRLIN
470 PRINT (0,0);"** ADDRESS /Mode Sel
ect **"
480 PRINT
490 CCOLOR 6
500 PRINT "          New data:"
510 PRINT "          Update  :"
520 PRINT "          Quit    :"
530 CCOLOR 2
540 PRINT "Select (N/U/Q):";
550 EVAL INKEY$
560 IF SETQ(0,INSTR("NnUuQq",INKEY$(1)
)/2)=0 THEN BEEP:GOTO 560
570 ON 0 GOTO *NEWDATA,*UPDATE,*QUIT
580 /
590 *NEWDATA
600 PRINT "New data":PAUSE 5
610 WIPE
620 CONSOLE 40,1,24,1
630 CCOLOR 7
640 PRINT (0,0);"** ADDRESS /Mode:New
data **"
650 GET #1,0
660 IF FNDELSP$(NAME$)<>"$record-numbe
r" THEN RECNO=1 ELSE RECNO=VAL(TEL$)
670 PRINT
680 CCOLOR 6:PRINT "Record No.":RECNO;
690 CCOLOR 2:PRINT "  Input new data (
Y/N):":
700 WHILE INSTR("Nn",INKEY$(1))=0
710   PRINT "Yes"
720   GOSUB *DATENT
730   PUT #1,RECNO
740   RECNO=RECNO+1
750   CCOLOR 6:PRINT "Record No.":RECNO;
0;
760   CCOLOR 2:PRINT "  Input new data
(Y/N):";
770 WEND

```

```

780 PRINT "No"
790 PRINT
800 NAME$="$record-number"
810 TEL$=STR$(RECNO)
820 ADR$=DATE$(2)+" "+DAY$+" "+TIME$(1
)
830 PUT #1,0
840 GOTO *SELECT
850 /
860 *UPDATE
870 PRINT "Update":PAUSE 5
880 WIPE
890 CONSOLE 40,2,23,1
900 CCOLOR 7
910 PRINT (0,0);"** ADDRESS /Mode:Upd
ate **"
920 GET #1,0
930 CCOLOR 7
940 IF FNDELSP$(NAME$)="$record-number
" GOTO 990
950 CCOLOR 4:PRINT "There is no data
in this file."
960 CCOLOR 2:PRINT "Press any key to
continue.";
970 EVAL INKEY$
980 EVAL INKEY$(1):GOTO *SELECT
990 PRINT "Last data: ";FNDELSP$(ADR$)
1000 LASTREC=VAL(TEL$)
1010 PRINT
1020 RECNO=1:LINEC=0
1030 WHILE RECNO<LASTREC
1040 CCOLOR 6
1050 PRINT USING "Record :####";RE
CNO
1060 GET #1,RECNO
1070 CCOLOR 3
1080 CCOLOR 3:PRINT "Name :";
1090 CCOLOR 7:PRINT FNDELSP$(NAME$)
1100 CCOLOR 3:PRINT "Age :";
1110 CCOLOR 7:PRINT FNSKIPSP$(FNDELSP
$(AGE$))
1120 CCOLOR 3:PRINT "Address :";
1130 CCOLOR 7:PRINT FNDELSP$(ADR$)
1140 CCOLOR 3:PRINT "Telephone: ";
1150 CCOLOR 7:PRINT FNDELSP$(TEL$)
1160 IF LINEC<3 AND RECNO<(LASTREC-1)
THEN LINEC=LINEC+1:GOTO 1270

```

```

1170 *REENT;CCOLOR 2:INPUT /L "Update
record NO.:",A#
1180 IF A#="" THEN LINEC=0:GOTO 1270
1190 IF INSTR("Qq",LEFT$(A#,1)) THEN
RECNO=LASTREC:GOTO 1280
1200 ERROR ON
1210 RECNO0=VAL(A#)
1220 ERROR OFF
1230 IF RECNO0<1 OR LASTREC<=RECNO0
THEN BEEP:GOTO *REENT
1240 GOSUB *DATENT
1250 PUT #1,RECNO0
1260 GOTO *REENT
1270 RECNO=RECNO+1
1280 WEND
1290 CCOLOR 2
1300 CCOLOR 4:PRINT "** Update END **"
1310 CCOLOR 2:PRINT "Press any key to c
ontinue.";
1320 EVAL INKEY#
1330 EVAL INKEY$(1)
1340 GOTO *SELECT
1350 /
1360 *QUIT
1370 PRINT "Quit"
1380 CLOSE
1390 END
1400 /
1410 *DATENT
1420 CCOLOR 3:PRINT "Name :";
1430 CCOLOR 7:INPUT /L,NAME#
1440 CCOLOR 3:PRINT "Age :";
1450 CCOLOR 7:INPUT /L,AGE#
1460 CCOLOR 3:PRINT "Address :";
1470 CCOLOR 7:INPUT /L,ADR#
1480 CCOLOR 3:PRINT "Telephone:";
1490 CCOLOR 7:INPUT /L,TEL#
1500 NAME#=FNSKIPSP$(NAME#)
1510 TEL#=FNSKIPSP$(TEL#)
1520 ADR#=FNSKIPSP$(ADR#)
1530 AGE#=RIGHT$(" "+FNSKIPSP$(AGE#)
,3)
1540 RETURN
1550 *ERRORCOR
1560 IF ERR=3 THEN BEEP:RESUME *REDEF
1570 IF ERR=6 THEN BEEP:RESUME *REENT
1580 STOP

```

PROGRAM TO READ AN ADDRESS LIST

```

100 /
110 / Address read
120 /
130 ON ERROR GOTO *SUBERR
140 DEF INT A-Z
150 DEF FNSKIPSP$(A$)=COND(LEFT$(A$,5)
="      ",FNSKIPSP$(RIGHT$(A$,LEN(A$)-5))
,COND(LEFT$(A$,1)=" ",FNSKIPSP$(RIGHT$(A
$,LEN(A$)-1))),A$))
160 DEF FNDELSP$(A$)=COND(RIGHT$(A$,5)
="      ",FNDELSP$(LEFT$(A$,LEN(A$)-5)),C
OND(RIGHT$(A$,1)=" ",FNDELSP$(LEFT$(A$,L
EN(A$)-1))),A$))
170 /
180 *OPENFILE
190 OPEN #1,"address.dat"
200 RECORD #1,LJUST 30 AS NAME$,LJUST
20 AS TEL$,LJUST 75 AS ADR$,RJUST 3 AS A
GE$
210 GET #1,0
220 IF FNDELSP$(NAME$)<>"$record-numbe
r" GOTO *NOFILE
230 LASTREC=VAL(TEL$)
240 DIM WORK$(LASTREC-2),WORK(LASTREC-
2)
250 SKEY=0
260 DA$=FNDELSP$(ADR$)
270 *SELECT
280 WIPE
290 CONSOLE 40,2,23
300 CCOLOR 7
310 X=CSRCLM:Y=CSRLIN
320 PRINT (0,0);"** ADDRESS /Mode Sel
ect **"
330 PRINT "Last data : ";DA$
340 PRINT
350 CCOLOR 6
360 PRINT "          Sort   ;"
370 PRINT "          sEarch:"
380 PRINT "          Quit   ;"
390 CCOLOR 2

```

```

400 PRINT "Select (S/E/Q):";
410 EVAL INKEY$
420 IF SETQ(0, INSTR("SsEeQq", INKEY$(1)
)/2)=0 THEN BEEP:GOTO 420
430 ON 0 GOTO *SORT,*SEARCH,*QUIT
440 /
450 *SORT
460 PRINT "Sort":PAUSE 5
470 WIPE
480 CONSOLE 40,2,23,1
490 CCOLOR 7
500 PRINT (0,0);"** ADDRESS /Mode:Sort
data **"
510 PRINT "Last data : ";DA$
520 PRINT
530 CCOLOR 6
540 PRINT (16,CSRLIN);"Name      : "
550 PRINT (16,CSRLIN);"Age       : "
560 PRINT (16,CSRLIN);"aDdress   : "
570 PRINT (16,CSRLIN);"Telephone:"
580 CCOLOR 2
590 PRINT "Select sort key (N/A/D/T):"
;
600 EVAL INKEY$
610 IF SETQ(SKEY, INSTR("NnTtDdAa", INKEY$
(1))/2)=0 THEN BEEP:GOTO 610
620 PRINT MID$("NameTeleaDdrAge ", SKEY
*4-3,4)
630 RECNO=1
640 WHILE RECNO<LASTREC
650   GET #1,RECNO
660   D$(0)=NAME$
670   D$(1)=TEL$
680   D$(2)=ADR$
690   D$(3)=AGE$
700   WORK$(RECNO-1)=FNDELSP$(D$(SKEY-
1))
710   WORK$(RECNO-1)=RECNO
720   RECNO=RECNO+1
730 WEND
740 FOR I=LASTREC-2 TO 0 STEP -1
750   FOR J=0 TO I-1

```

```

760 IF WORK$(J)>WORK$(J+1) THEN SWAP
WORK$(J),WORK$(J+1):SWAP WORK(J),WORK(J
+1)
770 NEXT
780 NEXT
790 CCOLOR 3
800 FOR I=0 TO LASTREC-2
810 CCOLOR 3
820 PRINT USING "Record #####:";WORK(
I);
830 CCOLOR 7
840 PRINT WORK$(I)
850 NEXT
860 GOTO *CONTINUE
870 /
880 *SEARCH
890 PRINT "Search":PAUSE 5
900 WIPE
910 CONSOLE 40,3,22,1
920 CCOLOR 7
930 PRINT (0,0);"** ADDRESS /Mode:Loo
k up **"
940 CCOLOR 7
950 IF SKEY GOTO 980
960 CCOLOR 4:PRINT "Can't search wit
hout SORT"
970 GOTO *CONTINUE
980 PRINT "Last data : ";DA$
990 PRINT "Search Key:";MID$("Name
TelephoneAddress Age",SKEY*9-8,9)
1000 CCOLOR 2
1010 PRINT "Enter key word:";
1020 CCOLOR 7
1030 INPUT ,EK$:IF SKEY=4 THEN EK$=RIGH
T$(" "+EK$,3)
1040 COUNT=0
1050 ERROR ON
1060 WHILE EK$>=WORK$(COUNT)
1070 IF EK$<>WORK$(COUNT) GOTO 1180
1080 GET #1,WORK(COUNT)
1090 CCOLOR 6:PRINT USING "Record
:#####";WORK(COUNT)
1100 CCOLOR 3:PRINT "Name :";

```

