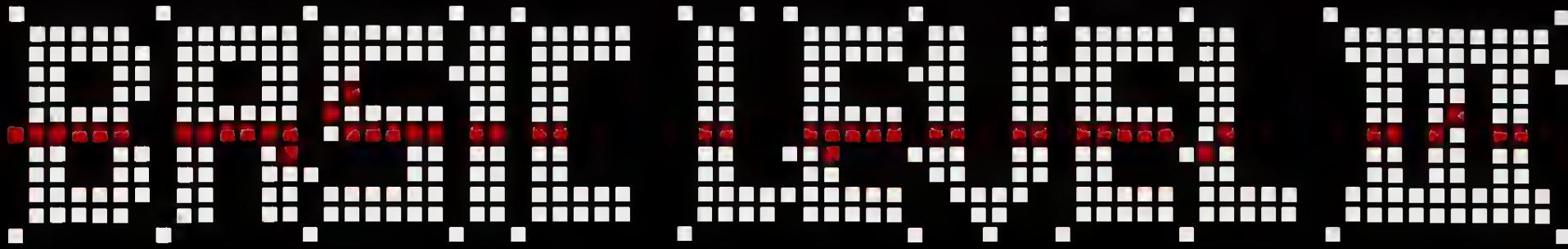


Operating manual for the John Sands Sega SC3000 Personal Computer

John Sands

SEGA®

BASIC LEVEL III

Operator's Manual for the John Sands Sega SC-3000 Personal Computer

John Sands

SEGA

TABLE OF CONTENTS

Published by
John Sands Electronics
Division of John Sands Limited
6 Bay Street Port Melbourne
Victoria 3207 Australia
Telephone (03) 645 3333
Telex AA 34206

First Edition 1983

Copyright © 1983 Sega Enterprises Ltd. All rights reserved.
No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Sega Enterprises Ltd. through John Sands Electronics.

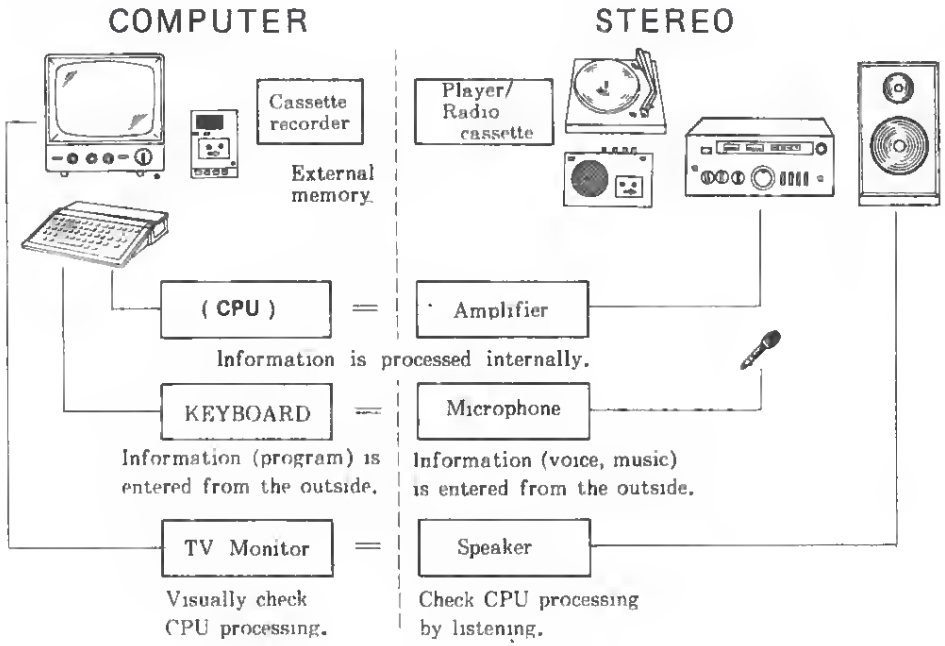
BASIC LEVEL III TEXT

PREFACE

The computer has now reached a level at which anyone can handle it with ease.

What is a computer, then?

Lets's compare computers to stereos.



The above figure shows the computer mechanism. When information (program) is entered from the outside, results are displayed on the TV screen.

Although there are various kinds of program languages available, the BASIC language is the most common language for personal computers.

Some of the things we can do by using the BASIC language are:

1) Computing
2) Filing of statements and data.
3) Drawing of patterns and graphics.
4) Enjoying games and music.

It is to your advantage to become familiar with the BASIC language so that you will be able to get the most out of your computer.

The term "language" sounds difficult, however, the BASIC language does not have too many commands to be remembered. You can write programs using only a few of these commands, and as you become more confident you can begin using more commands.

Firstly, operate the keys while referring to the text. You will probably find some errors. Do not worry about errors, persevere, and continue to operate the keys.

Very soon, you will find that the computer will become a most easy–going and reliable friend.

Chapter 1. How to Handle the Computer

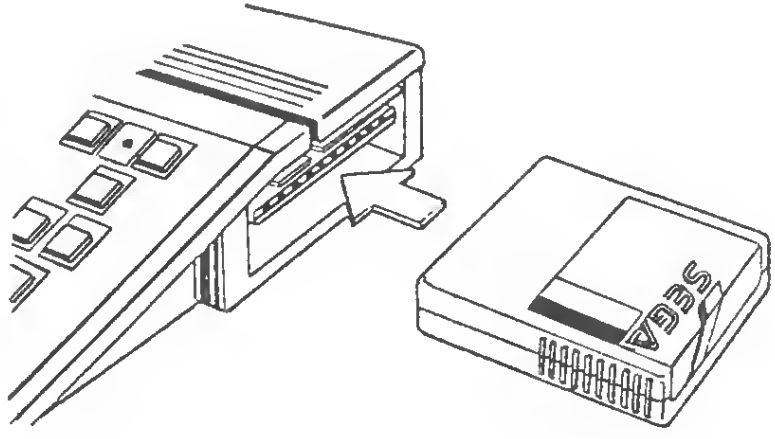
First, read the instruction leaflet contained in the SC – 3000 unit.

1. Be sure that switch box is connected to the TV. (Where a TV with video input is used, directly connect the computer to the video input terminal and audio terminal).

2. Provide a switch box near the computer and select the TV channel, CH 3 or CH 4.

3. Select the proper channel shift switch of the computer, either CH 3 or CH 4, whichever is unoccupied.

4. Insert the BASIC cartridge correctly.



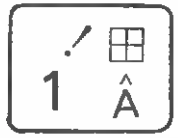
5. After connections are complete, check the cable connections.
   If cable connections are correct, turn the TV power on.

   Then, also turn the computer power on.

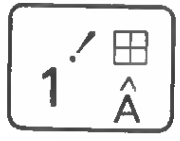
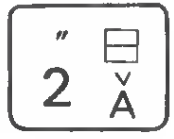
How to Use the KEYBOARD

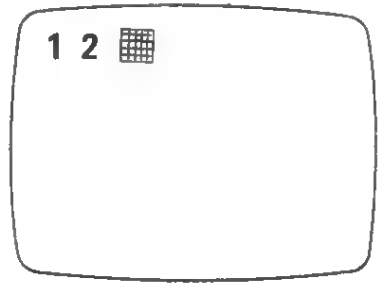
The KEYBOARD has keys on which letters, numerics, Dieresis characters (foreign language) and symbols are written.

Some keys have 4 characters or displays.

Example : 

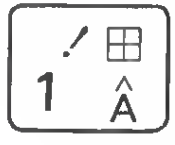
Key layout and spacing are the same as in typewriters.
So, you can push the keys with your fingers easily.
Try to press the keys first.

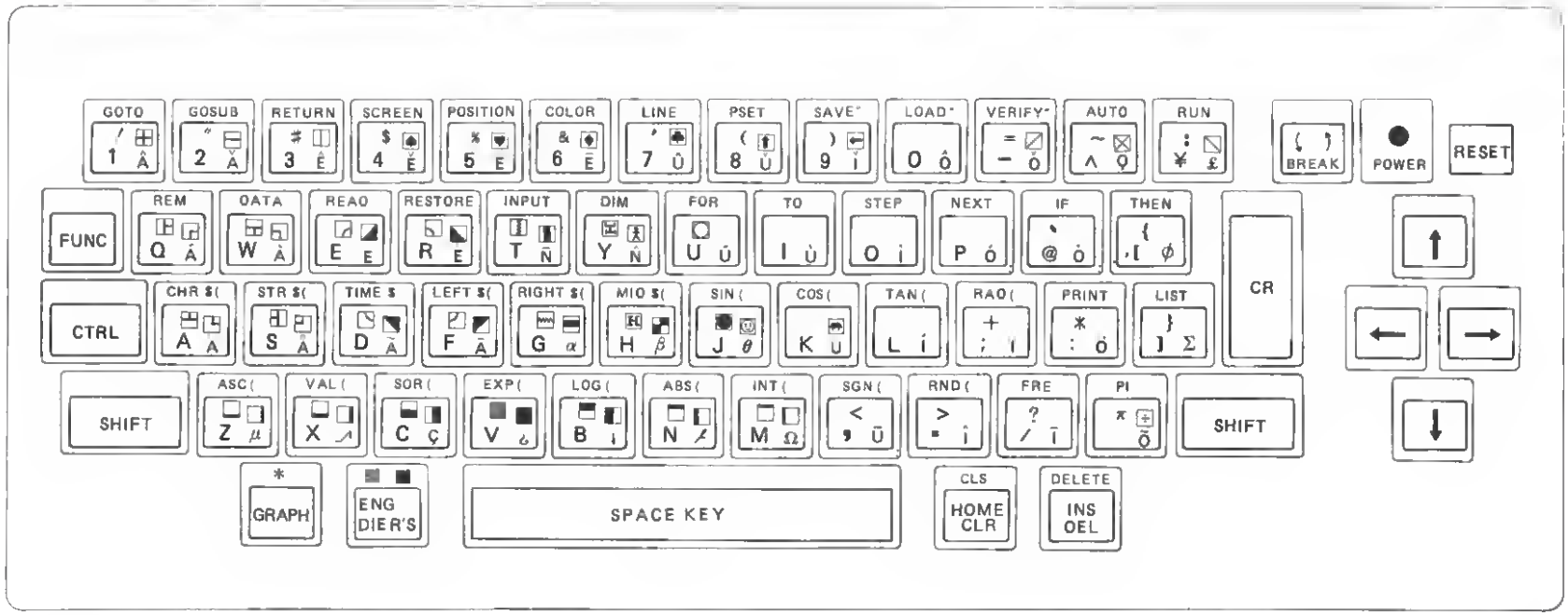
Push keys of  and then 



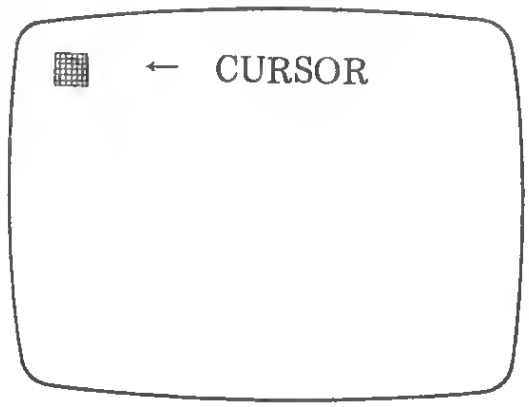
Now, you should have 12 displayed on the screen. If you press keys by themselves the lower left characters or symbols written on the keys will be displayed on the screen.

To display on the screen characters and symbols other than the above, use [SHIFT] key, [GRAPH] key or [DIER'S] key.

When [SHIFT] key is held down and then the  key is pressed ! appears on the screen.


GOTO
GOSUB
RETURN
SCREEN
POSITION
COLOR
LINE
PSET
SAVE"
LOAD"
VERIFY"
AUTO
RUN
BREAK
POWER
RESET
FUNC
REM
DATA
READ
RESTORE
INPUT
DIM
FOR
TO
STEP
NEXT
IF
THEN
CR
CTRL
CHR$(
STR$(
TIME$
LEFT$(
RIGHT$(
MID$(
SIN(
COS(
TAN(
RAD(
PRINT
LIST
SHIFT
ASC(
VAL(
SOR(
EXP(
LOG(
ABS(
INT(
SGN(
RND(
FRE
PI
SHIFT
GRAPH
ENG
DIER'S
SPACE KEY
CLS
HOME
CLR
DELETE
INS
OEL

CURSOR



▦ is blinking in the upper left portion of the screen. This is called the CURSOR and shows the position where characters and symbols which were entered from the keys are displayed.

To move the CURSOR, use the 4 light grey keys with arrows on them.

The type of the CURSOR varies depending on respective modes.

Alphanumeric mode
Dieresis mode } ..... ▦

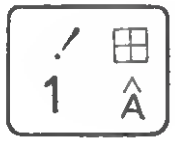
Graphic mode ..... *

Press [ENG/DIER'S] or [GRAPH] keys which are in the lower left hand side of the KEYBOARD.

The CURSOR will change. To return the CURSOR to the original position from graphic mode, press the [GRAPH] key again.

To display on the screen the characters and symbols which are written on the key surface, there are five methods available.

2 SHIFT
+
Alphanumerics



GRAPH 3

1 Alphanumerics

Dieresis 4

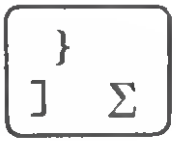
5 SHIFT
+
GRAPH

GRAPH

Alphanumerics

Dieresis

SHIFT
+
Alphanumerics



Dieresis

[SHIFT] (Shift key)

Right and left [SHIFT] keys work the same way. While holding down a [SHIFT] key, when the key with numerics is hit, the symbol to the top left of the key is entered.

While holding down the [SHIFT] key, when an alphabetic character key is hit, a small letter is entered.

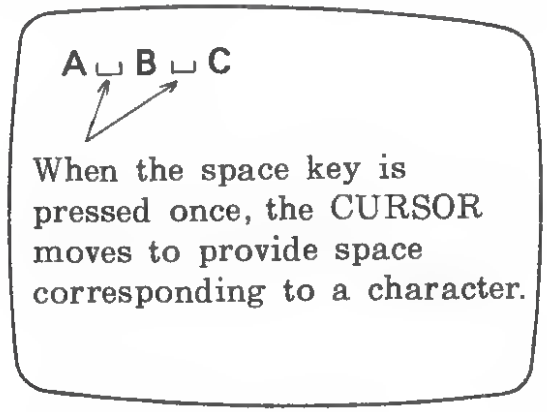
[GRAPH] (Graphic key)

This is used to input graphic symbols. The CURSOR shifts to *.

Graphic symbols may be used together with the [SHIFT] key.

This is used when entering dieresis.
Dieresis character can be typed while the Dier's key is held down.

SPACE key (This allows space between characters and symbols)



The elongated key (in the bottom row) is the space key which inputs space(s) between characters and symbols.

In computers, space is also handled as information, as in the case of characters.

Input various characters using previously mentioned keys.

Special Keys

Now, you notice that the screen is full of characters and symbols which were previously entered by keys.

It's no use leaving unwanted characters and symbols on the screen.
We can clear the whole screen, using the following key.

[HOME/CLR] (Home/Clear)

When this key is pressed, characters on the screen are erased and the CURSOR returns to the upper left "home" position. Use this key whenever you want to clear the screen.

When [HOME/CLR] key is pressed while holding down the [SHIFT] key, the screen will remain uncleared but the CURSOR returns to the home position.

[CR] (Carriage Return) or (Return)

In computers, even if characters are on the screen these are not stored inside the computer until the [CR] key is pressed.

Input any character and press the [CR] key. You notice a Syntax Error displayed on the screen.

Instructions for computers need to be written a certain way.
This is called the computer Syntax. If this Syntax is wrong, errors will occur.

For errors, refer to the error message table in the appendix.

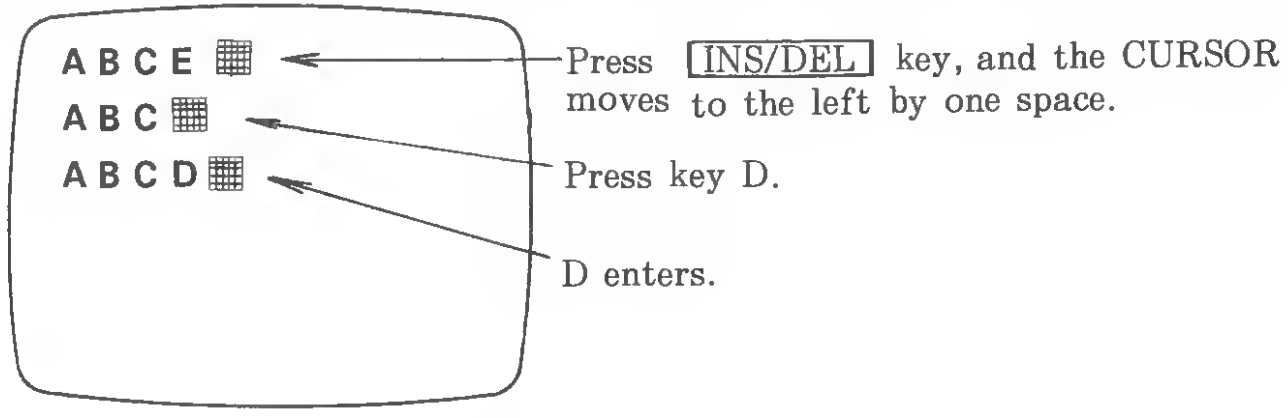
[INS/DEL] (Insert/Delete)

The [INS/DEL] key is used when deleting or adding characters one by one.

INS (Insert) refers to the addition of characters.

DEL (Delete) refers to the deletion of characters.

When [A] [B] [C] [E] is typed by mistake instead of
[A] [B] [C] [D] the CURSOR will move backward by one character if the [INS/DEL] key is pressed, and then character E is erased. Here, press D and a correction has been made to A B C D.



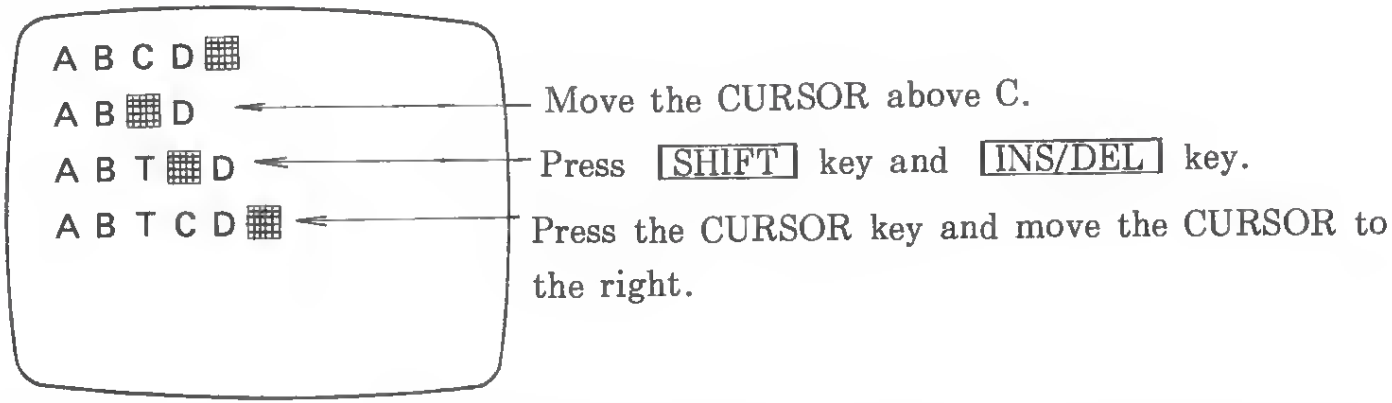
Now, let's put any character in between the B and C of A B C D.

Bring the CURSOR over the C of A B C D.

While holding down the [SHIFT] key, press the [INS/DEL] key.

You notice that the CURSOR blinks quicker than before. Input any character.

The character which was entered just now enters after B and C D moves to the right by one character space.



When the insert mode is used, as many characters as required can be entered (while the CURSOR is blinking quickly).

When returning the insert mode to the original state :

1. Press [CR] key.

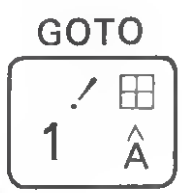
2. Press a CURSOR control key, (one of the light grey keys with arrows).

3. Press [SHIFT] + [INS/DEL] key.

By pressing either one of the above keys, the normal condition is restored. When the program is corrected, the MEMORY inside cannot be rewritten unless the [CR] key is pressed. Confirm this by actually operating the keys.

FUNC (Function)

This key allows you to enter many of the common BASIC words with a single keystroke.



The key has GOTO written on the upper part of it.
Each key has alphabetic characters written on it.

This is a command statement used in BASIC. While holding down the [FUNC] key, hit any key with a word above it, and the command statement written on the upper part of the key is entered.

This is a useful feature when typing in programs.

[⤵⤴/BREAK] (Screen shift/break)

This is used for stopping programs during program run.

The screen will change when [⤵⤴/BREAK] key is pressed while holding down the [SHIFT] key, by pressing the keys again the screen will change back.

This is used for changing the screen, as the computer has two screens. One screen is text screen for entering programs, and the other, graphic screen for displaying graphics.

[⤵⤴/BREAK]

⤵⤴ ➡ This is used to shift screens.
Use this while holding down [SHIFT] key.

This is because the computer has 2 screens as mentioned above.

BREAK ➡ This is used for stopping program run while they are running.

CTRL (Control)

Movement for which explanation is given below can be executed when the character key shown in the control key table is hit, while pressing the CTRL key.

CONTROL CODE

Key operation	PRINT CHR$ (Value)	Functions
CTRL + A	PRINT CHR$ (1) ;	NULL No character
C	——	BREAK Stops program run
E	5	Clears Characters after CURSOR
G	7	BELL Makes "beep" sound
H	8	DEL Deletes characters
I	9	HT Horizontal TAB
J	10	LF Line feed
K	11	HM Returns CURSOR to home position
L	12	CL Clears screen
M	13	CR Carriage Return
N	14	Dieresis ↔ Alphanumeric shift
O	15	{ } Screen shift, text ↔ graphic

Key operation	PRINT CHR$ (Value)	Functions
P	16	Standard character size
Q	17	Character size, horizontally 2 times as large (graphic) (Screen 2)
R	18	INS (Insert)
S	19	Key input (A~Z) no shift, capital letter
T	20	Key input (a~z) no shift, small letter
U	21	Clears lines and returns CURSOR to left head
V	22	Normal mode
W	23	GRAPH key input graphic mode ↔ alphabetic character shift
X	24	Click sound ON ↔ OFF shift
–	28	➡ CURSOR movement
–	29	⬅ CURSOR movement
–	30	⬆ CURSOR movement
–	31	⬇ CURSOR movement

When the control code is used in the program, input PRINT CHR$.

[RESET] (Reset)

During program run, or when problems appear on the screen, the screen returns to the situation as it was when the power was turned on within about 1 or 2 seconds after pressing the [RESET] key.

When pressing this key, the computer stops processing and the size of the MEMORY which has not been used is displayed.

```
×××  Bytes  Free
```

Even if the key is pressed, programs which were entered will remain in the memory.

Now that you know how to use the keyboard, you should now be able to put any symbol on the dark grey keys onto the screen.

```
10  SCREEN  2,2:CLS  [CR]
20  LINE  (50,50)-(150,150),5  [CR]
90  GOTO  90  [CR]
RUN  [CR]
```

RUN tells the computer to do what the program says.

(For 0, press O in the top row. O is used for numeral O)

After typing one line, press the [CR] key. Entered programs are stored in the computer and the CURSOR moves on to the line below.
Although using the keys may be difficult, try to hit character keys carefully one by one.

This program will continuously put a box on the screen changing the color of the box.

```
10 SCREEN 2,2 [CR]
20 CLS [CR]
30 LINE (80,100)-(150,170),C,BF [CR]
40 C=C+1 [CR]
50 IF C=16 THEN C=0 [CR]
60 GOTO 20 [CR]
RUN [CR]
```

To stop it push

[(→)/BREAK]

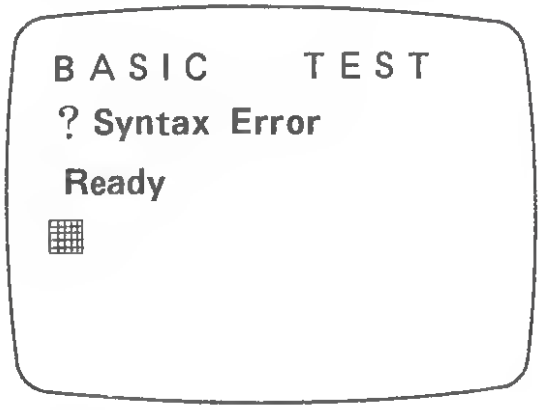
Chapter 2. Using the Computer

DIRECT MODE (Direct Command)

This is to show you how to make the computer work without writing a program.

Let's try to print something on the screen.

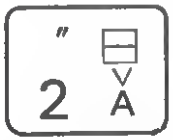
[HOME/CLR] [B][A][S][I][C] [SPACE] [T][E][S][T] [CR]



On the screen, the above will be displayed.

ERROR is displayed because the computer cannot understand the command which was entered. Now, you want the computer to write BASIC TEST on the screen. To do this, give the computer a command to [PRINT] the character which was entered.

The command for "write" is expressed as [PRINT] .



To get the quotation mark " press this key while holding down a shift key.

[HOME/CLR] [P] [R] [I] [N] [T] ["] [B] [A] [S] [I] [C] [SPACE] [T] [E] [S] [T] ["] [CR]

PRINT statement is a command statement for display on the screen.

```
PRINT"BASIC  TEST"  <-- Characters entered by the keys.
BASIC  TEST         <-- Characters which were output by PRINT
Ready                   statement.
```

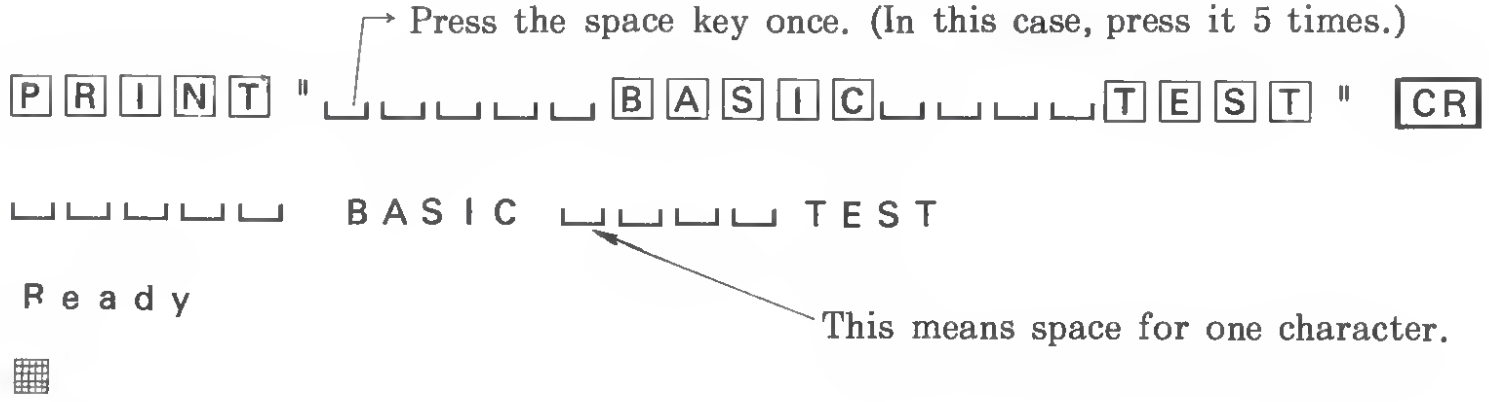
Hopefully this time, no ERROR will be displayed because the computer understood what it was supposed to do.

When printing characters and symbols, use the PRINT statement.

When telling the computer to print characters and symbols by PRINT statements, be sure to put " (double quotation marks) at the start and end of what you want printed.

Now, PRINT your name. Use alphabetic characters or Dieresis characters. All the numerics, symbols and graphic symbols can also be entered in " ".

Spaces in " " are printed too.