```

1110      CCOLOR 7:PRINT FNDELSP$(NAME$)
1120      CCOLOR 3:PRINT "Age      ";
1130      CCOLOR 7:PRINT FNSKIPSP$(FNDELSP$(AGE$))
1140      CCOLOR 3:PRINT "Address  ";
1150      CCOLOR 7:PRINT FNDELSP$(ADR$)
1160      CCOLOR 3:PRINT "Telephone:";
1170      CCOLOR 7:PRINT FNDELSP$(TEL$)
1180      COUNT=COUNT+1
1190  WEND
1200  *NOHIT
1210  ERROR OFF
1220  CCOLOR 4:PRINT "** Search END **"
1230  GOTO *CONTINUE
1240  /
1250  *QUIT
1260  PRINT "Quit"
1270  CLOSE
1280  END
1290  /
1300  *NOFILE
1310      CCOLOR 4:PRINT "There is no data
in this file."
1320      CCOLOR 2:PRINT "Create data by t
he Address writer."
1330      END
1340  *CONTINUE
1350      CCOLOR 2:PRINT "Press any key to
continue.";
1360      EVAL INKEY$
1370      EVAL INKEY$(1)
1380      GOTO *SELECT
1390  *SUBERR
1400  RESUME *NOHIT

```



APPENDIX

1. KEYS TO PROGRAMMING

It is important to write a program that can be easily understood, modified, and is not susceptible to errors. Keys to writing such a program are as follows :

- Insert a space at the beginning of a line (indentation) to facilitate finding loops.

Write a variable after NEXT to clarify the correspondence between FOR and NEXT.

- Insert a REM statement or assign a label so that program units are clarified and the values to be received and the processing to be performed by the subroutine are specified.
- Reduce the number of GOTO statements.
- Assign variable names so that the meaning of the data can be easily understood. Make it clear whether a variable value changes in a program.

This facilitates debugging and addition of lines. If it is necessary to reduce program execution time and to save the memory, improve the entire algorithm or, in some cases, modify the existing program. Concrete examples are as follows : (Modifying as follows may make the program complex and hard to be understood.)

Reduce the number of operations for an expression

To compute

$$A=3*B+3*C+3*D$$

five operations are required. If this expression is rewritten as

$$A=3*(B+C+D)$$

only three operations will be required, and the computation speed is accelerated.

The computation speed of multiplication is faster than that of an exponential operation. For example,

$$Y=AX^2 + BX + C$$

can be computed in three different ways :

- ① $Y=A*X^2+B*X+C$
- ② $Y=A*X*X+B*X+C$
- ③ $Y=X*(A*X+B)+C$

The number of operations for these expressions are as follows :

① 5

② 5

③ 4

The computation speed of ③ is the fastest.

The numbers of operations of ① and ② are the same ; however, the computation speed of ② is faster than that of ① because ① includes a power operation. On the other hand, the speed of addition and subtraction is not the same as that of multiplication and division, and expressions including parentheses take more time. Computation speeds can not be judged simply from the number of operations ; however, the above mentioned definitions apply in most cases.

Use integer data instead of floating point data

A floating point datum uses eight bytes of the memory. An integer datum uses two bytes. Therefore, if only integers are handled, memory area can be saved by using integer variables. The speed of addition/subtraction/multiplication/division of floating point data is slower, and interpretation of the floating point expression and storage of the interpretation result take more time because the data length is longer.

Use of integer data reduces operation time.

Reduce the number of type conversions

When

`A#=A#*2`

is executed, 2 is converted to the floating point type and then calculated. If

`A#=A#*2.`

is executed, the operation speed will be accelerated even though more memory area is required.

Reduce the variable name length

When short variable names are used, variable names in the memory can be referenced more rapidly, and program execution speed is accelerated.

Omit unnecessary spaces and tabs

The execution speed becomes slightly faster and memory can be slightly saved.

Reduce the number of GOTO statement

When a GOTO statement is executed, line numbers in a program are sequentially searched from the lowest line number in order to find the called line number. This takes a long time if many GOTO statements are executed in a program. When GOTO statements are used, write the line number to be called as near as possible to the beginning of a program to reduce the search time. If a GOSUB statement is used, it will take time to find the called line number. However, the program will be reduced in size and become legible, and memory can be saved.

Reduce the number of array variables

When an array variable is used, the subscript value is obtained to determine where it is in the array. Then data is referenced, and the data reference time takes longer compared with an ordinary variable.

Write multiple statements separated by colons in a line

Each program line requires five bytes (including two bytes for the line number) in addition to the memory area for statements proper. If as many statements as possible are arranged in a line, memory can be saved, and execution speed can be accelerated.

Omit the REM statement

If the REM statement is omitted, execution speed will become faster, and memory can be saved.

If the REM statement describing a subroutine is specified outside the subroutine so that the REM statement is not executed, program execution speed will be accelerated.

Use a machine language subroutine

If a program is written in a machine language subroutine and is called from Sony disk BASIC, execution time will be reduced.

2. TRIGONOMETRIC AND HYPERBOLIC FUNCTIONS

Values of trigonometric and hyperbolic functions other than the functions incorporated into Sony disk BASIC are obtained as follows :

Function	Operational expression
Secant	$1/\text{COS}(X)$
Cosecant	$1/\text{SIN}(X)$
Cotangent	$1/\text{TAN}(X)$
Arcsine	$\text{ATN}(X/\text{SQR}(1-X^2))$
Arccosine	$-\text{ATN}(X/\text{SQR}(1-X^2)) + \text{PI}/2$
Arcsecant	$\text{ATN}(\text{SQR}(X^2-1)) + (\text{SGN}(X)-1)*\text{PI}/2$
Arccosecant	$\text{ATN}(1/\text{SQR}(X^2-1)) + (\text{SGN}(X)-1)*\text{PI}/2$
Arccotangent	$-\text{ATN}(X) + \pi/2$
Hyperbolic sine	$(\text{EXP}(X) - \text{EXP}(-X))/2$
Hyperbolic cosine	$(\text{EXP}(X) + \text{EXP}(-X))/2$
Hyperbolic tangent	$-\text{EXP}(-X)/(\text{EXP}(X) + \text{EXP}(-X))*2 + 1$

3. MEMORY CONFIGURATION

When BASIC is executed in the CP/M prompt mode, Sony disk BASIC interpreter is moved to the TPA (Transient Program Area) from the address &H100.

The memory configuration will be as follows :

&H0000	System area
&H100	Sony disk BASIC interpreter approx. 40 k bytes
(&HA000)	BASIC work area approx. 14 k bytes
(&HD800)	CP/M (BDOS, BIOS) approx. 10 k bytes
&HFFFF	

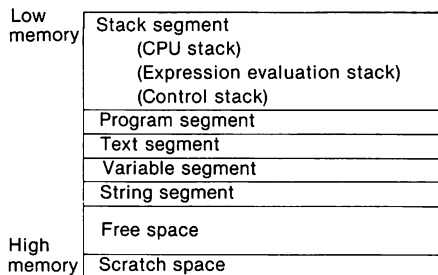
- The highest memory address of the Sony disk BASIC interpreter or the lowest memory address of the CP/M differs among the versions. The addresses in () are the approximate value. The addresses &H6 and &H7 indicates the start address of the CP/M of your version. Using a PEEK function, the lowest memory address of the CP/M can be obtained.

PRINT HEX\$(PEEK (7) * 256 + PEEK (6) - 1, 4)

The lowest address of the Sony disk BASIC work area can be obtained by calculating from the free memory area using the FRE function.

The usable Sony disk BASIC work area (low and high memory addresses) is limited when MCLEAR command is executed.

When Sony disk BASIC is used, the continuous area between the low and high memory addresses is divided into several segments as follows :



Stack segments

This is the system stack area.

Program segment

The file loaded with LINK is stored.

Text segment

The Sony disk BASIC program is stored after conversion to internal codes. A statement entered in the direct command mode is also stored, but cleared after execution.

Statements entered by the 1 or 2 command in the debug mode are stored.

Variable segment

The pairs of names and values of all variables that appeared during execution of the Sony disk BASIC program are sequentially stored. See the next page and after for the formats of the values in the memory.

String segment

The contents of character strings, values of string variables, are stored. Garbage collection is performed when FRE (" ") is used or when the memory becomes full.

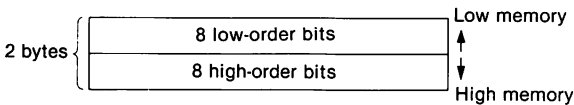
Scratch space

This is a temporary buffer used for evaluating an expression. Operation is performed using this buffer.

INTERNAL REPRESENTATION OF NUMERIC AND STRING DATA

For Sony disk BASIC, integer, floating-point, and string data are stored in the memory as follows :

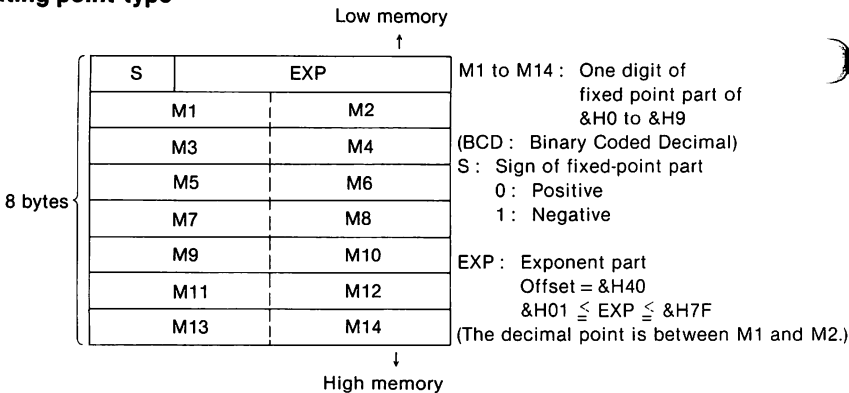
Integer type



Example : 12345 (&H3039) -12345 (&HCFC7)

&H39	&HC7
&H30	&HCF

Floating-point type



Therefore, if S = 0 :

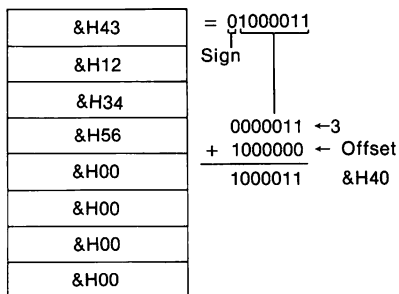
$$[M1 \times 10^0 + M2 \times 10^{-1} + \dots + M14 \times 10^{-13}] \times 10^{EXP - \&H40}$$

if S = 1 :

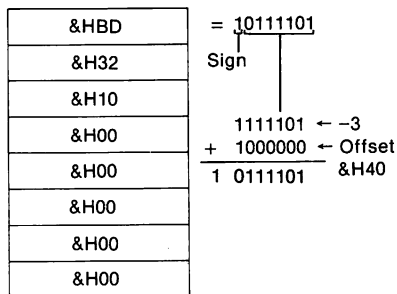
$$-[M1 \times 10^0 + M2 \times 10^{-1} + \dots + M14 \times 10^{-13}] \times 10^{EXP - \&H40}$$

However, if the data is 0, all eight bytes are &H00.

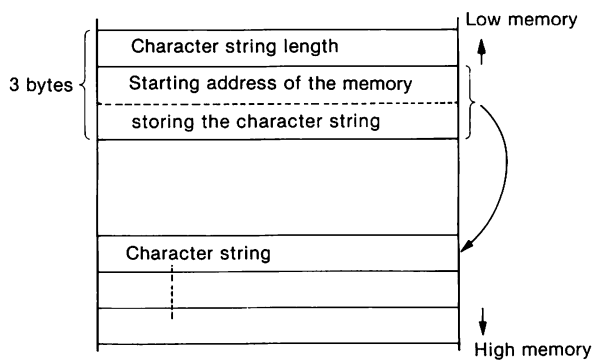
Example: 1.23456E3



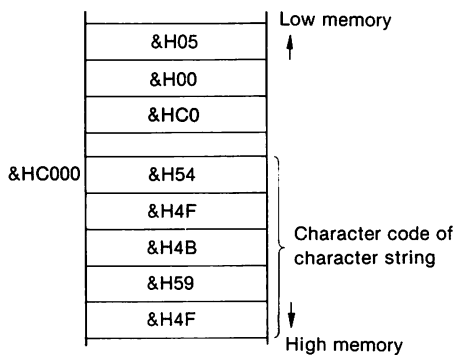
-3.21E-3



String type



Example: "TOKYO"

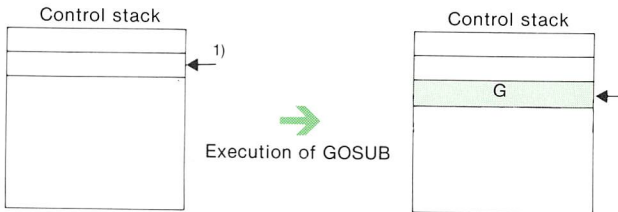


4. FOR TO NEXT LOOP PROCESSING AND RELATIONSHIP WITH GOSUB

When FOR to NEXT and GOSUB to RETURN statements are executed, the control stack of the stack segments allocated in the memory is used. FOR and GOSUB statement processing, and stack states are explained below.

Execution of GOSUB

Put the line number *G* in the program to which control must return after execution of a subroutine.



Execution of RETURN

If *G* is in the place indicated by the control stack pointer, program execution will return to line number *G*. If *G* is not found, *G* will be searched for repeatedly until it is found.



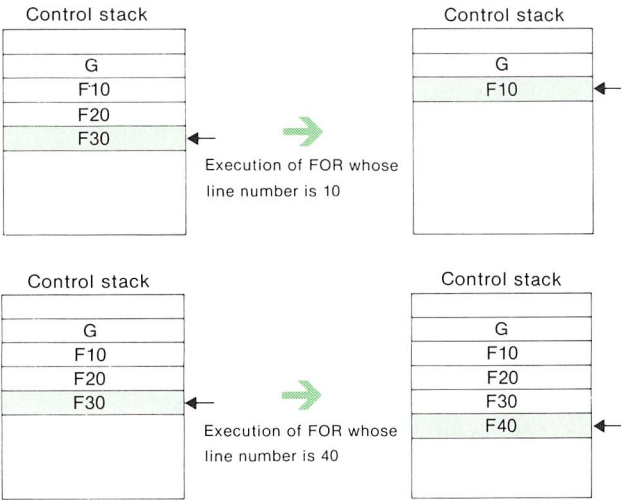
If *G* cannot be found at last, the “RETURN without GOSUB” error will occur.

Execution of FOR

Checks whether there is F with the same line number as the number of the current FOR statement between the pointer position and the position where data other than F is stored.

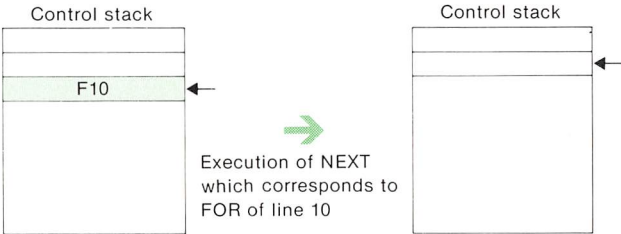
If there is F with the same line number, the pointer will be moved there and clear the stack up to that point.

If there is no F with the same line number, the F with the line number will be put.



Execution of NEXT

Checks whether there is F with the corresponding line number between the pointer position and the position where data other than F is stored. If F is found, it will transfer control to F, the beginning of the loop. If F is not found (data other than F is found or the stack becomes empty), the “NEXT without FOR” error will occur.

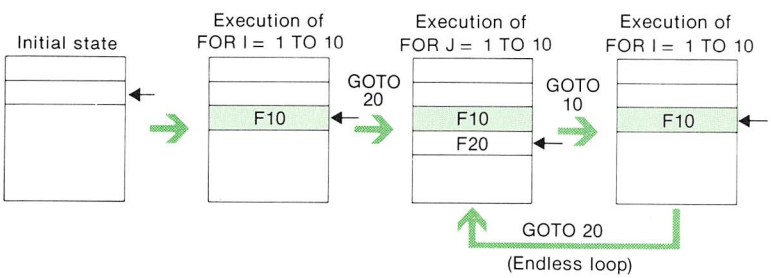


Examples

Here, explains the execution of the example programs and the state of control stack.

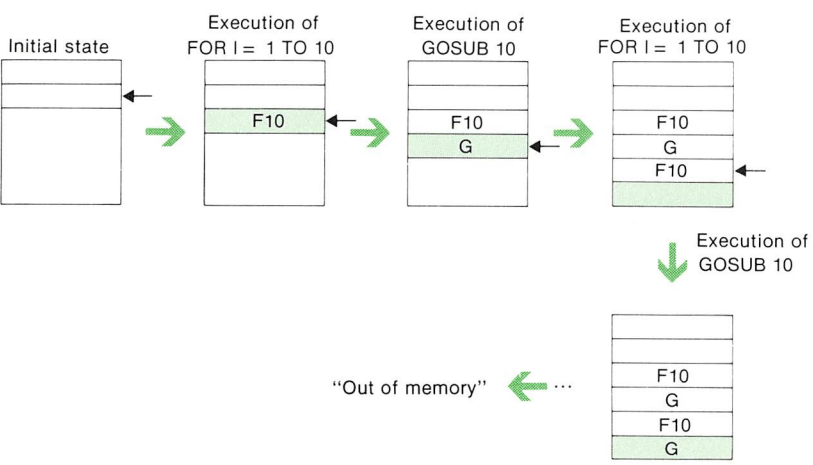
Example 1:

```
10 FOR I=1 TO 10 : GOTO 20 : NEXT I
20 FOR J=1 TO 10 : GOTO 10 : NEXT J
```



Example 2:

```
10 FOR I=1 TO 10
20 GOSUB 10 :NEXT I
```

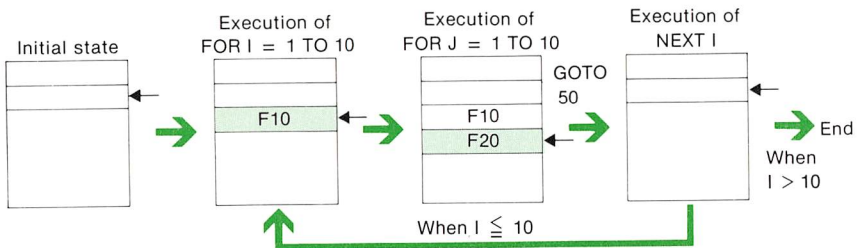


Example 3:

```

10 FOR I=1 TO 10
20 FOR J=1 TO 10
30 GOTO 50
40 NEXT J
50 NEXT I

```

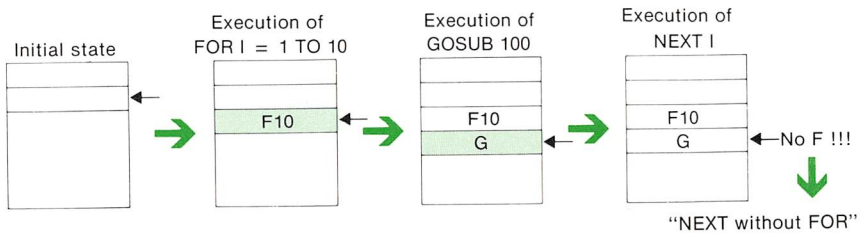


Example 4:

```

10 FOR I=1 TO 10
20 GOSUB 100
30 NEXT I
40 END
100 GOTO 30

```



5. MACHINE LANGUAGE SUBROUTINE

There are two types of machine language subroutines : A user routine and a routine loaded from a device with a LINK command. Both routines can be called from Sony disk BASIC with a CALL statement. When control is transferred to a machine language subroutine with a CALL statement, the variable values used can be transferred to the subroutine. Moreover, values can be transferred to a subroutine, and the subroutine execution results can be returned to Sony disk BASIC with an ANN function.

TRANSFERRING VALUES TO USER ROUTINE WITH A CALL COMMAND

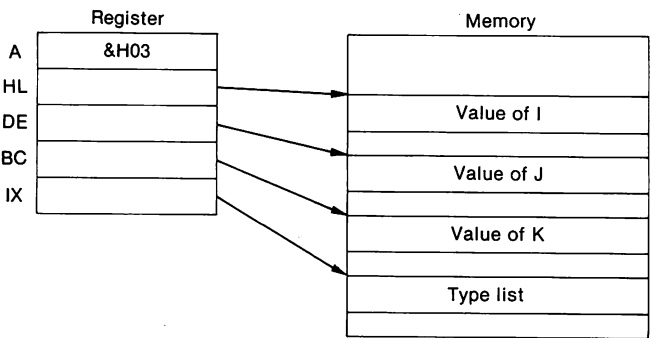
Arrange the variables or constants to be transferred in a CALL statement. When the CALL statement is executed, the starting address of the memory storing the variables or constants to be transferred is set in the CPU register. The setting in the CPU register depends on the number of values to be transferred. Up to 10 values can be transferred.

Three or fewer values

Register A	Number of values to be transferred
Register HL	Starting address of first value
Register DE	Starting address of second value
Register BC	Starting address of third value
Register IX	Starting address of type list

Example :

CALL SUBROUT (I , J , K)

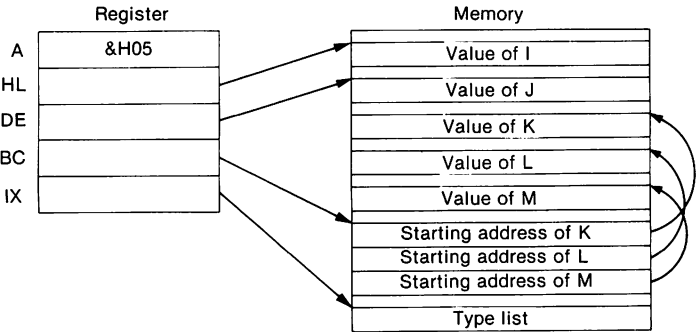


Four or more values

Register A	Number of values to be transferred
Register HL	Starting address of first value
Register DE	Starting address of second value
Register BC	Starting address of the starting address list of third and succeeding values
Register IX	Starting address of type list

Example :

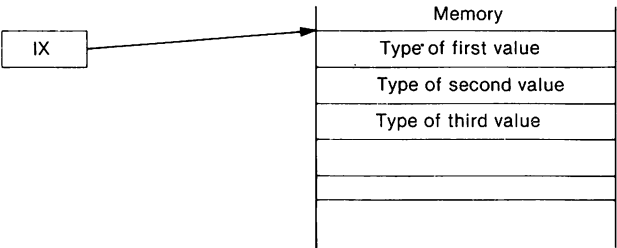
```
CALL SUBROUT1(I,J,K,L,M)
```



Type list

The type list is generated according to the types of the values to be transferred when the CALL statement is executed.

Type list structure

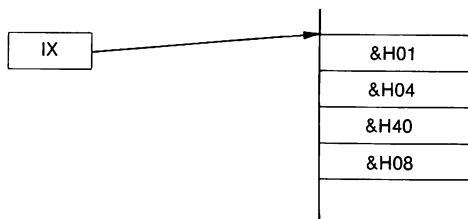


Type list contents

&H01	Integer type
&H04	Floating-point type
&H08	String type
&H40	Omitted

Example :

```
CALL SUB(I%,J#,,K$)
```



TRANSFERRING VALUES WITH A CALL COMMAND TO A MACHINE LANGUAGE SUBROUTINE WHICH IS LOADED WITH LINK

Arrange the variables or constants to be transferred in a CALL statement. The machine language subroutine loaded with LINK has a type descriptor, which is a type list of values to be transferred from Sony disk BASIC to the subroutine. Values are transferred only when the types of the values arranged in the CALL statement match or can be converted to those indicated by the type descriptor. When type conversion is impossible, an error will occur indicating "Type mismatch" (for numeric and string types) or "Overflow" (when floating-point type is converted to integer type).

The starting addresses of the memory storing the values after conversion (if there is a conversion) is set in the CPU register like an ordinary transfer to the machine language subroutine, as explained in the last section ; however, the type list lists the types of the values specified in the CALL statement, not the types of the values after conversion.

Structure

Contents

Note

91

Type conversion of the data to be transferred

Data type arranged in CALL statement

Value of type descriptor

	&H40 (Omitted)	&H08 (String type)	&H04 (Floating-point type)	&H01 (Integer type)
&H80 (No type check)	○	○	○	○
&H48 (String type, can be omitted)	○	○	×	×
&H45 (1-byte type, can be omitted)	○	×	*1	*4
&H44 (Floating-point type, can be omitted)	○	×	○	*5
&H42 (Unsigned integer type, can be omitted)	○	×	*2	○
&H41 (Integer type, can be omitted)	○	×	*3	○
&H08 (String type)	×	○	×	×
&H05 (1-byte type)	×	×	*1	*4
&H04 (Floating-point type)	×	×	○	*5
&H02 (Unsigned integer type)	×	×	*2	○
&H01 (Integer type)	×	×	*3	○

Meaning of symbols used in this table are as follows:

O: Transferred as it is

X: Type conversion is impossible; an error occurs

*1: Rounded to the nearest whole number

Conversion is possible if the rounded value is from 0 to 255.

Example: 0.49 → 0 (&H00)

254.5 → 255 (&HFF)

300 → Error

*2: Rounded to the nearest whole number

Conversion is possible if the rounded value is from
-32768 to 65535

Example: -32768.5 → -32768 (&H8000)

32767.49 → 32767 (&H7FFF)

65535.49 → 65535 (&HFFFF)

-65535 → Error

*3: Rounded to the nearest whole number

Conversion is possible if the rounded value is from
-32768 to 32767.

Example: -32768.5 → -32768 (&H8000)

-0.5 → 0 (&H0000)

32766.5 → 32767 (&H7FFF)

65535 → Error

*4: 0 to 255 are transferred as they are. Other values cause errors.

*5: The integer type is converted to floating-point type.

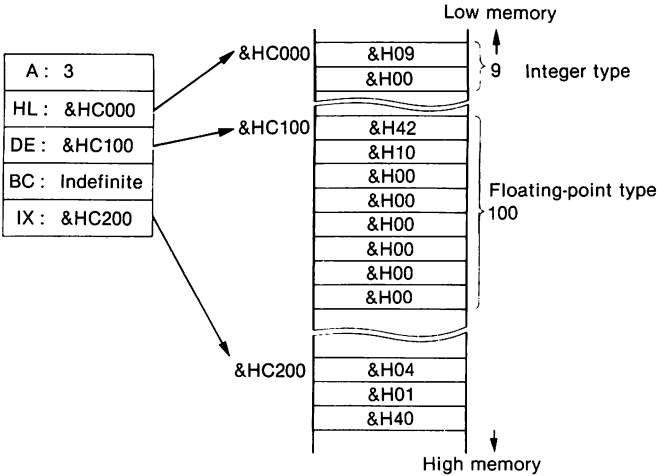
Example: When the subroutine (SUBROUT) loaded with LINK receives three values in the following sequence—integer, floating-point, and string—and the second and the third values may be omitted, the type descriptor of SUBROUT must be as follows :

&H01
&H44
&H48
&H00

If

```
CALL SUBROUT(8.9,100)
```

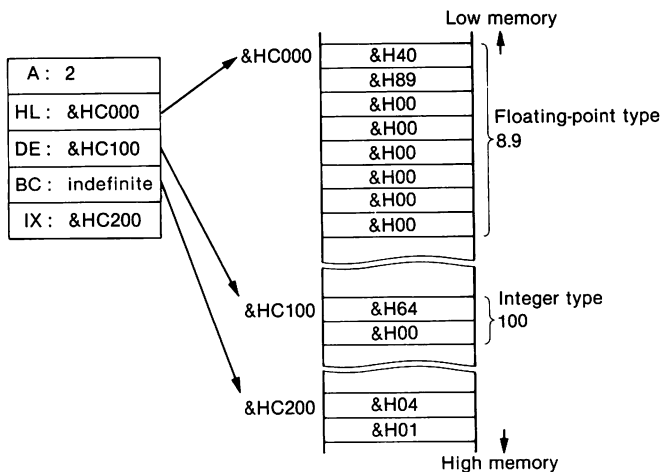
is executed, the values can be transferred by setting the registers as follows :



If the subroutine does not have a type descriptor, the following values will be transferred when

```
CALL SUBROUT(8.9,100)
```

is executed :



CALLING A SUBROUTINE WITH AN ANN FUNCTION

Use the ANN function to return the subroutine execution results to Sony BASIC.

First, define the subroutine starting address in the DEF ANN statement.

DEF ANN N = starting address

N is an integer from 0 to 9, and up to 10 subroutine starting addresses can be defined as ANN functions.

Then, specify

```
X=ANN N(I)
```

using the ANN function; the following is set :

Register A	Type of value of I (&H01: Integer type &H04: Floating-point type &H08: String type
Register HL	Starting address of the memory storing the value of I

The value of I is transferred to the subroutine starting with the address defined as ANN N, and then the subroutine is executed. After execution the values set in CPU registers A and HL are given as the ANN function value when they return to Sony disk BASIC.

Register A	Type of value to be returned &H01 or &H02 : Integer type &H04 : Floating-point type &H08 : String type
Register HL	Start address of the memory storing the value returned (Register A : &H01, &H04, &H08) Value returned (Register A : &H02)

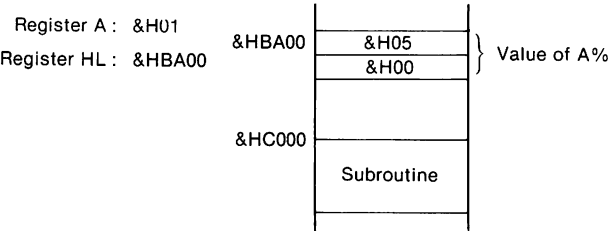
Example: Assume that a certain subroutine starts with address &HC000. When

```
DEF ANN 0=&HC000
```

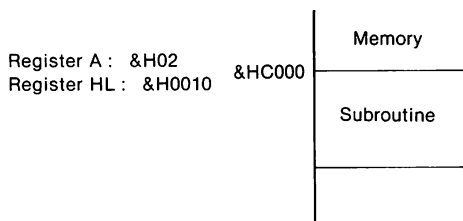
is executed, the above subroutine is defined as ANN 0. If the following is specified :