PRINT statement is a command statement very often used in programming to show what the computer has been doing.

For numbers and equations the above quotation marks are unnecessary.
eg. PRINT 3 , 3 * 4.
Now, try printing in a different way.

[?] [2] [+] [4] [CR] ? Can be used instead of PRINT.

6

Ready

We gave the computer a command to calculate 2 + 4 and then PRINT the answer.

Let the computer display the calculation results also by a PRINT statement.

Symbol ? is a shortened form of PRINT statement. This is true of most personal computers.

OPERATION OF THE FOUR RULES OF ARITHMETIC.

In computing, some of the symbols used are different from those normally used.

Symbol used in the computer			Normal Symbols
Addition	+	Plus	+
Subtraction	−	Minus	−
Multiplication	*	Asterisk	×
Division	/	Slash	÷
Raising to Power	∧	.....	X^n

A symbol used in computing is called OPERATOR.
Other than the operator, parentheses ( ) (brackets) are also used in numerical expressions.
RELATIVE OPERATOR is used for the comparison of numeric values (numeric magnitude).

The following table summarizes the above. Be sure to read the table before application.

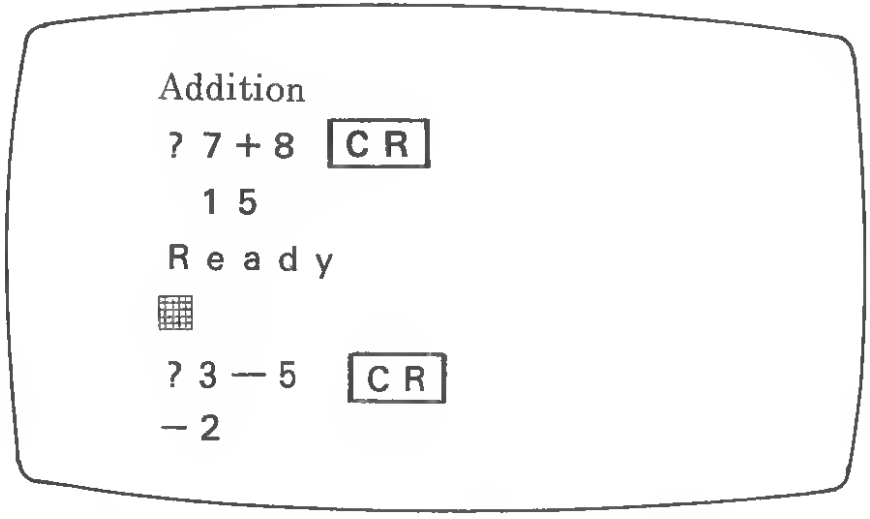
Operator

	Symbol	Where usable: Numeric variable	Where usable: String variable	Description	Priority
Arithmetic operation	∧	○	×	Power (0 ∧ 0 = 1)	1
	+	○	×	Code +	2
	−	○	×	Code −	
	*	○	×	Multiplication	3
	/	○	×	Division	
	MOD	○	×	Residual	4
	+	○	○	Addition (Character combination in case of string variables)	5
	−	○	×	Subtraction	

- ( ) is given the first priority.
- Where more than 2 operators with the same priority are used, the left side operator takes precedence.
- An addition symbol "+" used for string variables shows a linkage.
  (Example) "AB" + "C" ⇨ "ABC"
  Arithmetic operation is decimalized.
  (Example) : Even with values of 0.01, no cancelling in digit occurs.
  Logical operation is binary.
  Arithmetic operation : Decimal 12 digit calculation, 11 digit display.

Notice symbol ÷ in the lower right-hand side of the KEYBOARD.
This is a graphic symbol and unusable for calculations.

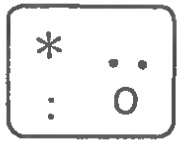
Let's try calculating again.




```
PRINT 5 ∗6
 30
```

↑
└ Where the answer is positive (plus), symbol + is omitted, resulting in one empty character space.

```
PRINT10/3
 3.3333333333
```

For PRINT, [FUNC] key and 

can also be used to make it easier.

```
?INT((3∧4)+0.1) [CR] means (3×3×3×3)
 81
```

Operation functions allow highly accurate decimal calcualations with 11 digit display.

```
?1000000*10000 [CR]
 100000000000
```
11 digits

```
?10/3
 3.3333333333 [CR]
```
10th decimal place

For numbers greater or smaller than the above, the scientific notation system is used.

```
?19380000000*10000000
 1.938E+16  (1.938×10^16)
```

Calculation priority

```
? 6 + 2 * 4
```

In calculations in which two numerical expressions are contained, are the calculations done from the beginning position ? Priority applies to the four rules of arithmetic. Priority is as shown in the table of operators.

Let's run through an example again.

```
? 6 + 2 * 4
  1 4
```

Note that calculation for multiplication is done first.
For calculating addition first, use ( ).

```
? ( 6 + 2 ) * 4
    3 2
```

When addition, subtraction, multiplication and division are involved in the formula in one line, use ( ) for the expression to be calculated first. ( ) can be multiply used more than once in an equation, but brackets and braces as in mathematics are not used.
Only parentheses are used.

```
   ④      ①    ③    ②
? 3＊(（8＋6）／（4－2））
  21
```

Calculations of the above example are done in the order of the sequence number given. If the priority is the same, calculation starts from the left side expression.

When using parentheses, the number of the left side parentheses and that of the right side parentheses need to be the same. If the number is not the same, errors will occur. Be sure to check the number of parentheses.

ANOTHER WAY TO USE THE PRINT STATEMENT.

At the beginning of this chapter we printed letters or symbols enclosed by " ". When this is used in the calculation formula, what will happen ?

Be sure to insert this.

```
? " 2 + 3 = " ; 2 + 3
2 + 3 =  5
```

Previously, only the answer was displayed. But this time, the equation was also displayed. The upper formula of the above is the combination of two statements. " 2+3 " = was handled as characters and not as a calculation formula. Thus, it is displayed as it is. After breaking by ; (semicolon), the statement thereafter was handled as a formula of 2+3, and thus, the answer was displayed.

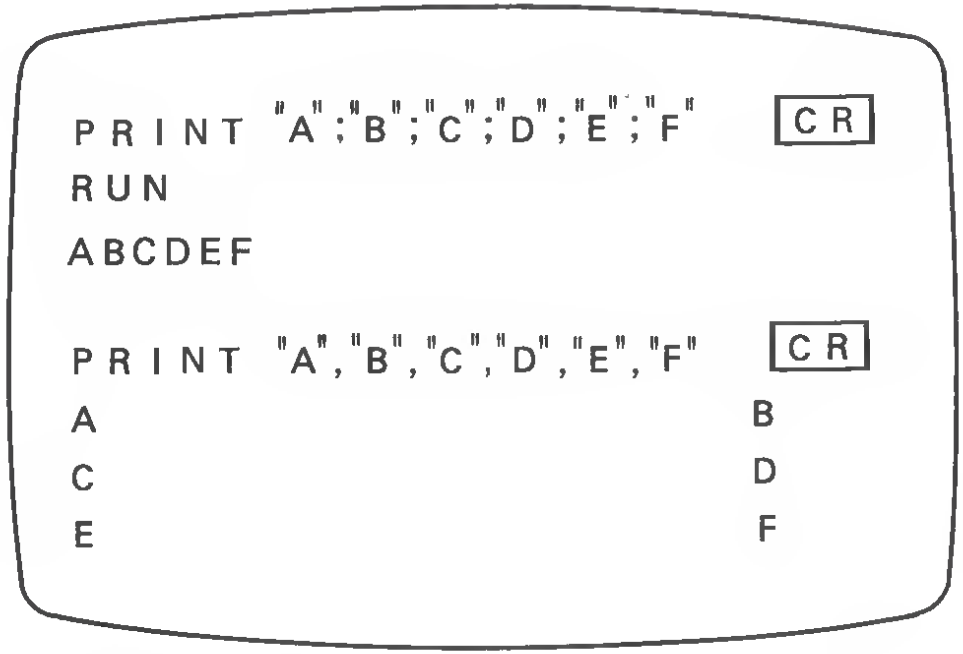
Another example :

```
? " 2 + 3 = " , 2 + 3
  2 + 3 =                    5
  R e a d y
```

The answer is given far apart on the screen. When a comma, (,) is used, the answer displayed is found in the position 20 digits away from the end of the screen.

When using PRINT statement, pleasing displays can be obtained by properly using semicolon and comma.

How to use (,) and (;) in PRINT statement.



So far, you have learned how to make the computer do arithmetic.

Notice that the equal (=) symbols which is normally used in mathematics is not displayed in the calculation formula. In computers, symbols are used in a different way. This will be explained in the next chapter, How to Program.

Chapter 3. How to Program

Let's generate programs by using BASIC. Programs which were entered from the KEYBOARD are stored into the MEMORY inside the computer.
The computer works from the smallest line number to the greatest line number. After line number, there are statements to let the computer know how to perform its task.

LET

(Example)

```
10 LET A=3                    CR
20 LET B=5                    CR
30 LET C=A+B                  CR
40 PRINT C                    CR
50 END                        CR
```

R U N CR

8

Ready

A new statement RUN is displayed. This is a command statement to make the computer do whatever the program tells it to.

LET , VARIABLES

LET (substitution statement) is a command statement used for giving a number value for a variable.

```
10 LET A=3
```

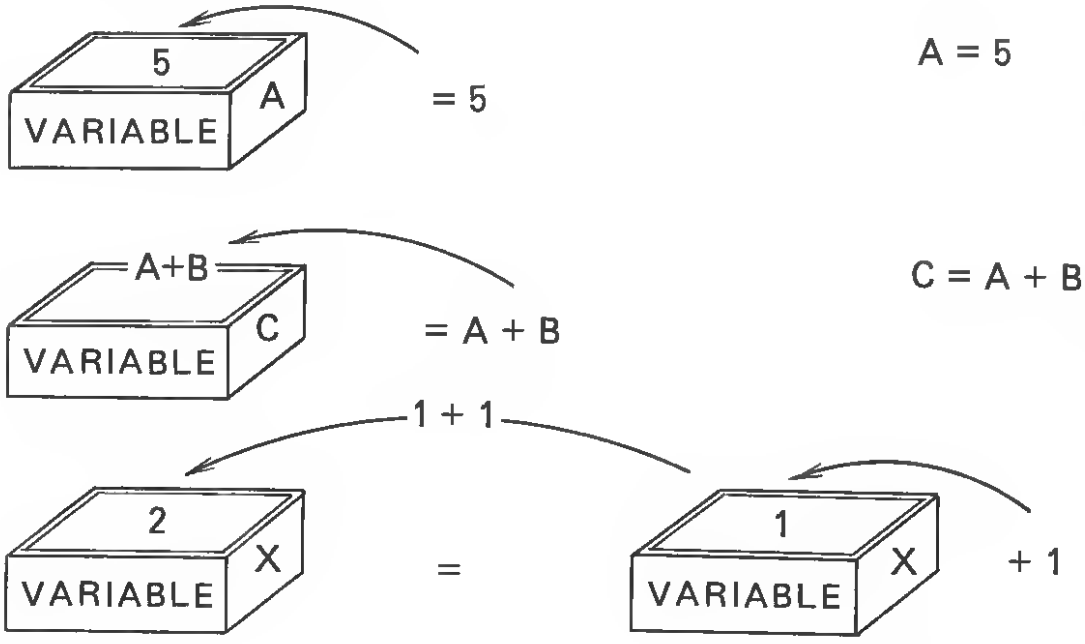
The above A could be thought of as an empty box called A.
We call this box a variable. This expression doesn't mean that A equals 3.
It means putting the number 3 in Box A. The LET statement is optional.
Therefore, the following can be entered.

```
10 A=3
```

For the following, LET is also omissible.

```
30  LET  C=A+B
```

Symbol equals (=) doesn't mean to be equal, but refers to putting a number in a certain place.



A = 5

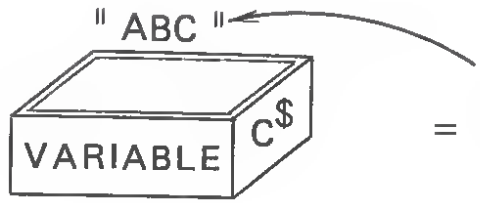
Input 5 in A.

C = A + B

Input A + B in C.

X = X + 1

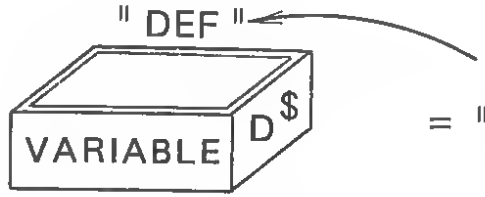
Substitute the result of X + 1 for variable X.



= " ABC "

C $ = " ABC "

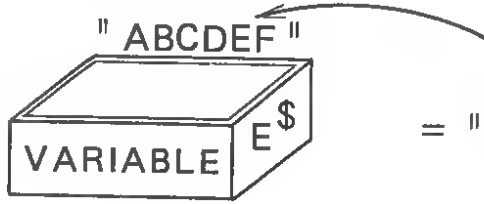
Substitute string ABC in C$.



= " DEF "

D $ = " DEF "

Substitute DEF for D$.



= " ABC " + " DEF "

E $ = C $ + D $

Substitute ABC + DEF for E$.

Substitution statement can be used as follows.

X = X + 1

This is not true mathematically, but qenerally used in the substitution statement.

Example

```
10  CLS
20  X=X+1
30  PRINT  X;
40  GOTO  20      ← Line No.
RUN
  1  2  3  4  5  6  7  8 . . .
```

Numbers consecutively appear every other character.

X starts with 0.

With $X^{(0)} = X^{(0)} + 1$, 1 is substituted for the left side X. After jumping from line No. 40 to No. 20, RUN returns to $X^{(1)} = X^{(1)} + 1$ and with 1 added to X, the left side X becomes 2. In computers, the above-mentioned applications are often adopted.

String variables

Attach $ (dollar) mark to string variables.

```
10  A=3
20  B=5
30  C=A+B
40  M$="ANSWER␣"
50  PRINT  M$;C
RUN
ANSWER   8
Ready
```

␣ represents space

```
40  M$="ANSWER␣"
```

Provide one empty character space by the space key.

In this way, a character string " ANSWER " was entered in box M$.
For a character string, the left and right side of the character needs to be enclosed by double quotation marks ("). If this is not done errors will occur. Numerics or graphic symbols can be used for string variables. Furthermore, two string variables can be connected.

```
10  A$=" I␣"
20  B$=" AM␣"
30  C$="A  BOY"
40  PRINT A$+B$+C$
50  END
RUN
I  AM  A  BOY
Ready
```

Provide one character space.

CLS, LIST, NEW

Various statements are available for BASIC. At the time of RUN, programs remaining on the screen may be confusing. If a command for screen erasure is entered in the beginning of a program in advance, all displays on the screen are erased at the time of RUN.

Input new statements in the current program.

CLS

```
5 CLS ←
```

command to erase screen display.

Enter line No. 5 CLS below the program which was previoulsy entered, in a place where there is no writing.

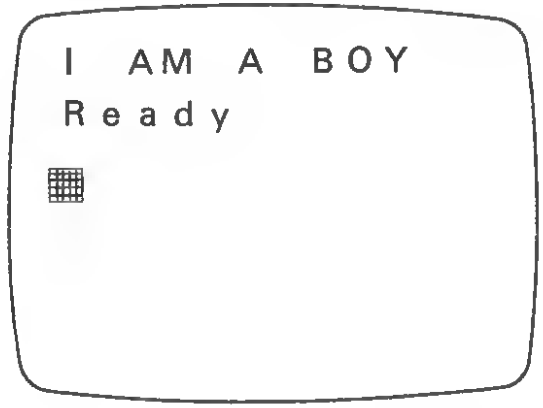
Program input from line No. 1 to 65535 is possible. If programs are generated with closely arrayed line numbers (e.g. 1 , 2 , 3 ,), additions between programs which were previously entered are impossible. Therefore, it is common practice to go up by 10 for each line number so that lines can be put in between other lines without difficulty. Where programs are entered up to line No. 100 (going up by 10), even if unused line numbers such as 25 and 55 are entered after line No. 100, these numbers can be rearranged by the computer internally. Lets check the outcome of the entering of the above-mentioned 5 CLS.

LIST

Enter [HOME/CLR] L I S T [C R]

```
  5  C L S
10  A $ = " I   "
20  B $ = " AM   "
30  C $ = " A  B O Y "
40  P R I N T  A $ + B $ + C $
50  E N D
```

Note that line No. 5 is entered at the beginning. Now, start run.



LIST is a command statement to make the computer display programs which were entered. How to use LIST is as follows.

LIST commands are used as follows :

LIST	Displays the entire contents of programs.
LIST LINE NO.	Only one line is displayed.
LIST LINE NO. – LINE NO.	Line No. to line No. is displayed.

LIST LINE NO. –	The content of programs after line No. is displayed.
LIST – LINE NO.	This displays from the beginning of the program to line No.

The content of the program displayed by LIST statement can be rewritten using the CURSOR.

```
LIST 30
30 C$="A BOY"
```

Move the CURSOR to the place above B and input M. Subsequently, input AN and press [CR] . After rewriting the content of the program, be sure to press the [CR] key. If you forget to press the [CR] key, the content of MEMORY does not change even if characters on the screen do. When the [SPACE] key is pressed during the list display, the list display is halted. For immediate correction of the program, press the [BREAK] key. When the [SPACE] key is pressed during LIST display, the display restarts.

NEW

When entering a new program after finishing one program, if the preceding program remains intact in the MEMORY, the new program may not work normally.

The program which was previously entered cannot be erased by CLS statement from the inside of the MEMORY. To erase programs, input NEW and press [CR] .

* Let's display LIST.

```
LIST [CR]
Ready
```

Nothing is displayed. All programs were deleted from the MEMORY. From explanations given so far, you have learnt something about computer programs. While entering programs, errors in typing and statements may occur until you become familiar with programming. When starting RUN with such errors remaining, execution stops at the line No. which has such errors. This sort of an error is called a BUG. As you know, BUG refers to insects. So, this is called a worm-eating problem. Thus, correction of this is called DEBUG. Although BUG finding is easy with a short program, it is not so with a lengthy program. So, be careful when entering programs.

INPUT, GOTO

Let's write calculation programs.

In the calculation program written at the beginning, the values of variables were set in the program, so numbers to be calculated had to be corrected each time.

- Let's write calculation programs which are consecutively usable.

```
10  CLS
20  INPUT  A
30  INPUT  B
40  C=A+B
50  PRINT  C
60  GOTO  20  <------ Line No. to which execution jumps.
RUN   [CR]
```

```
? ▦  (WAITING FOR INPUT A) [4] [CR] — INPUT 4
? ▦  (WAITING FOR INPUT B) [5] [CR] — INPUT 5
  9
? ▦ [8] [CR]
? ▦ [7] [CR]
  1 5
? ▦            Here, press [BREAK] key.
B R E A K   I N   2 0   RUN was stopped at
R e a d y               line No. 20.
▦
```

GOTO

When computer's operation flow encounters GOTO statements, RUN unconditionally jumps to the assigned line No. This program returns to line No. 20 from line No. 60 and restarts from INPUT A. This sort of program is repeated endlessly and thus it's called an infinite loop. The only way the program is stopped is by the [BREAK] key.

When the program flow encounters INPUT statement, values are entered in variables from the KEYBOARD and the flow is halted until the [CR] key is pressed.

INPUT statement is also applicable to string variables. Comments enclosed by " " can also be displayed, as in PRINT statements.

```
NEW
10  CLS
20  INPUT "NAME ?␣" ; A$
30  PRINT  A$
40  END
RUN
NAME ? HANAKO  ← Input character
HANAKO
Ready
▦
```

END, STOP

END informs you of the program end.
STOP halts program flow.

CONT

This is used when restarting the program which was interrupted by STOP statement and the [BREAK] key.

```
10  X=X+1
20  PRINT  X
30  GOTO  10
```

Run this program, and halt run during operation by the [BREAK] key.

```
Break  in  20
CONT  [CR]
```

In this way, the program restarts from the position the program was halted.

FOR – TO, NEXT, STEP

These are used to make the computer do its work repeatedly for a specific number of times.

```
10  CLS
20  FOR  N=0  TO  9
30  PRINT  N
40  NEXT  N
50  END
```

Repeats 10 times.

FOR – TO is used together with NEXT as a pair. In this program, N increases one by one from 0 to 9.

```
10  CLS
20  FOR  N=0  TO  20  STEP  2
30  PRINT  N;
40  NEXT  N
50  END
RUN
 0   2   4   6   8   10   12   14   16   18   20
```

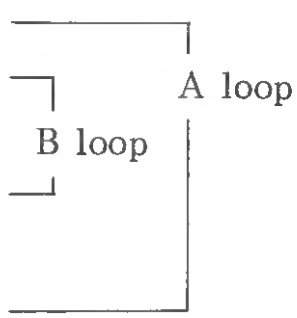
With step 2, increases from 0 to 20 take place by an increment of 2.

The – (minus) symbol can also be used for STEP. Let's change line No. 20.

```
20 FOR N=20 TO 0 STEP-2
RUN
20 18 16 14 12 10 8 6 4 2 0
```

From 20, decreases take place by an increment of 2. In this way, STEP is used when increasing or decreasing a specific number at one time. FOR – NEXT statements can be nested.

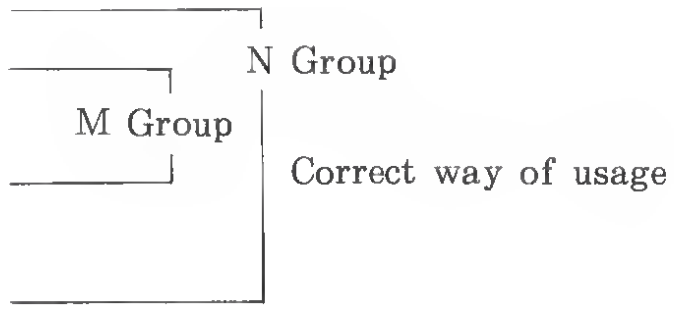
```
10 CLS
20 FOR A=1 TO 9
30 FOR B=1 TO 9
40 PRINT A*B;
50 NEXT B
60 PRINT
70 NEXT A
```



The multiple use of FOR – NEXT statement is called "nesting". Multiple nesting up to 16 levels is possible. If the specified nesting is exceeded, nesting errors will occur.

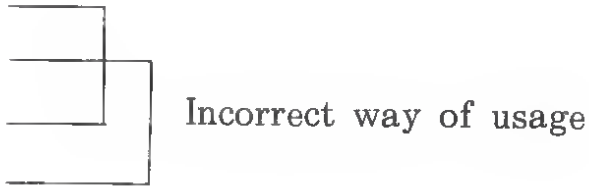
Variables such as FOR I = 1 TO N can also be used.

```
10  CLS
20  FOR  N=1  TO  20
30  FOR  M=1  TO  N
40  PRINT"O";
50  NEXT  M
60  PRINT
70  NEXT  N
```



Cautions for nesting

```
FOR  N=1  TO  20
FOR  M=1  TO  10 .....
NEXT  N
NEXT  M
```



FOR, NEXT groups cannot be intersected.

IF – THEN, GOSUB

Program flow sequentially proceeds from the side with the smallest number. When reaching a given situation, let's try to change the flow.

("ACCEPTABLE" OR "UNACCEPTABLE" PROGRAM)

```
10  CLS
20  INPUT "SCORE" ; A
30  IF  A>=65  THEN  GOSUB  100
40  IF  A<=65  THEN  GOSUB  200
50  GOTO  20
100  PRINT "ACCEPTABLE"
110  RETURN
200  PRINT "UNACCEPTABLE"
210  RETURN
```

IF – THEN is a command to analyze a situation. The above program analyzes the SCORE entered by INPUT statement, evaluating by IF statement and changes the program flow. If the score entered is more than 65 points, the program goes to line No. 100 by GOSUB statement. If it is less than 65 points, operation goes to line No. 200.

```
IF  A>=65  THEN  GOSUB  100
```

This statement means : IF A is greater than 65, THEN, jump to the subroutine beginning at line No. 100 and return.
Following IF – THEN, statements other than line No. can also be used.

(Omissible)

```
IF~THEN GOTO     LINE NO. (Jumps to the assigned line No.)
IF~THEN GOSUB    LINE NO. (Jumps to the assigned line No.)
IF~THEN PRINT"×××"   (Enters on the screen)
IF~THEN END      (Ends program)
IF~THEN STOP     (Stops program run)
IF~THEN BEEP     (Produces sound)
```

To analyze a situation, use relative operators shown in the table below.

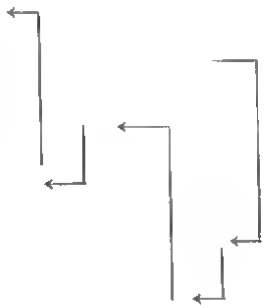
Table of Relative Operators

	Symbol	Description
Relative operation	=	Equal to (−1 for true, 0 for false)
	<>	Not equal to (−1 for true, 0 for false)
	>	Greater than (−1 for true, 0 for false)
	<	Less than (−1 for true, 0 for false)
	>=	Greater than or equal to (−1 for true, 0 for false)
	<=	Less than or equal to (−1 for true, 0 for false)
Logical operation	NOT	Logical denial
	AND	Logical product
	OR	Logical sum
	XOR	Exclusive OR

GOSUB, RETURN

The preceding program had a GOSUB statement. With GOTO statements, the program only goes to the assigned line No. GOSUB statement, however, is used together with RETURN statement. After jumping to the assigned No., operation returns to the line following GOSUB statement by RETURN statement.

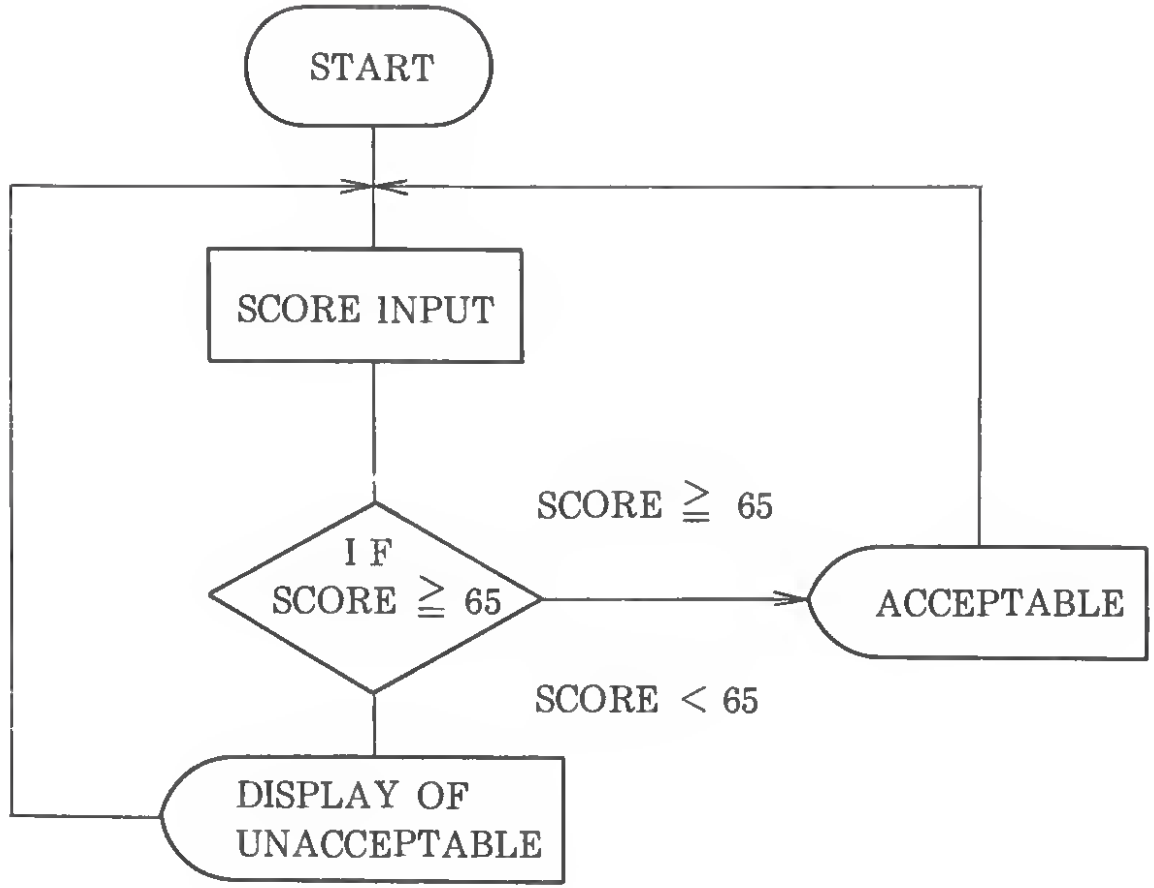
```
20  INPUT A
30  IF A>=65 THEN GOSUB 100
40  IF A<65 THEN GOSUB 200
50  GOTO 20
100  PRINT "ACCEPTABLE"
110  RETURN
200  PRINT "UNACCEPTABLE"
210  RETURN
```



When using GOSUB statement, errors will occur if you forget to enter RETURN STATEMENT.