```
A%=5
I=ANN 0(A%)
```

the CPU register will be set as shown below, and control will be transferred to the subroutine starting with address &HC000.



If the CPU register is set as shown below after execution of the subroutine :



Control will return to Sony disk BASIC, and the value of register HL (that is &H0010) will be returned as the value of ANN 0, because register A is &H02, &H0010 will be assigned to the variable I.

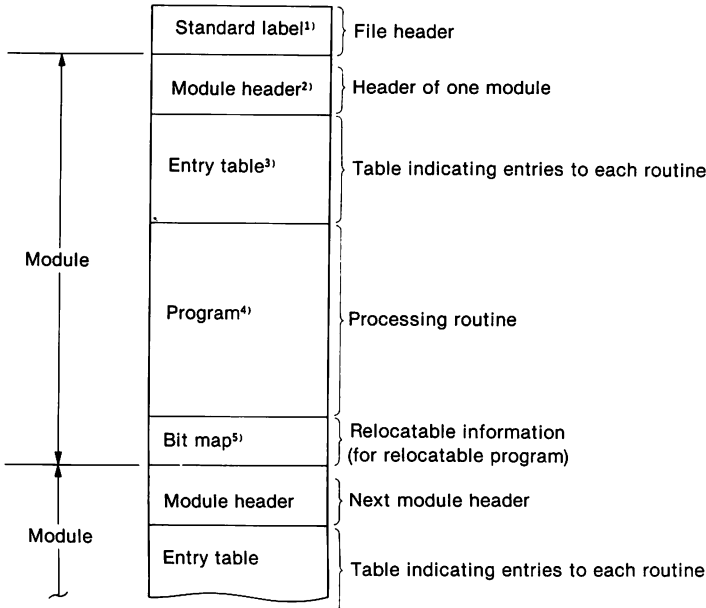
TO OCCUR A BASIC ERROR DURING EXECUTION OF THE MACHINE LANGUAGE SUBROUTINE

When the control is transferred to the machine language subroutine, the Sony disk BASIC interpreter pushes its error routine address into the stack, then pushes the normal return address.

When the control returns to the Sony disk BASIC program from the subroutine, the normal return address is popped and is referred. If a pop of the normal return address has been executed in the subroutine and the control is returned to the Sony disk BASIC program, the error routine address is popped and the control returns to the BASIC error routine in the interpreter. The error whose error number is specified by the contents of the A register at that time will occur.

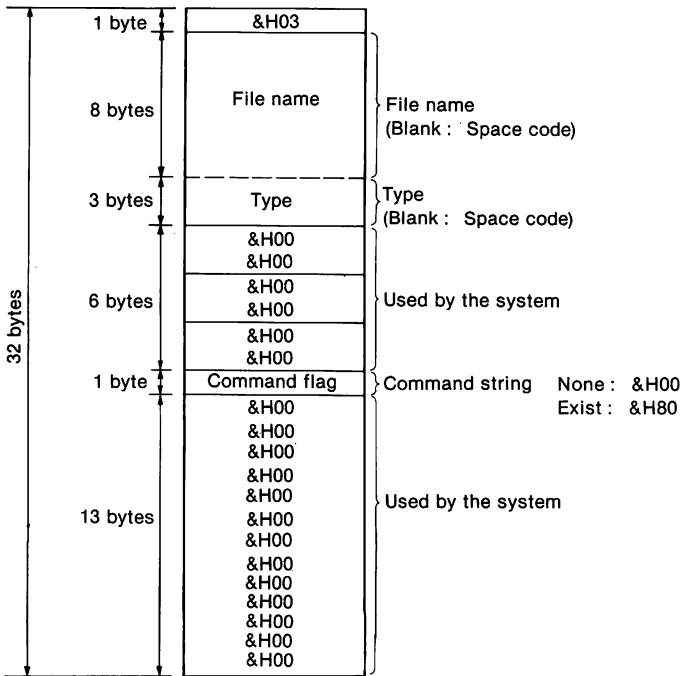
6. FORMAT OF THE LINK MODULE

A LINK command can be used to load files having the following format :

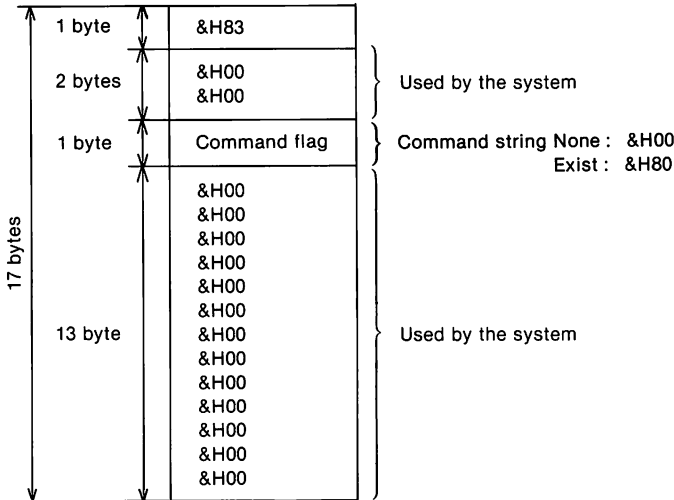


1) Standard label

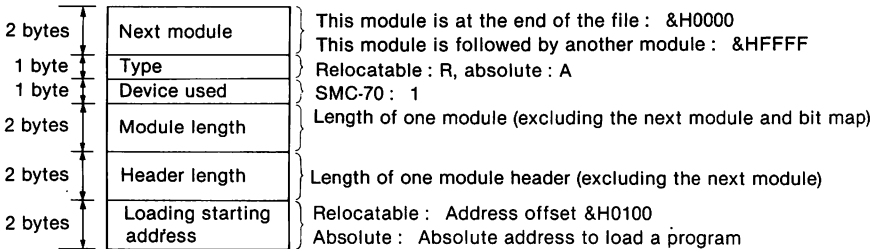
When a file is generated on a device except a disk



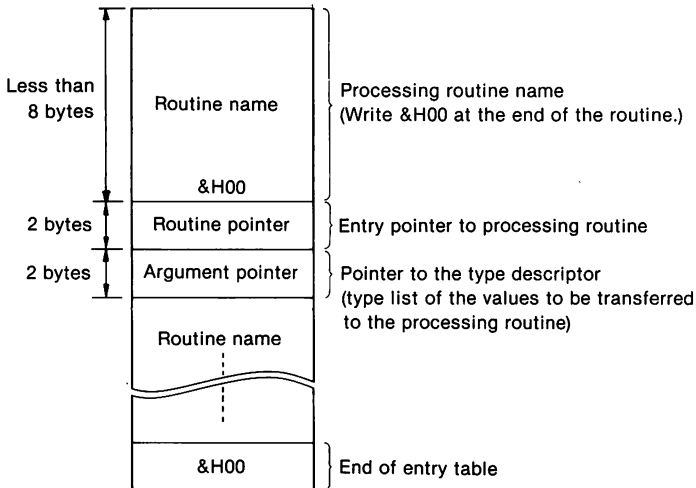
When a file is generated on a disk



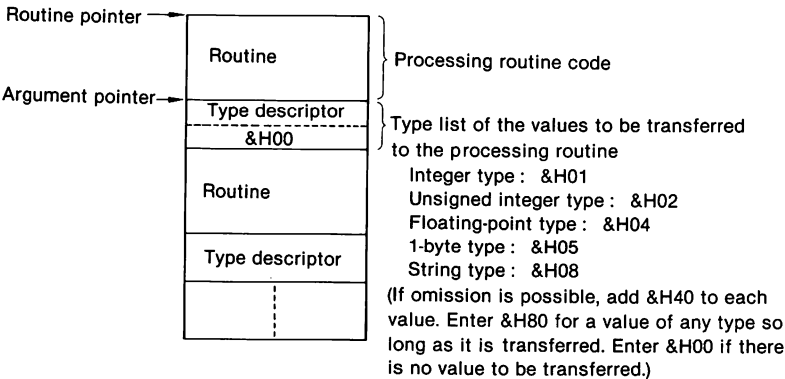
2) Module header



3) Entry table



4) Program



5) Bit map (only for relocatable program)

Each 1-bit data in the bit map indicates whether relative processing is to be performed for each program byte.

Relative processing required : 1

Relative processing not required : 0

Example : JP 200

...	C3	00	02	...	(Code)
	0	0	1	...	(Bit map)

When a file is loaded with a LINK command, the file name and type indicated by the standard label are used.

When a subroutine is called with a CALL command, the routine name in the entry table is used.

7. I/O PORT ASSIGNMENT

BUILT-IN I/O

	Application	Port number	
Display control	Character RAM	&H00 to &H07	VRAM
	Attribute RAM	&H08 to &H0F	
	PCG	&H10 to &H17	
	Graphic RAM	&H80 to &HFF	
	CRT controller	&H18 &H19	
	Display mode switching	&H20	
	Border area control	&H23	
Keyboard	Keyboard data	&H1A	
	Keyboard status/command	&H1B	
Printer	Printer output data	&H22	
	Printer strobe	&H1C &H1D	
	Printer status	&H1D	
RS-232C	Send/receive data	&H26	
	Mode/command/status	&H27	
	Interruption	&H1E &H1F	
Timer	Write data/addressing	&H24	
	Reading	&H25	
Others	60 Hz interrupt control	&H21	
	Various status and control data	&H1C &H1D	

EXTERNAL INTERFACE INPUT/OUTPUT

I/O device		Port number
Floppydisk control	No. 1	&H30 to &H34
	No. 2	&H28 to &H2C
RS-232C	No. 2	&H2D to &H2F
	No. 3	&H35 to &H37
Cache disk unit		&H38 to &H3B
RGB superimposer		&H3C
IEEE-488 interface		&H40 to &H47
ROM pack	Auto start ROM	&H70
	IEEE-488 interface ROM	&H74
	Reserved	&H75 &H76
	Others	&H71 to &H73 &H77 to &H7F
System reserved		&H3D to &3F &H48 to &H51 &H54 to &H5B
Not used		&H52, &H53 &H5C to &H6F

8. VIDEO RAM ADDRESS

Refer to the “Hardware Reference Manual” for details.

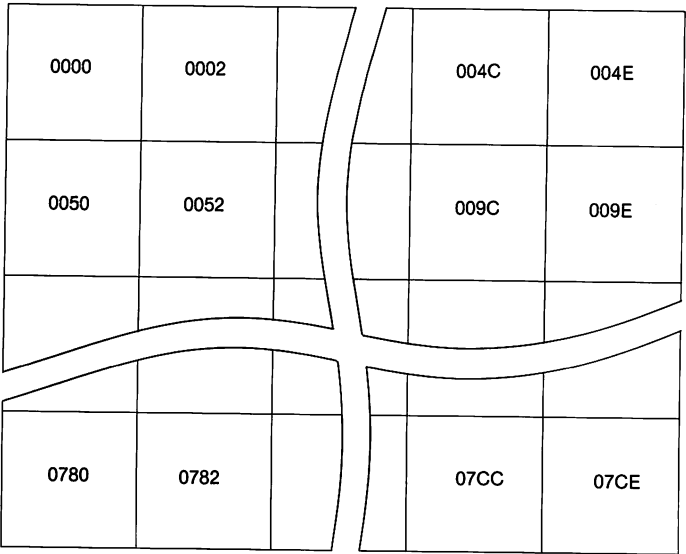
Character display screen (GRAM)

① 80 characters × 25 lines mode

0000	0001	0002	0003	0004	0005	004D	004E	004F
0050	0051	0052	0053	0054	0055	009D	009E	009F
0780	0781	0782	0783	0784	0785	07CD	07CE	07CF