After operation's returning by the RETURN statement, the program proceeds from the line following GOSUB. So, change the program flow again as in the case of line No. 50.

Where the program branches midway through, a flow chart is prepared so that the program flow can be easily understood.



In the flow chart, the flow proceeds from the top downwards. The flow changes by the situation analyzing statement. When generating complicated programs, the flow can be clearly understood by preparing the flow chart.

ON GOTO

ON GOTO statement is used similiar to the conditional statement.

```
ON A GOTO 100,200,300
```

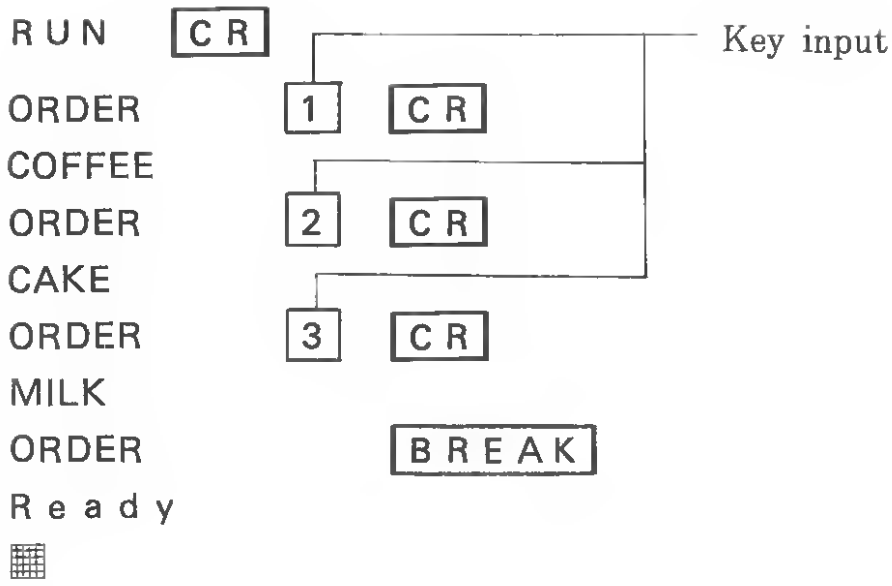
Where variable A is 1, the program jumps to line No. 100.
Where variable A is 2, it jumps to line No. 200.
The values of variables corresponding to the number of line No. following GOTO statement can be used.

(For A, input values 1~3)

```
10 INPUT "ORDER"; A
20 ON A GOTO 100,200,300
100 PRINT "COFFEE" :GOTO 10
200 PRINT "CAKE" :GOTO 10
300 PRINT "MILK" :GOTO 10
```

(COLON)

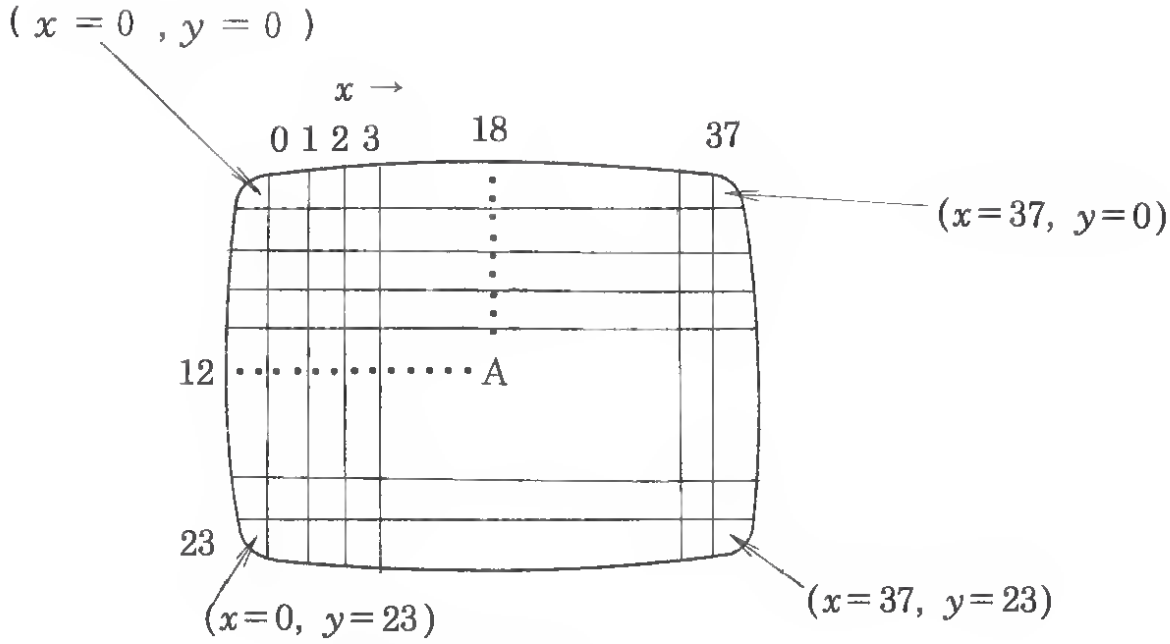
More than 2 command statements can be entered in one line (multi-statement). When entering more than one independent statement in one line, break them by : (colon).



ON GOSUB is also used in the same way. Before giving you an explanation, let's use the new command statements.

CURSOR

Previously, at the time of RUN, you had displays only on the left side of the screen. If programming is done so as to have displays in specific positions on the screen, you can see displays more easily. The CURSOR statement is a statement to define the display positions.



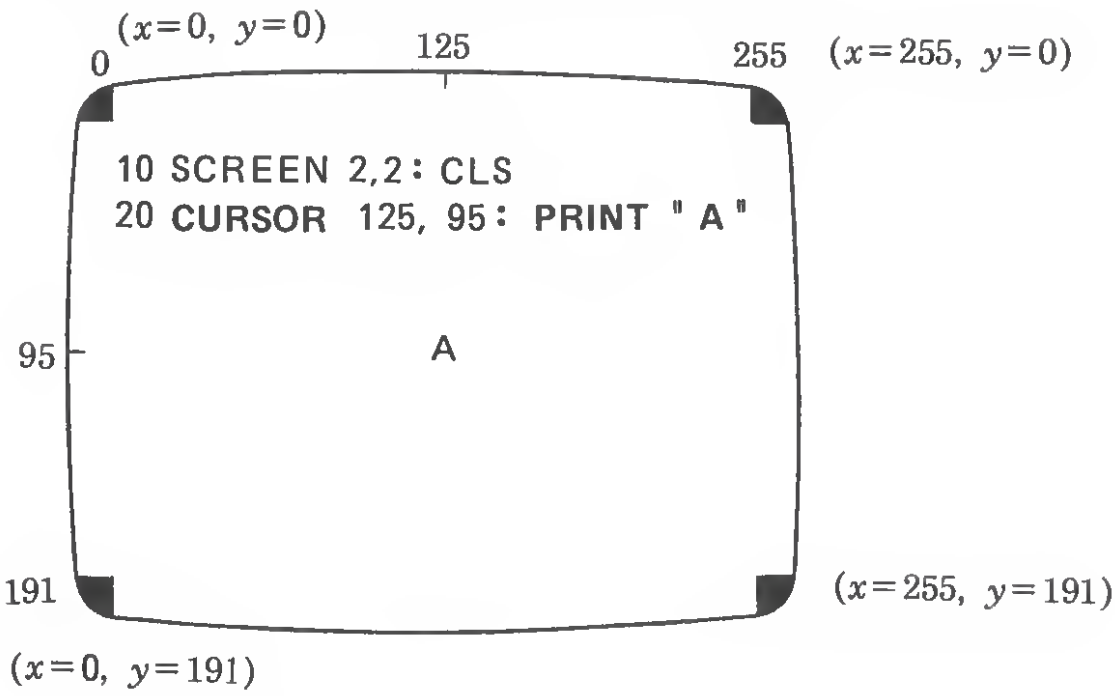
TEXT SCREEN

The text screen on which the program is written consists of 912 locations formed by 38 digits × 24 lines.

```
CURSOR 18,12 :PRINT"A"
```

If you enter it directly without using line number, you will notice that character A is displayed in the central position on the screen.

When using the CURSOR statement on the graphic screen:



x axis direction 0 ~ 255 (256 dots)

y axis direction 0 ~ 191 (192 dots)

The beginning position of characters you want to display is defined if coordinates are assigned by the CURSOR statement.

ON GOSUB

```
10  CLS
20  CURSOR  10,3:PRINT "MENU"
30  CURSOR  10,6:PRINT "1... DRINK"
40  CURSOR  10,8:PRINT "2... FOOD"
50  CURSOR  10,10:PRINT "3... DESSERT"
60  CURSOR  10,13:INPUT "ORDER";A
70  ON A GOSUB 100,200,300
80  GOTO 60
```

Erase previous display.

```
100  CURSOR  10,16:PRINT"                    "
110  CURSOR  10.16:PRINT" COFFEE... $3.00  "
120  RETURN
200  CURSOR  10,16:PRINT"                    "
210  CURSOR  10,16:PRINT" CAKE... $2.00     "
220  RETURN
300  CURSOR  10,16:PRINT"                    "
310  CURSOR  10,16:PRINT" MELON... $2.50    "
320  RETURN
```

ON GOSUB is a form varied from the program with GOTO.
Although only the CURSOR statements are used, displays are in the central positions. So, you can notice displays more easily. When using GOSUB, don't forget to enter RETURN.

READ, DATA, RESTORE

Data previously entered in the program can be read by READ Statement.

```
10  READ  A , B , C , D
20  PRINT  A+B+C+D
100  DATA  1 , 2 , 3 , 4
RUN
 10
Ready
```

The numeric values of DATA were added to display the result.

You can also enter string variables by using READ and DATA statements.

When encountering READ statement, the program flow reads the DATA statement first wherever the statement may be.

```
10  READ  A$,B$,C$,D$
20  PRINT  A$+B$+C$+D$
30  DATA  S,E
40  PRINT
50  DATA  G
60  DATA  A
RUN
SEGA

Ready
```

Provides space for one line.

Where string variables were used, even if numerics are entered in DATA, these numerics are handled as characters and unusable in mathematical calculations.

The number of DATA and that of READ statement variables need to be the same.

When the number of DATA is more than that read, only the DATA corresponding to the variables of READ statement will be displayed.

Where the variables of READ statement exceed the DATA, errors will occur.

When the same data is used repeatedly from the beginning, use RESTORE statement.

RESTORE

```
10  READ  A,B,C,D
20  DATA  1,2,3,4
30  RESTORE
40  READ  E    ← Reads the beginning data 1.
50  PRINT  A+B+C+D+E
RUN
  11
```

DIM (array)

1 – Dimensional Array

In the previous program, variables A, B, C and D were used for the DATA. As the number of DATA increases, it becomes more troublesome to set variables one by one.

In such a case, arrays are used.

DIM A (5) ← Value in ( ) is called a subscript. This means that six variables A(0), A(1), A(2), A(3), A(4) and A(5) were dimensioned.

String variables can also be dimensioned.

```
10  CLS
20  DIM  A$(5),B(5)
30  FOR  I=0  TO  5
40  READ  A$(I),B(I)
50  PRINT  A$(I),B(I)
60  PRINT  ← To provide one line space
70  NEXT I
100  DATA  COFFEE, 250, MILK, 150, CAKE, 200
110  DATA  TEA, 280, TOAST, 180, BREAD , 100
```

Although DIMA $ (5) was entered, READ statement refers to A $ (1). This is because 1 is the one used in FOR 1=0 to 5. Thus, while 1 changes from 0 to 5, DATA is read.

In READ statement, DATA alternatively array string variables and numeric values since A$ and B are consecutively read.

Let's experiment to array the DATA.

2 – Dimensional Array

In 2 – Dimensional array, subscripts in ( ) are divided into two parts, for example, DIMA (9 , 9).

Multiplication Table

```
10  CLS
20  DIM  A(9,9)
30  FOR  J=1  TO  9
40  FOR  K=1  TO  9
50  A(J,K)=J*K
60  PRINT  A(J,K);
70  NEXT  K
80  PRINT  ← To change line
90  NEXT  J
RUN
```

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

3 – Dimensional Array

```
DIM A(5,5,5)
```

Array declaration is possible up to 3–Dimensions. If no array declaration is given by DIM statement, the subscript with the maximum value of 10 is automatically applicable to the declaration.

ERASE

This is used to make array declaration invalid during program run.

When 100 ERASE is entered, all the arrays in the program will become invalid.

Where array name such as 100 ERASE A, B$ is entered, the array will become invalid.

DELETE

DELETE statement is used to delete unwanted lines when correcting programs.

DELETE 180 – 220 [CR] , (comma) may be used instead of – sign.

Line numbers 180 –220 are deleted by the above.

DELETE –250 Deletes programs from the beginning to line No. 250.

DELETE 600– Deletes all numbers from No. 600 onwards.

DELETE 100 Deletes line No.100 only.

So far, some of the BASIC statements have been used. When programming, be sure to enter the line number. The following shows the statement which automatically generates the line number.

AUTO

Enter AUTO [CR] without line No.

```
10 [CR]
20
```

Line numbers are automatically generated by STEP 10.

```
AUTO 100 [CR]
100 [CR]
110
```

Note that the line numbers are displayed starting from No. 100 by STEP 10.

AUTO , 10 , 20 [CR]

(Starting line No. and STEP number)

10 [CR]

30 [CR]

50

Each time [CR] key is pressed, line No. is displayed by STEP 20.

RENUM (Renumber)

When line numbers are too closely arranged while the program is entered and line numbers are added, RENUM is used to renumber line Nos.

RENUM [CR]

Line numbers are renumbered from the beginning of the program in the order of 10 , 20 , 30 and so on.

RENUM 100 [CR]

Correction is made by renumbering with step 10 starting from line No. 100.

```
RENUM  300 , 200
       |     |
       |     └ Previous No.
       └ New No.
```

The program is renumbered by step 10 starting from the previous line No. 200 which is to be renumbered as line No. 300

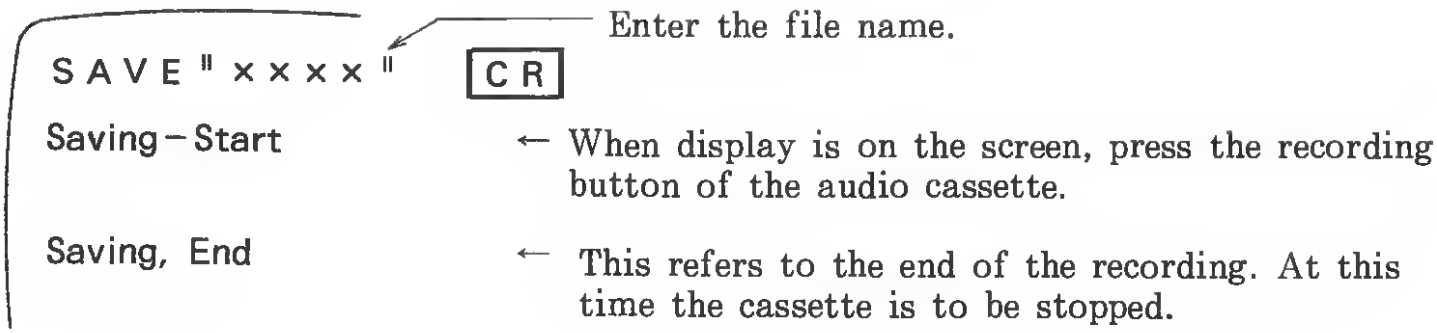
```
RENUM  300 , 200 , 50
                   |
                   └ Step assignment
```

Renumbering with step 50 starts from line No. 300.

SAVE, LOAD, VERIFY

SAVE

SAVE allows the recording of generated programs, data, etc. in the cassette tape.



Use the file name so that the program content is easily understandable.

For naming, use no more than 16 characters. If you have a cassette deck available, use it. If the cassette deck has a counter provided, write the counter number on the cassette tape label.

VERIFY

Whether or not SAVE was done accurately is checked by the VERIFY statement.

Rewind the tape, input CR and press the play button. If accurately saved, VERIFY OK will be displayed. If VERIFY OK is not displayed, repeat SAVE from the beginning.

LOAD

Shift the program data in the cassette tape to the computer.

```
LOAD " ××× "  [CR]      ← Program name.

Loading Start    ← Press play button.
Found " ××× "
Loading End      ← Stop cassette.
```

When using your cassette deck, writing or reading may be impossible depending on the sound level.
This does not mean computer trouble but sometimes results from cassette deck performance. Try to change the levels of sound volume and quality. If writing is still impossible, use a cassette deck compatible with the computer.

For connection to the cassette tape recorder, use the mini-plug available on the market.

SC − 3000 side				Cassette recorder side
IN	↔	LOAD , VERIFY	↔	Earphone (EAR)
OUT	↔	SAVE	↔	Microphone (MIC)

REM

When generating programs, it is convenient to prepare the remark statement in advance so that details of the program or subroutine can easily be understood by looking at the program list later.

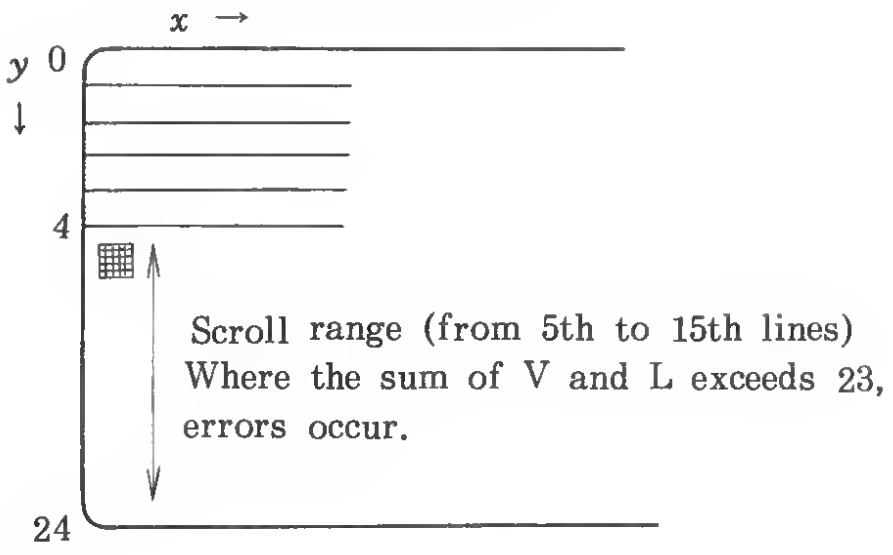
```
10  REM ××× CALCULATION ×××
20  CLS
30  PRINT  2+3
```

The REM statement in the program is ignored and not executed.

CONSOLE

This assigns the scroll range of the text screen, ON/OFF of click sound as well as the shifting from capitals to small letters or vice versa.

Scroll range assignment




```
                V   L    C   S
CONSOLE         5 , 15 , 0 , 1
```

V ; Upper limit of scroll (0～22)
L : Length of scroll (2～24)

C ; Presence of click sound 0 : Not present
1 : Present

S ; Size of Alphanumerics 0 : Capital w/o shift
1 : Small w/o shift

The screen is divided by lines 0～23 in the direction of y.
The numbers after CONSOLE show the starting and ending lines of scroll. If the scroll range is assigned as shown in the above example, the CURSOR moves within the range between the 5th and 15th lines. Set the CURSOR moving range to 0～23. If the highest number of the set range is 24 or more, errors will occur.

The scroll length can be from 2 to 24 (the upper limit set number).

The third numeric of CONSOLE,, 0, refers to whether or not the CURSOR is set so as to produce a click sound at the time the key was pressed.

0 ; No sound
1 ; Makes sound

The 4th numeric of CONSOLE ,,, 1 defines the capital or small alphabetical characters.

0 ; Capital letters w/o shift
1 ; Small letters w/o shift

For omitting the 4 numerics, input only (,). The CONSOLE statement can be cleared by the RESET key.

Chapter 4. Functions

In the function group, there are mathematical functions and string functions. Be sure to remember functions which are frequently used.

RND (Random number)

This function is used to have the values of variables generated at random. This is frequently used, being useful in simulating dice rolls, and irregularly moving games and targets.

RND (1) Random numbers between zero and one.

RND (0) The previous random number will be given.

RND (−1) Random number generation pattern is reset.

Let's generate random numbers.

```
10 FOR N=0 TO 10
20 R=RND(1)
30 PRINT R
```

```
40  NEXT  N
RUN

 .2380546294
 .7041382496
 .4925371138
Ready
```

Numbers from 0 to 1 are generated at random. These are random numbers. If numbers which appear next are predictable, these cannot be called random numbers. What is interesting is that until you cast dice you can't tell what the result will be. Decimal fractions are not practical.
The random numbers are arranged as follows to allow you to use them more easily.

INT (n) (integer)

INT functions convert real decimal numbers into integers.

```
? INT ( 3.14 )  CR

3   ← Decimal numbers are erased.
Ready
```

DICE

```
10  FOR  N=0  TO  20
20  S=INT(RND(1)*6)
30  PRINT  S;
40  NEXT  N
RUN
3  5  0  4  1  5  0  .....
```

Decimal points have disappeared. While 0 exists, 6 doesn't.
This is not applicable for dice. Let's correct it.

```
20  S=INT(RND(1)*6)+1
```

To eliminate 0 and insure necessary numbers.

The number wanted.

Now, this time, numbers from 1 to 6 appear. Try in different ways by changing numerals.

Program for rounding to the nearest whole number.

```
10  INPUT  A   ← Value with decimal fractions
20  PRINT  INT(A+0.5)
30  GOTO  10
```

This is the program in which decimal fractions are rounded to the nearest whole number. The program is applicable to various calculations. Be sure to keep this in mind.

CHARACTER STRING FUNCTION

More explanation about string variables is given here.

String variables are also called character string variables.
In other text books, the term of string variables is used.

Let's see how characters are handled in computers.

ASC ("N") (ASCII Functions)

ASCII refers to American Standard Code Information Interchange which with numbered characters and symbols allows computers to process information with ease.

Now, let's try this one.

```
? A S C ( " A " ) [C R]   (Run by direct mode)
6 5
R e a d y
```

The above number 65 represents A. When using ASC statements, enclose the character in ( ) by " " as in (" A ").
Now try to enter a symbol in ( ), instead of A.

```
? ASC (" ! ")  Now, 33 is displayed.
```

In the inside of the computer, characters and symbols on the KEYBOARD are entered, corresponding to numbers from No. 32 to No. 255. Unlike humans, computers are unable to understand characters as they are. In computers, all characters are handled as numerics. Thus, characters are classifiable on an understanding that A (65) precedes B (66). Even if more than two characters enclosed in ( ) are entered as in ? ASC (" BA ") the computer checks only the first character and displays the numeric.

Let's try another one.

```
10  INPUT  A$
20  Q=ASC(A$)
30  PRINT  Q
40  GOTO  10
```

After RUN, when characters and symbols are entered, corresponding code numbers are displayed.

CHR $

This is the opposite of ASC statement and gives control functions for variables and characters.

```
? C H R $ ( 6 5 ) [CR]
A
```

In the ASCII code, A was represented by 65.

Lets's look at the characters entered in the computer.

```
1 0  F O R  M = 3 2  T O  2 5 5
2 0  P R I N T  C H R $ ( M ) ;
3 0  N E X T  M
R U N
```

Characters and symbols printed on KEYBOARD are displayed in rows. These are characters and symbols contained in the computer.
Look at the character set. Notice that codes displayed on the code table and the screen are the same.

LEFT$, RIGHT$, MID$

These are functions which take out part of the characters from the lengthy character strings.

```
10  A$ = "COFFEE COCOA MILK"
20  M$ = LEFT$(A$, 6)
30  PRINT M$          ←—(String up to the 6th character from the left)
RUN
COFFEE
```

Take out the character string up to the 6th character in A$ (space is also counted) and substitute it in M$ so that these are displayed on the screen.

```
10  A$ = "COFFEE COCOA MILK"
20  M$ = RIGHT$ (A$, 4)
30  PRINT M$
RUN
MILK
```

String up to the 4th character from the right.

Now, take out characters from the 4th character from the right up to the end.

```
10  A$ = "COFFEE COCOA MILK"
20  M$ = MID$ (A$, 8, 5)
30  PRINT M$
RUN
COCOA
```

No. of characters to be taken out.

Starting point

Take out 5 characters starting from the 8th character from the left of the character string.

LEN (length)

LEN (A$) will give you the counted character numbers of A$. Also in this case, characters include all, even the space enclosed by " ". In " ", even if characters consist of space only, these spaces are treated as characters.

```
10  A$ = "SEGA  PERSONAL  COMPUTER"
20  PRINT  LEN(A$)
RUN
22
```

LEN gives you the character numbers including spaces.

The following way of use is also possible.

```
10  A$="**********"
20  FOR  I=1  TO  LEN(A$)
30  PRINT  LEFT$(A$,I)
40  NEXT  I
RUN
*
**
***
****
⌇
**********
```

STR$, VAL

These convert values into string variables or convert numeric string used as string variables into values.

STR $

```
10  A=1 : B=3
20  D$=STR$(A)+STR$(B)
30  D=A+B
40  PRINT  D$, D
RUN
1  3              4  ← Result of line No. 30
  └─ Result of line No. 20
```

When STR $ (A) is entered, numerics convert into characters.
In the addition of characters, characters are in a row but no calculation answer is displayed.

VAL

VAL functions have features quite opposite to STR $ and convert character string numerics into values.

```
10  A$="12345"
20  B$="11111"
30  C$=A$+B$        ← Addition of character string
40  C=VAL(A$)+VAL(B$)    ← Addition of values
50  PRINT  C$
60  PRINT  C
RUN
1234511111   ← Character string
  23456      ← Numeral value
```

TIME$

The computer has built−in clock functions provided inside.
The clock is an accurate digital quartz clock with a quartz oscillation mechanism.

When the computer power switch is turned on, the clock starts to work in increments of 1 second from that moment on.

When the power is turned on, the display is as follows :

```
00:00:00
```

After a specific time elapse, the time elapsed is displayed.

```
PRINT TIME$ [CR]
 00:12:32  ← Time after power switch was turned on.
```

When it is used as a clock, the following applies.

```
10 TIME$ ="08:15:00"  Current time ("hour : minute :
20 CURSOR 15,15:PRINT TIME$          second")
30 GOTO 20
```

Once the time is entered, it will remain set until the [RESET] key is pressed or the power is turned off.

SPC (space), TAB (tabulation)

These are used in PRINT statement.

SPC functions assign spaces between characters.

```
10 PRINT"ABC";SPC(10);"XYZ"
RUN
ABC ␣␣␣␣␣␣␣␣␣␣ XYZ
```

10 spaces

If characters are within the range assigned by SPC, these will be deleted.

TAB FN (function) assignment defines that at which character No., counting from the screen end, the tabulation position is to be displayed.

```
10 PRINT TAB(5);"ABC"
RUN

␣␣␣␣␣ ABC
```

Spaces corresponding to 5 characters.