ARAM address = (GRAM address + &H0800)

② 40 characters × 25 lines mode,



1 dot

105

Page 2

[illegible]

Page 3

[illegible]

② Standard mode (320 × 200 dots)

80:08	80:01	180:28	80:03	80:04	80:05	80:06	80:07	80:08	80:09	80:0A	80:0B	80:0A	80:9B	80:9C	80:9D	80:9E	80:9F
90:00	90:01	190:02	90:03	90:04	90:05	90:06	90:07	90:08	90:09	90:0A	90:0B	90:0A	90:9B	90:9C	90:9D	90:9E	90:9F
A0:00	A0:01	1A0:02	A0:03	A0:04	A0:05	A0:06	A0:07	A0:08	A0:09	A0:0A	A0:0B	A0:0A	A0:9B	A0:9C	A0:9D	A0:9E	A0:9F
B0:00	B0:01	1B0:02	B0:03	B0:04	B0:05	B0:06	B0:07	B0:08	B0:09	B0:0A	B0:0B	B0:0A	B0:9B	B0:9C	B0:9D	B0:9E	B0:9F
C0:00	C0:01	1C0:02	C0:03	C0:04	C0:05	C0:06	C0:07	C0:08	C0:09	C0:0A	C0:0B	C0:0A	C0:9B	C0:9C	C0:9D	C0:9E	C0:9F
D0:00	D0:01	1D0:02	D0:03	D0:04	D0:05	D0:06	D0:07	D0:08	D0:09	D0:0A	D0:0B	D0:0A	D0:9B	D0:9C	D0:9D	D0:9E	D0:9F
E0:00	E0:01	1E0:02	E0:03	E0:04	E0:05	E0:06	E0:07	E0:08	E0:09	E0:0A	E0:0B	E0:0A	E0:9B	E0:9C	E0:9D	E0:9E	E0:9F
F0:00	F0:01	1F0:02	F0:03	F0:04	F0:05	F0:06	F0:07	F0:08	F0:09	F0:0A	F0:0B	F0:0A	F0:9B	F0:9C	F0:9D	F0:9E	F0:9F
80:A0	80:A1	180:A2	80:A3	80:A4	80:A5	80:A6	80:A7	80:A8	80:A9	80:AA	80:AB	80:AA	80:9B	80:9C	80:9D	80:9E	80:9F
90:A0	90:A1	190:A2	90:A3	90:A4	90:A5	90:A6	90:A7	90:A8	90:A9	90:AA	90:AB	90:AA	90:9B	90:9C	90:9D	90:9E	90:9F
A0:A0	A0:A1	1A0:A2	A0:A3	A0:A4	A0:A5	A0:A6	A0:A7	A0:A8	A0:A9	A0:AA	A0:AB	A0:AA	A0:9B	A0:9C	A0:9D	A0:9E	A0:9F
B0:A0	B0:A1	1B0:A2	B0:A3	B0:A4	B0:A5	B0:A6	B0:A7	B0:A8	B0:A9	B0:AA	B0:AB	B0:AA	B0:9B	B0:9C	B0:9D	B0:9E	B0:9F
C0:A0	C0:A1	1C0:A2	C0:A3	C0:A4	C0:A5	C0:A6	C0:A7	C0:A8	C0:A9	C0:AA	C0:AB	C0:AA	C0:9B	C0:9C	C0:9D	C0:9E	C0:9F
D0:A0	D0:A1	1D0:A2	D0:A3	D0:A4	D0:A5	D0:A6	D0:A7	D0:A8	D0:A9	D0:AA	D0:AB	D0:AA	D0:9B	D0:9C	D0:9D	D0:9E	D0:9F
E0:A0	E0:A1	1E0:A2	E0:A3	E0:A4	E0:A5	E0:A6	E0:A7	E0:A8	E0:A9	E0:AA	E0:AB	E0:AA	E0:9B	E0:9C	E0:9D	E0:9E	E0:9F
F0:A0	F0:A1	1F0:A2	F0:A3	F0:A4	F0:A5	F0:A6	F0:A7	F0:A8	F0:A9	F0:AA	F0:AB	F0:AA	F0:9B	F0:9C	F0:9D	F0:9E	F0:9F
1 dot																	
8F:08	8F:01	18F:02	8F:03	8F:04	8F:05	8F:06	8F:07	8F:08	8F:09	8F:0A	8F:0B	8F:0A	8F:9B	8F:9C	8F:9D	8F:9E	8F:9F
9F:00	9F:01	19F:02	9F:03	9F:04	9F:05	9F:06	9F:07	9F:08	9F:09	9F:0A	9F:0B	9F:0A	9F:9B	9F:9C	9F:9D	9F:9E	9F:9F
AF:00	AF:01	1AF:02	AF:03	AF:04	AF:05	AF:06	AF:07	AF:08	AF:09	AF:0A	AF:0B	AF:0A	AF:9B	AF:9C	AF:9D	AF:9E	AF:9F
BF:00	BF:01	1BF:02	BF:03	BF:04	BF:05	BF:06	BF:07	BF:08	BF:09	BF:0A	BF:0B	BF:0A	BF:9B	BF:9C	BF:9D	BF:9E	BF:9F
CF:00	CF:01	1CF:02	CF:03	CF:04	CF:05	CF:06	CF:07	CF:08	CF:09	CF:0A	CF:0B	CF:0A	CF:9B	CF:9C	CF:9D	CF:9E	CF:9F
DF:00	DF:01	1DF:02	DF:03	DF:04	DF:05	DF:06	DF:07	DF:08	DF:09	DF:0A	DF:0B	DF:0A	DF:9B	DF:9C	DF:9D	DF:9E	DF:9F
EF:00	EF:01	1EF:02	EF:03	EF:04	EF:05	EF:06	EF:07	EF:08	EF:09	EF:0A	EF:0B	EF:0A	EF:9B	EF:9C	EF:9D	EF:9E	EF:9F
FF:00	FF:01	1FF:02	FF:03	FF:04	FF:05	FF:06	FF:07	FF:08	FF:09	FF:0A	FF:0B	FF:0A	FF:9B	FF:9C	FF:9D	FF:9E	FF:9F

③ High-resolution mode (640 × 200 dots)

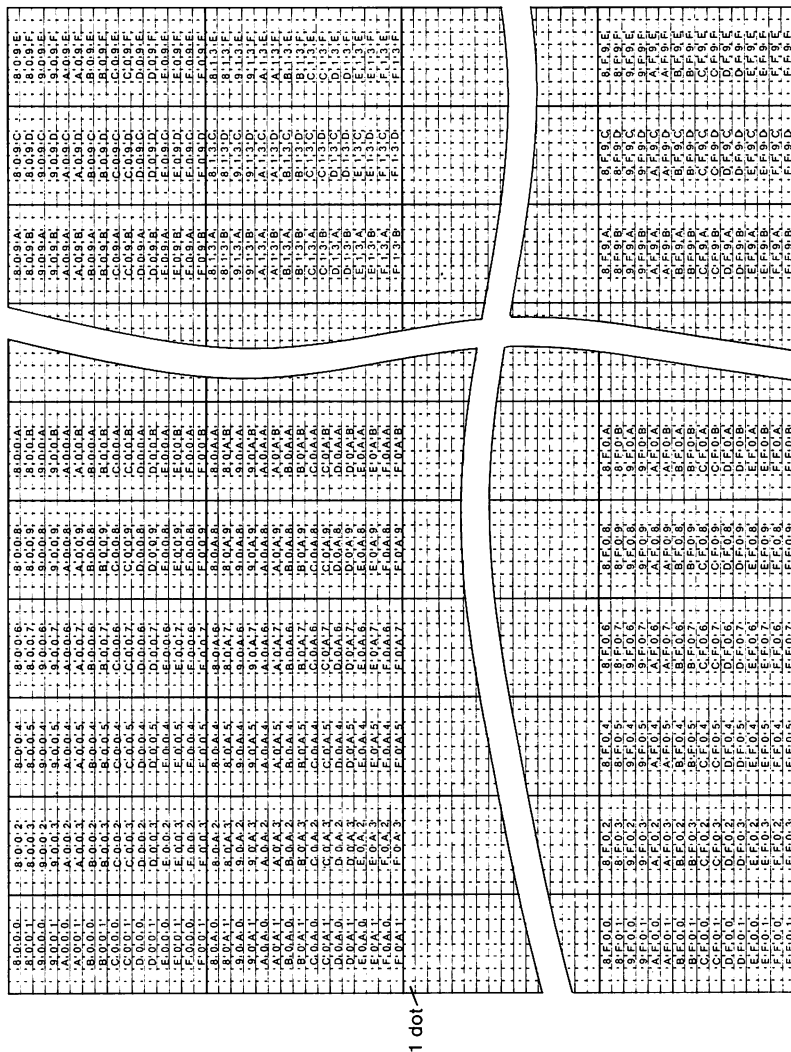
80:00	80:01	80:02	80:03	80:04	80:05	80:06	80:07	80:08	80:09	80:0A	80:0B
90:00	90:01	90:02	90:03	90:04	90:05	90:06	90:07	90:08	90:09	90:0A	90:0B
A0:00	A0:01	A0:02	A0:03	A0:04	A0:05	A0:06	A0:07	A0:08	A0:09	A0:0A	A0:0B
B0:00	B0:01	B0:02	B0:03	B0:04	B0:05	B0:06	B0:07	B0:08	B0:09	B0:0A	B0:0B
C0:00	C0:01	C0:02	C0:03	C0:04	C0:05	C0:06	C0:07	C0:08	C0:09	C0:0A	C0:0B
D0:00	D0:01	D0:02	D0:03	D0:04	D0:05	D0:06	D0:07	D0:08	D0:09	D0:0A	D0:0B
E0:00	E0:01	E0:02	E0:03	E0:04	E0:05	E0:06	E0:07	E0:08	E0:09	E0:0A	E0:0B
F0:00	F0:01	F0:02	F0:03	F0:04	F0:05	F0:06	F0:07	F0:08	F0:09	F0:0A	F0:0B
80:A0	80:A1	80:A2	80:A3	80:A4	80:A5	80:A6	80:A7	80:A8	80:A9	80:AA	80:AB