In the case of TAB FN, even if characters exist between the assigned tabulation positions, these characters are not deleted. This function is used in PRINT statements. So be sure to keep this in mind.

INKEY$

This statement is to check which one of the keys for characters or numerics was pressed. Like in games, it is useful to move some kind of patterns by the KEYBOARD.

```
10  X$=INKEY$
20  IF  X$=""  THEN  10
30  PRINT  X$;
40  GOTO  10
```

Line No. 20 checks to see whether a key has been pressed.
Where nothing is entered, X$ value is referred to as null string. In this case, nothing is displayed and execution is endlessly repeated between line No. 10 and 20 (referred to as infinite loop). When any key is pressed, the key value is substituted in X$ and displayed by line No. 30. In order to get out of this infinite loop, add the following.

```
25  IF  X$="Z"  THEN  100
100  PRINT"END":END
```

Then, when the Z key is pressed, program ends.
Example Operation can be started by [←], [→].

```
10 DIM  D (29)
20 CLS
30 X=18: Y=20
40 D (29) = −1 : D (28) = 1 : D (0) = 0
50 K$ = INKEY$
60 IF  K$ = " " THEN  K = 0 : GOTO  90
70 K = ASC (K$ )
80 IF  K >29  THEN  K = 0
90 X = X + D (K)
100 IF  X<0  THEN  X = 0
110 IF  X >33  THEN  X = 33
120 CURSOR  X , Y : PRINT "   I + I   "
130 GOTO  50
```

FRE

As programs are entered, the remaining memory decreases. FRE FN (function) is used to determine how much space is left in the computer's memory.

Example

```
PRINT FRE
 8300
```

This means that additional programs of up to 8300 BYTES can still be entered.

PRINTER CONTROL COMMAND

LLIST

Print the program list on PRINTER.

The command statement is uscd in the same manner as in the LIST.

```
LLIST                          ← Prints the whole program.
LLIST   Line No.               ← Prints assigned line numbers.
LLIST   Line No. — Line No.
LLIST — Line No.
LLIST   Line No. —
```

LPRINT

This causes the computer to print the content of the PRINT statement on the PRINTER.

- How to apply the command statement is the same as in the PRINT statement.

  LPRINT A prints the content of A on the PRINTER.

```
10  INPUT  A , B
20  C=A+B
30  LPRINT  C
40  GOTO  10
RUN
```

  C value is printed on the PRINTER.

HCOPY

Characters and symbols displayed on the TV screen are printed on the PRINTER by this command. The PRINTER can print numerics, capital and small letters, and ASCII Code symbols. Graphic mode symbols and Dieresis characters cannot be printed.

Chapter 5. Graphics

Let's use graphics

The SC-3000 has two screens available to the user, the text screen and the graphics screen.

The text screen cannot display graphics other than graphic characters, and cannot show more than two colors at a time.

The graphic screen can have all fifteen colors displayed at the same time, and can use commands such as LINE, CIRCLE and PAINT to draw shapes on the screen.

The graphic screen is made up of a grid of dots. We have to be able to describe the position on the screen that we wish to plot. We do this by first giving a number between 0-255, this is how far across the screen. We then give a number between 0-191, which is how far down the screen.

eg. PSET will set a single dot in the color that you choose.
To set a dot in the middle of the screen we go half way across (127 is halfway across the screen) and halfway down (95 is halfway down the screen).

We would have to put:

```
PSET(127,95),8
              ↑
              └── This is the color red.
```

To make the computer do this we will enter a short program.

This selects the graphic screen.	10 SCREEN 2,2:CLS
This sets the dot.	20 PSET(127,95),8
This is to stop the program from finishing and returning to the text screen.	30 GOTO 30

SCREEN

SCREEN statement selects the writing screen and display screen.

If the screen is used only for the text, SCREEN assignment is unnecessary.

The SCREEN statement is used when displaying characters and graphics on the screen.

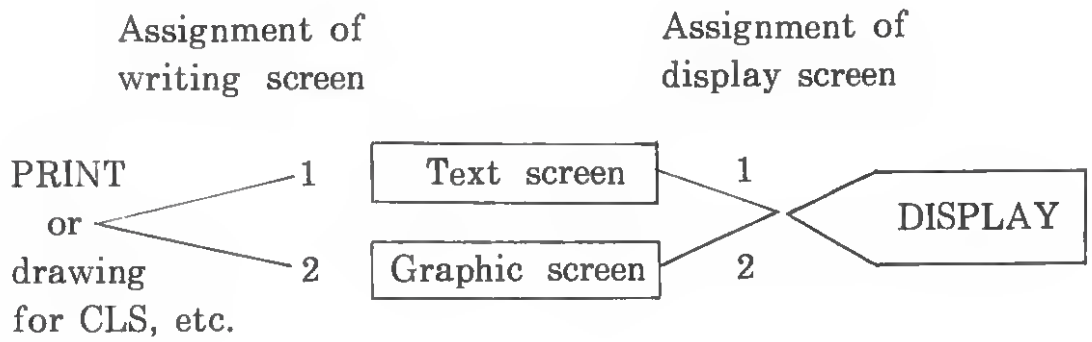
Selects the graphic screen.
Stops the program from finishing and going back to the text screen.

```
10  SCREEN  2,2:CLS
20  GOTO  20
```

Push the [BREAK] key to stop the program.

SCREEN Writing screen , Display screen
*

* { 1 : Text screen
  { 2 : Graphic screen



COLOR

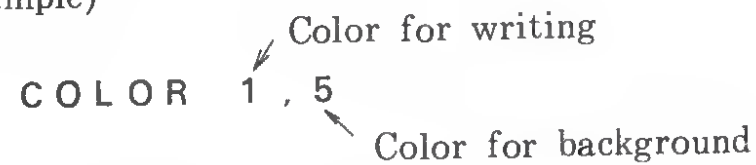
The color command allows you to select colors for the different parts of the display.

Text screen (screen on which program is entered)

(Screen 1)

The writing color assigns colors for characters, etc.
In the example, characters are in black and the background is in blue.

(Example)



Graphic screen (displayed by graphic statement)

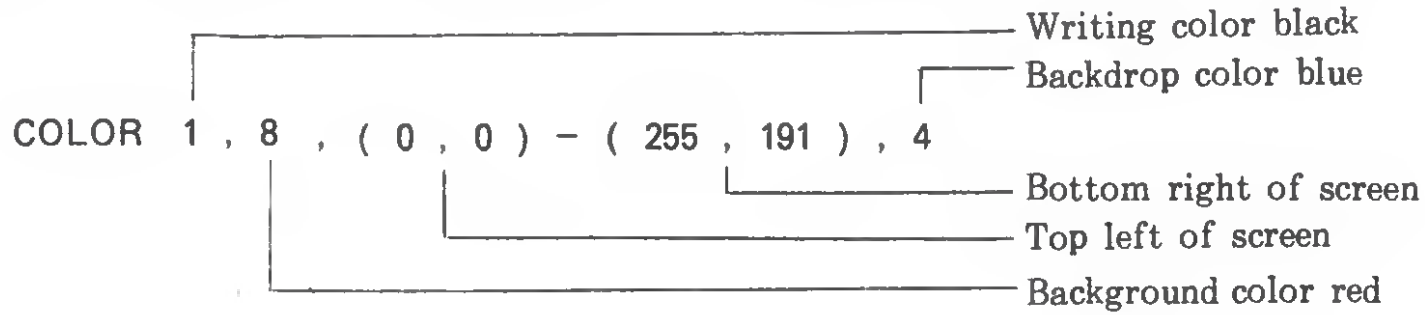
(Screen 2)

The writing color refers to the following:
Character color by PRINT statement.
Color for lines or dots by LINE and PSET statements.
LINE,
Painting color by PAINT statement.

THE BACKGROUND COLOR

A range must be given for the background color. The corners of a box containing the area to have the background color are given.

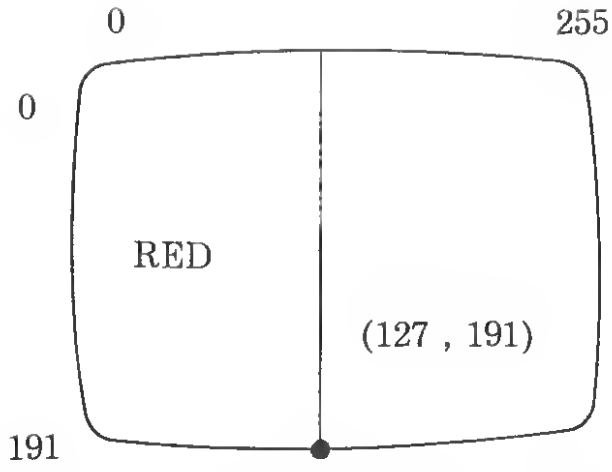
eg.



This will put the color red on the whole screen, with blue as the backdrop (the very top and bottom of the screen) color.

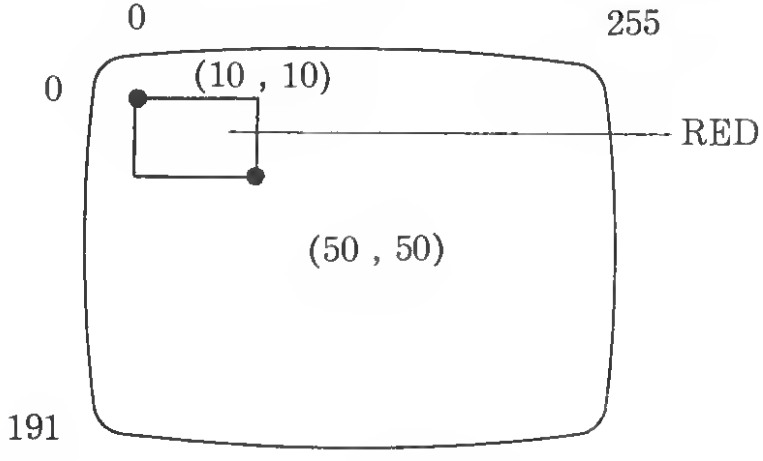
```
COLOR 1 , 8 , ( 0 , 0 ) – ( 127 , 191 ) , 4
```

This will put the color red on the left half of the screen, leaving the right half unchanged.




```
COLOR  1 , 8 , ( 10 , 10 ) - ( 50 , 50 ) , 4
```

This will put the color red in the box with the corners (10 , 10) and (50 , 50).



Example

```
10 CLS
20 FOR C=0 TO 15
30 COLOR 1,C
40 FOR I=0 TO 300
50 NEXT I,C
```

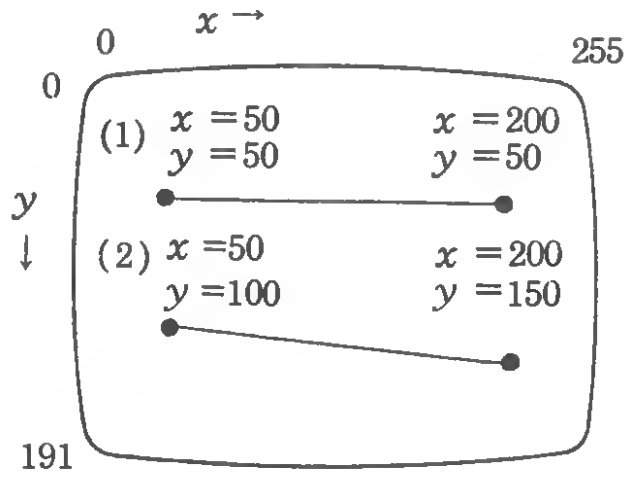
Colors on the screen consecutively change.

Color No.	Color
0	Transparent
1	Black
2	Green
3	Light green
4	Dark blue
5	Light blue
6	Dark red
7	Cyan
8	Red

Color No.	Color
9	Light red
10	Deep yellow
11	Light yellow
12	Dark green
13	Magenta
14	Gray
15	White

LINE

Line statement causes the computer to draw lines after SCREEN 2,2 are entered.



The graphic screen has a coordinate with 0 - 225 (256 dots) in the x direction and one with 0 - 191 (192 dots) in the y direction.

The LINE statement causes the computer to draw lines by assigning coordinates between 2 points.

Screen examples

```
( 1 ) L I N E ( 50 , 50 ) − ( 200 , 50 ) , 1      Black color assignment
( 2 ) L I N E ( 50 , 100 ) − ( 200 , 150 ) , 8    Red color assignment
```

BLINE

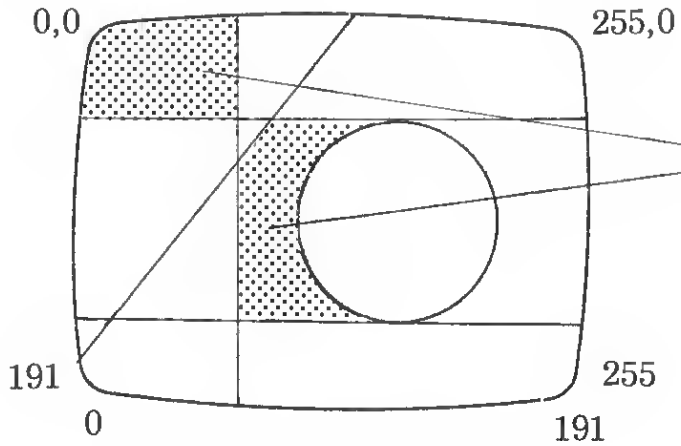
BLINE statement functions erase lines and the box drawn by LINE statements. It is used in the same way as the LINE statement except that color assignment is unnecessary.

```
10  SCREEN2,2:CLS
20  LINE(50,50)-(200,50),1
30  FOR I=0 TO 300:NEXT I   ← Takes time
40  BLINE(50,50)-(200,50)
50  GOTO 50
```

For erasing the box the same applies. However, when drawing a box smaller than the drawn box by BLINE, the color of the particular portion disappears. The BLINE statement is used to erase all the graphics previously drawn or part thereof.