90:A0	90:A1	90:A2	90:A3	90:A4	90:A5	90:A6	90:A7	90:A8	90:A9	90:AA	90:AB
A0:A0	A0:A1	A0:A2	A0:A3	A0:A4	A0:A5	A0:A6	A0:A7	A0:A8	A0:A9	A0:AA	A0:AB
B0:A0	B0:A1	B0:A2	B0:A3	B0:A4	B0:A5	B0:A6	B0:A7	B0:A8	B0:A9	B0:AA	B0:AB
C0:A0	C0:A1	C0:A2	C0:A3	C0:A4	C0:A5	C0:A6	C0:A7	C0:A8	C0:A9	C0:AA	C0:AB
D0:A0	D0:A1	D0:A2	D0:A3	D0:A4	D0:A5	D0:A6	D0:A7	D0:A8	D0:A9	D0:AA	D0:AB
E0:A0	E0:A1	E0:A2	E0:A3	E0:A4	E0:A5	E0:A6	E0:A7	E0:A8	E0:A9	E0:AA	E0:AB
F0:A0	F0:A1	F0:A2	F0:A3	F0:A4	F0:A5	F0:A6	F0:A7	F0:A8	F0:A9	F0:AA	F0:AB
8F:00	8F:01	8F:02	8F:03	8F:04	8F:05	8F:06	8F:07	8F:08	8F:09	8F:0A	8F:0B
9F:00	9F:01	9F:02	9F:03	9F:04	9F:05	9F:06	9F:07	9F:08	9F:09	9F:0A	9F:0B
AF:00	AF:01	AF:02	AF:03	AF:04	AF:05	AF:06	AF:07	AF:08	AF:09	AF:0A	AF:0B
BF:00	BF:01	BF:02	BF:03	BF:04	BF:05	BF:06	BF:07	BF:08	BF:09	BF:0A	BF:0B
CF:00	CF:01	CF:02	CF:03	CF:04	CF:05	CF:06	CF:07	CF:08	CF:09	CF:0A	CF:0B
DF:00	DF:01	DF:02	DF:03	DF:04	DF:05	DF:06	DF:07	DF:08	DF:09	DF:0A	DF:0B
EF:00	EF:01	EF:02	EF:03	EF:04	EF:05	EF:06	EF:07	EF:08	EF:09	EF:0A	EF:0B
FF:00	FF:01	FF:02	FF:03	FF:04	FF:05	FF:06	FF:07	FF:08	FF:09	FF:0A	FF:0B

1 dot

80:9A	80:9B	80:9C	80:9D	80:9E	80:9F
90:9A	90:9B	90:9C	90:9D	90:9E	90:9F
A0:9A	A0:9B	A0:9C	A0:9D	A0:9E	A0:9F
B0:9A	B0:9B	B0:9C	B0:9D	B0:9E	B0:9F
C0:9A	C0:9B	C0:9C	C0:9D	C0:9E	C0:9F
D0:9A	D0:9B	D0:9C	D0:9D	D0:9E	D0:9F
E0:9A	E0:9B	E0:9C	E0:9D	E0:9E	E0:9F
F0:9A	F0:9B	F0:9C	F0:9D	F0:9E	F0:9F
81:3A	81:3B	81:3C	81:3D	81:3E	81:3F
91:3A	91:3B	91:3C	91:3D	91:3E	91:3F
A1:3A	A1:3B	A1:3C	A1:3D	A1:3E	A1:3F
B1:3A	B1:3B	B1:3C	B1:3D	B1:3E	B1:3F
C1:3A	C1:3B	C1:3C	C1:3D	C1:3E	C1:3F
D1:3A	D1:3B	D1:3C	D1:3D	D1:3E	D1:3F
E1:3A	E1:3B	E1:3C	E1:3D	E1:3E	E1:3F
F1:3A	F1:3B	F1:3C	F1:3D	F1:3E	F1:3F

8F:9A	8F:9B	8F:9C	8F:9D	8F:9E	8F:9F
9F:9A	9F:9B	9F:9C	9F:9D	9F:9E	9F:9F
AF:9A	AF:9B	AF:9C	AF:9D	AF:9E	AF:9F
BF:9A	BF:9B	BF:9C	BF:9D	BF:9E	BF:9F
CF:9A	CF:9B	CF:9C	CF:9D	CF:9E	CF:9F
DF:9A	DF:9B	DF:9C	DF:9D	DF:9E	DF:9F
EF:9A	EF:9B	EF:9C	EF:9D	EF:9E	EF:9F
FF:9A	FF:9B	FF:9C	FF:9D	FF:9E	FF:9F

④ Super high-resolution mode (640 × 400 dots)



9. CHARACTER/KEY CODE TABLE

Decimal	Hexa-decimal	Character	Decimal	Hexa-decimal	Character	Decimal	Hexa-decimal	Character
0	0	NUL	48	30	0	96	60	·
1	1	F1	49	31	1	97	61	a
2	2	F2	50	32	2	98	62	b
3	3	↑F5	51	33	3	99	63	c
4	4	F3	52	34	4	100	64	d
5	5	↑F4	53	35	5	101	65	e
6	6	F4	54	36	6	102	66	f
7	7	▲F4	55	37	7	103	67	g
8	8	BS	56	38	8	104	68	h
9	9	TAB	57	39	9	105	69	i
10	A	LF	58	3A	:	106	6A	j
11	B	F5	59	3B	;	107	6B	k
12	C	▲F5	60	3C	<	108	6C	l
13	D	RETURN	61	3D	=	109	6D	m
14	E	CLR	62	3E	>	110	6E	n
15	F	INS	63	3F	?	111	6F	o
16	10	▲F2	64	40	@	112	70	p
17	11	DEL	65	41	A	113	71	q
18	12	↑F3	66	42	B	114	72	r
19	13	▲F3	67	43	C	115	73	s
20	14	HOME	68	44	D	116	74	t
21	15	↑F1	69	45	E	117	75	u
22	16	←	70	46	F	118	76	v
23	17	↑	71	47	G	119	77	w
24	18	↑F2	72	48	H	120	78	x
25	19	→	73	49	I	121	79	y
26	1A	▲F1	74	4A	J	122	7A	z
27	1B	ESC	75	4B	K	123	7B	{
28	1C	↓	76	4C	L	124	7C	
29	1D	HELP	77	4D	M	125	7D	}
30	1E		78	4E	N	126	7E	~
31	1F		79	4F	O	127	7F	RUB OUT
32	20	SPACE	80	50	P			
33	21	!	81	51	Q			
34	22	"	82	52	R			
35	23	#	83	53	S			
36	24	\$	84	54	T			
37	25	%	85	55	U			
38	26	&	86	56	V			
39	27	'	87	57	W			
40	28	(	88	58	X			
41	29	)	89	59	Y			
42	2A	*	90	5A	Z			
43	2B	+	91	5B	[			
44	2C	,	92	5C	\			
45	2D	—	93	5D	]			
46	2E	.	94	5E	▲			
47	2F	/	95	5F	—			

F1- F5 ... Function keys 1- 5

↑F1- ↑F5 ... **Shift** + Function keys 1- 5

▲F1- ▲F5 ... **CTRL** + Function keys 1- 5

10. SONY DISK BASIC RESERVED WORD CODE ASSIGNMENT

	8	9	A	B	C	D
0	FOR	OFF	DEF ANN	CLOAD	CHAIN	LJUST
1	TO	DATA	CLEAR	^	GO TO	RJUST
2	STEP	RESTORE	INPUT	*	WHILE	DIR
3	NEXT	REM	PRINT	/	WEND	ERA
4	GOTO	SEND	RECEIVE	MOD	RESUME	RENAME
5	CLOSE	OPTION BASE	OPEN	\	REM DEBUG	TYPE
6	GOSUB	RUN	'	+	SWAP	USER
7	RETURN	STOP	LIST	-	USING	EXIT
8	IF	END	LLIST	<	DEF	DISK RESET
9	THEN	CONT	EDIT	=	MERGE	LOG IN
A	ELSE	SYSTEM	DEBUG	>	LOAD	
B	LINK	DIM	NEW	NOT	SAVE	
C	:	LET	DELETE	AND	RECORD	
D	CALL	DEF FLT	RENUM	OR	AS	
E	ON	DEF INT	AUTO	XOR	GET	
F	READ	DEF STR	CSAVE	KEY LIST	PUT	

With &HFE

	8	9	A
0	POKE	PLAY	APUT
1	OUT	CURSOR	EVAL
2	RANDOMIZE	GPLOT	ERROR
3	MCLEAR	BOXF	CIRCLE
4	BOXE	PAINT	BEEP
5	CCLEAR	GGET	GMODE
6	CCOLOR	DEF FONT	GPLANE
7	GPUT	DEF KEY	GLOCATE
8	CONSOLE	PEN READ	PAUSE
9	LOCATE	SET TIME	
A	GCLEAR	WIPE	
B	GCOLOR	SET DATE	ACLEAR
C	DRAW	CPLOT	FONT RESET
D	VOUT	CGET	
E	MOTOR	CPUT	
F	LINE	AGET	

With &HFF

	8	9	A	B
0	ABS	PI	DATE\$	PROG
1	INT	LEN	TIMES\$	SPACES\$
2	FIX	ASC	APOINT	COND
3	CFLT	VAL	GPOINT	STRING\$
4	SGN	INSTR	CSRCLM	FN
5	SIN	LEFT\$	CSRLIN	REPLACES\$
6	COS	RIGHT\$	CPOINT	PENCLM
7	TAN	MID\$	ANN	PENLIN
8	ATN	CHR\$	CINT	GPOSX
9	TAB	STR\$	DAY\$	GPOSY
A	VINP	HEX\$	PENX	PEN
B	LOG	INKEY\$	PENY	IRND
C	LN	INP	CADR	EOF
D	EXP	PEEK	ERL	REC
E	SQR	FRE	ERR	
F	RND	VARPTR	SETQ	

11. COLOR CODE TABLES

CHARACTER DISPLAY SCREEN

Character color codes

Code	Character color	Code	Character color
0	Black	4	Red
1	Deep blue	5	Brilliant pink
2	Bright green	6	Yellow
3	Turquoise	7	White

Background color codes

Code	Background color
0	Transparent
1	White
2	Black
3	Complementary color

Attribute codes

Code	Character blinking	Character and back-ground colors
0	No	As specified
1	No	Inverted
2	Yes	As specified
3	Yes	Inverted

GRAPHIC DISPLAY SCREEN

Color codes

Standard or low resolution mode

Code	Color	Code	Color
0	Black	8	Dark green
1	Deep blue	9	Moss green
2	Bright green	10	Salmon
3	Turquoise	11	Tan
4	Red	12	Ash blue
5	Brilliant pink	13	Light blue
6	Yellow	14	Pale pink
7	White	15	Gray

High resolution mode

Code	Type 0	Type 1
0	Black	Black
1	Red	Red
2	Bright green	Bright green
3	White	Deep blue

Super high resolution mode

Code	Color
0	Black
1	White

Modes

Code	Logical operation
0	No logical operation (specified color is displayed)
1	NOT of current color (specified color is ignored)
2	AND
3	OR
4	XOR

Note

If a value outside the range indicated in the tables is specified, the following value will be assumed to have been specified.

Character display screen

Character color code: (specified value) MOD 8

Background color code: (specified value) MOD 4

Attribute code: (specified value) MOD 4

Graphic display screen

Color code:

(specified value) MOD 16 in standard or low resolution mode

(specified value) MOD 4 in high resolution mode

(specified value) MOD 2 in super high resolution mode

Mode of logical operation: An error will occur.

If the value of a color code, attribute code, mode, or coordinate in a command or function has figures after the decimal point, it will be rounded to the nearest whole number.

12. SCREEN MAP

1) 40-character mode character display screen

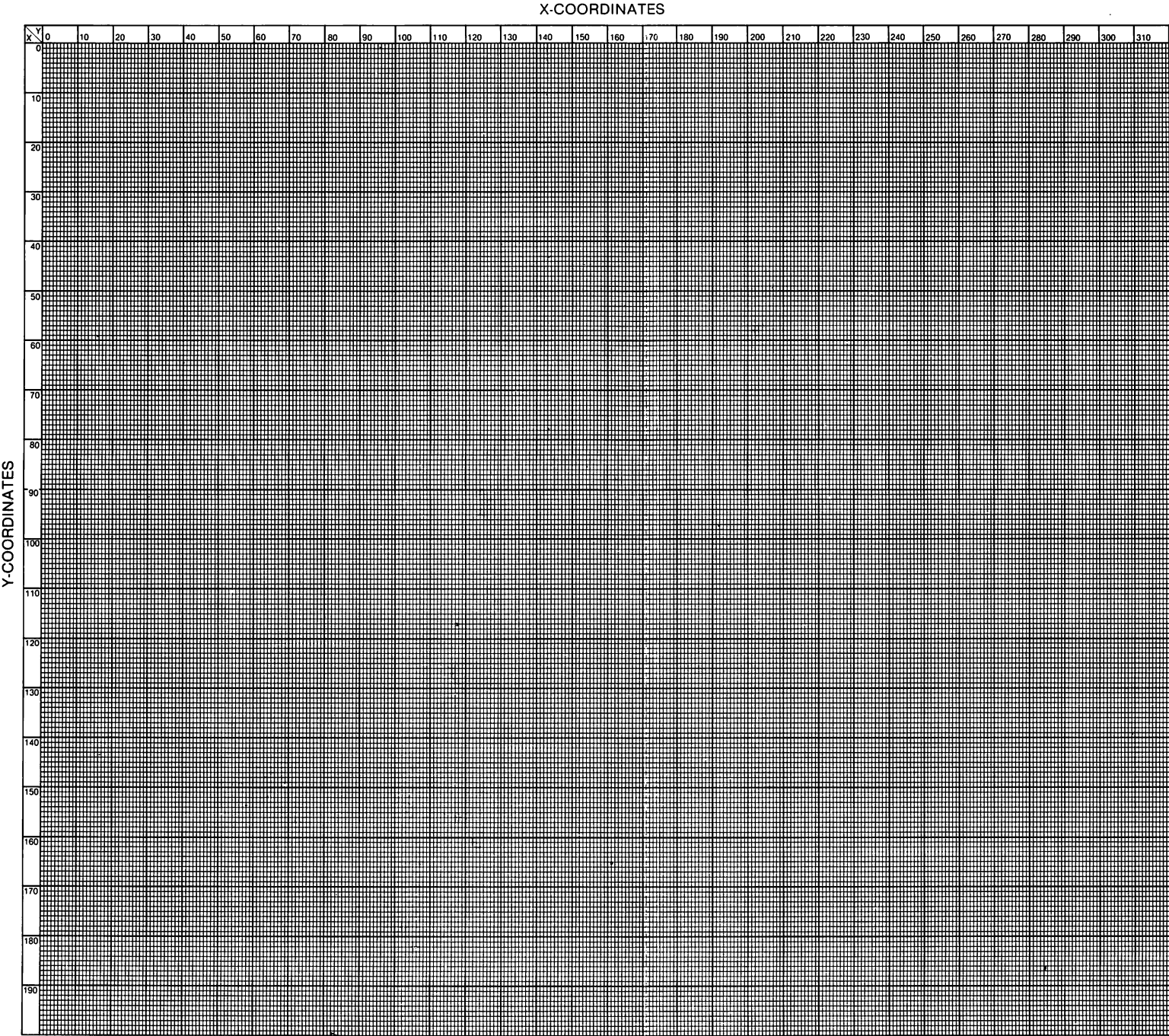
X-COORDINATES

[illegible]

X-COORDINATES

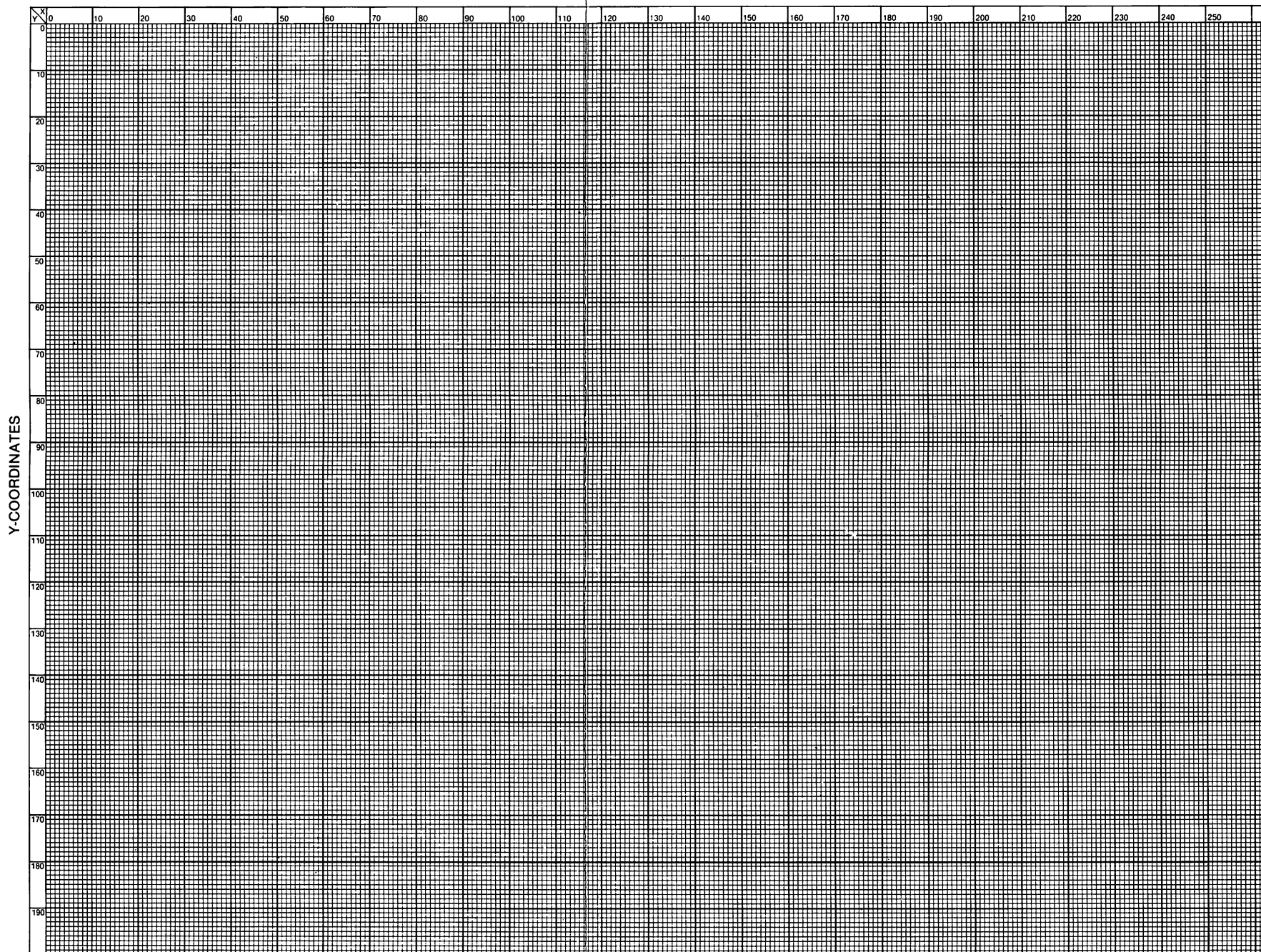
[illegible]

3) Graphic display screen (standard resplution, without compensation)



4) Graphic display screen (standard resolution, with compensation)

X-COORDINATES



ADDENDA

DEMONSTRATION PROGRAM

This Sony disk BASIC disk includes a program for demonstration named "DEMO.BAS".

After starting Sony disk BASIC, execute

LOAD "DEMO.BAS"

so that you can run the program.

FUNCTION KEY PANEL

The supplied panel is to be put on the keyboard as illustrated below. Initial definition of each key is indicated and you can use the keys easily.