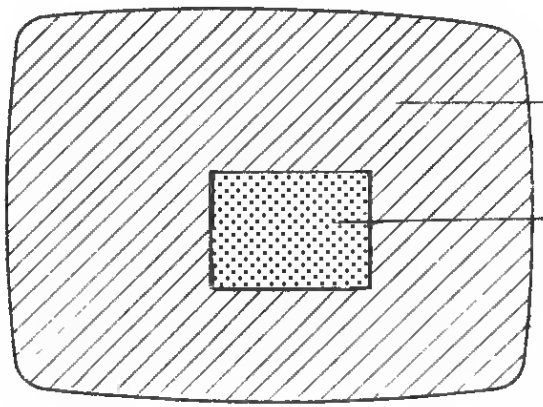
PAINT

Paint screen portions separated by LINE statement or CIRCLE statement.



PAINT (x , y) , color

Paint portions enclosed by coordinate lines where painting starts from.



Paint the entire periphery.

Box drawn by line and BF statements.

```
                    10  SCREEN 2,2:CLS
Draws a blue line   20  LINE(100,10)-(10,180),5
Draws a red line    30  LINE(90,5)-(90,190),8
Draws a green line  40  LINE(5,80)-(150,80),2
PAINT area in middle 50 PAINT(80,50),1
 with color black   60  GOTO 60
```

CIRCLE

Now, let's draw circles. Line drawings or the CIRCLE inside can be painted. Various values enter the CIRCLE statement. So, when entering values, refer to the text until you become familiar with the statement.

```
                (1)  ,   (2)  ,  (3)  ,  (4)  ,    (5)    ,   (6)    ,  (7)
              Coordinate, Radius, Color, Ratio, Start Point, End Point
CIRCLE  (125,95), 50   ,   5   ,   1   ,   0   ,    1     ,    BF
```

Explanation of examples

(1) Coordinate x=125, y=95 (255 (max.) in the x direction.
191 (max.) in the y direction.)

(2) Radius Radius from the center

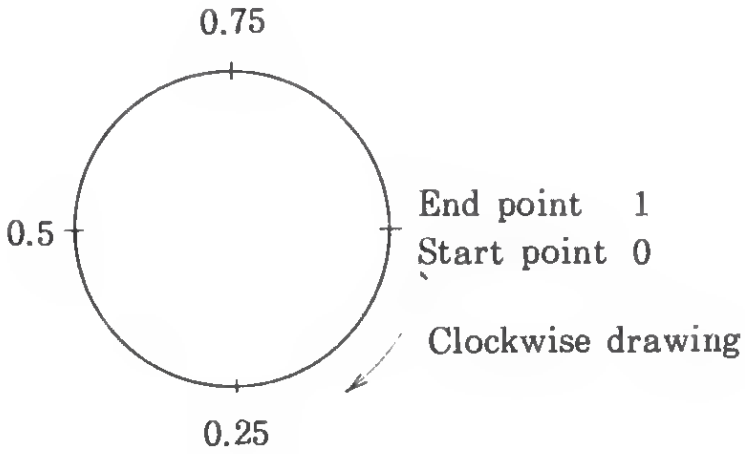
(3) Color 0～15

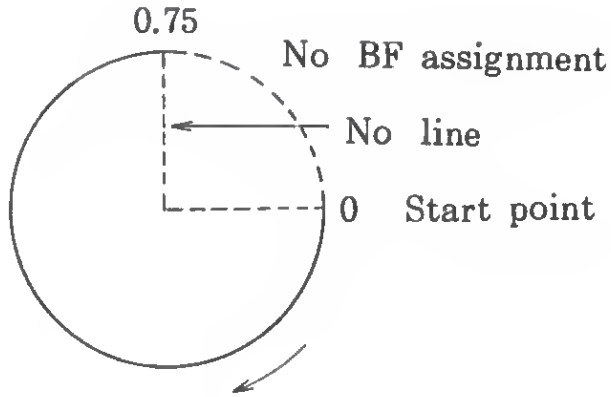
(4) Ratio at 1 True roundness
Less than 1 Ellipse (long sideways)
Greater than 1 Ellipse (longitudinally long)

(5) Start point Position where printing starts from.
(Enter numerics between 0～1 with decimal fractions.)

(6) End point position where printing ends.

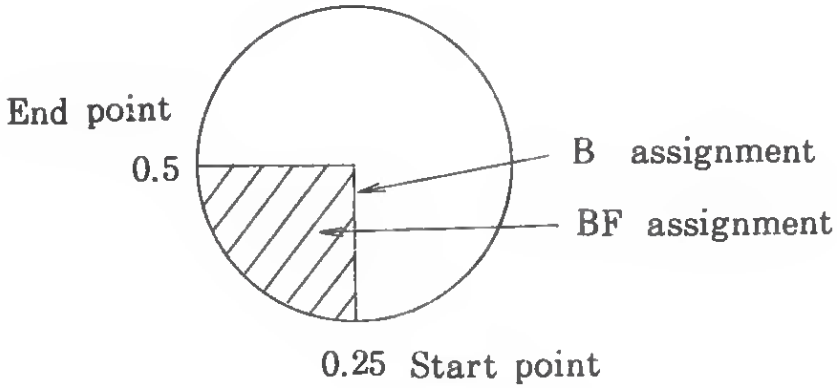
(7) When BF is not assigned, only a circumference is drawn. When B is assigned, lines can be drawn inside also.
When BF is assigned, the assigned color is used for painting part or the whole of the circle.


0.75
0.5
End point 1
Start point 0
Clockwise drawing
0.25


0.75
No BF assignment
No line
0 Start point


Ratio 0.5


Ratio 2


End point
0.5
B assignment
BF assignment
0.25 Start point

Draws a blue circle

```
10  SCREEN 2,2:CLS
20  CIRCLE(130,100),30,4
30  GOTO 30
```

Draws a red eclipse

```
10  SCREEN 2,2:CLS
20  CIRCLE(127,95),30,8,.5
30  GOTO 30
```

Draws an open circle

```
10  SCREEN 2,2:CLS
20  CIRCLE(127,95),30,8,1,0,.75
30  GOTO 30
```

Draws a closed partial circle

```
10  SCREEN 2,2:CLS
20  CIRCLE(127,95),30,8,1,0,.75,B
30  GOTO 30
```

Draws a filled partial circle

```
10  SCREEN 2,2:CLS
20  CIRCLE(127,95),30,8,1,0,.75,BF
30  GOTO 30
```

Draws a filled complete circle

```
10  SCREEN 2,2:CLS
20  CIRCLE(127,95),30,8,,,,BF
30  GOTO 30
```

BCIRCLE

BCIRCLE is used when erasing circles drawn by CIRCLE statements. In this case, color assignment is disregarded and CIRCLE is drawn in the same color as the background, so CIRCLE becomes unnoticeable.

Let's add the following program to the previous program.

Draws & erases circles.

```
10 SCREEN 2,2:CLS
20 FOR R=10 TO 50 STEP 10
30 CIRCLE(125,95),R,8
40 NEXT R
50 FOR R=10 TO 50 STEP 10
60 BCIRCLE(125,95),R,1,
70 NEXT R
80 GOTO 10
```

Draws a circle inside a box

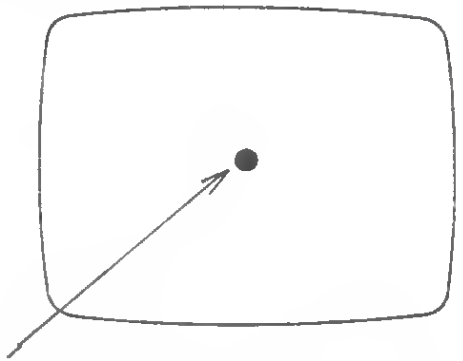
```
10 SCREEN 2,2:CLS
20 LINE(20,20)-(240,170),6,BF
30 BCIRCLE(128,96),30,,,,,BF
40 GOTO 40
```

PSET

This statement allows setting of dots in the specified position on the screen.

```
PSET(x ,y ) , 1        Coordinates color
```

By consecutively varying coordinates, straight lines and curves can be drawn.



A dot can be generated.

```
10  SCREEN  2 , 2 : CLS
20  X = 0 : Y = 95 : E = 1
30  PSET(X , Y ) , 8
40  X = X + 1 : Y = Y + E
50  IF  Y = 120  THEN  E = -1
55  IF  Y = 85  THEN  E = 1
60  IF  X = 250  THEN  END
70  GOTO  30
```

PSET allows setting of dots and the generation of graphs by using mathematical functions.

PRESET

PRESET erases dots, counterworking to PSET. Application is the same as in PSET except that PRESET plays the role to erase dots instead of generating them.

Coordinates

PRESET (x , y)

POSITION

The upper left position of the coordinates is 0. When coordinates are assigned by POSITION statements, the assigned position becomes the center which is x=0, y=0.

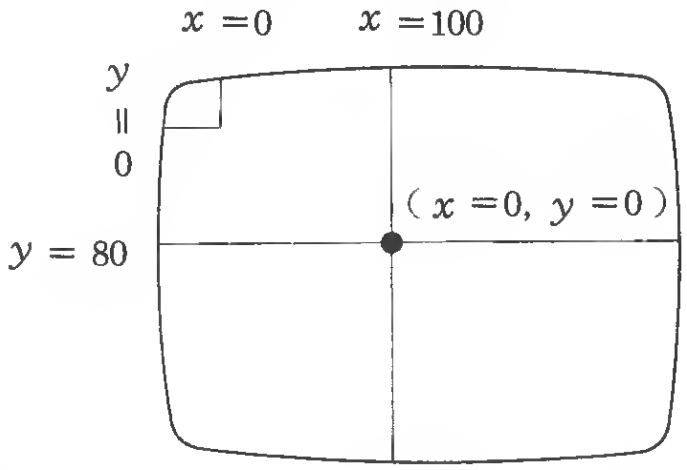
x axis y axis

POSITION (100 , 80) 0 , 0

1 1

Axial direction of x

Axial direction of y

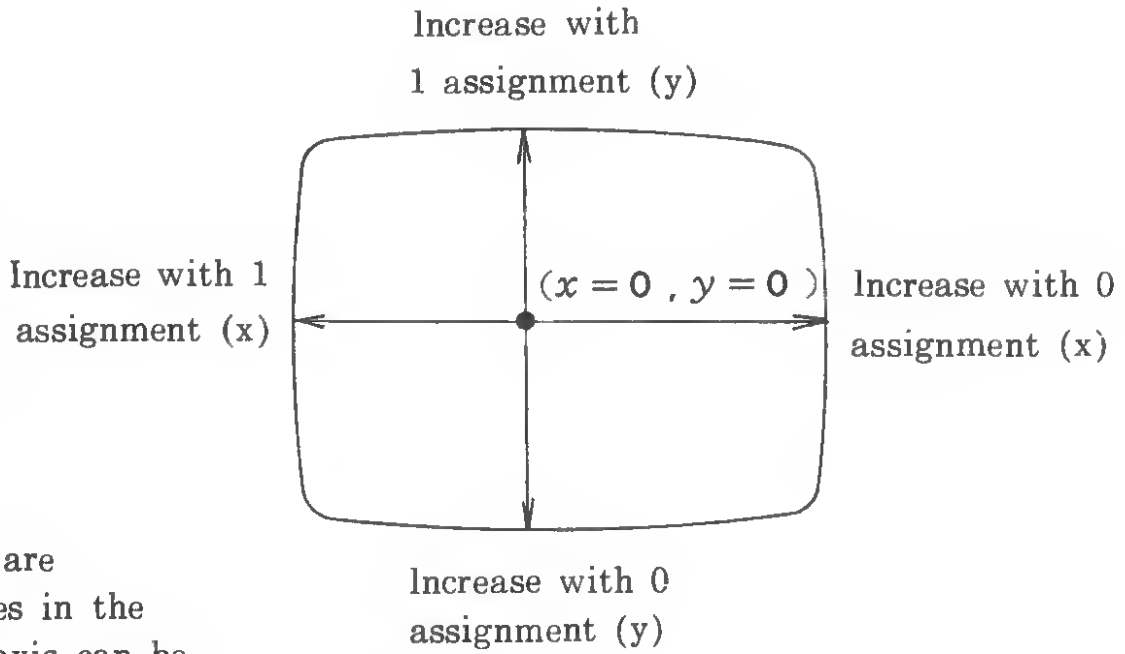


Numeric values following coordinate symbols define the increase directions of x axis and y axis.

With 0 Assignment, x values increase to the right and y values increase downward.

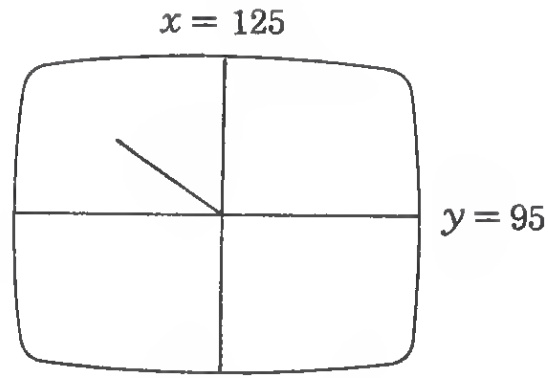
With 1 assignment, x values increase to the left, and y values increase upward.

Where x and y assignments are combined, the value increases in the directions of x axis and y axis can be varied.



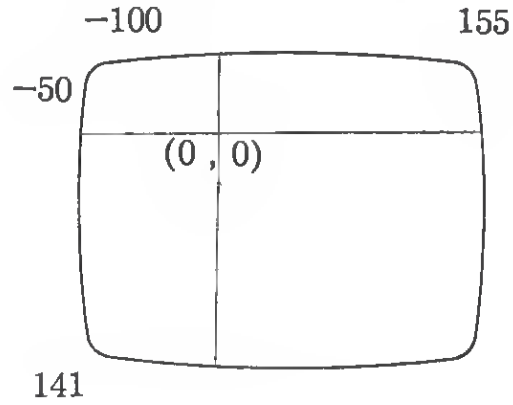
Normally the screen is set at POSITION (0,0),0,0.
When the reset button is pressed it will return to this.

```
10  SCREEN  2,2:CLS
20  POSITION(125,95),1,1
30  FOR  N=0  TO  50
40  PSET(X,Y),1
50  X=X+1:Y=Y+1
60  NEXT  N
```



The combined use of POSITION statements and PSET permits Fn (function) graph drawing.

```
10  SCREEN  2,2:CLS
20  POSITION(100,50),0,0
30  FOR  N=-10  TO  1  STEP.1
40  X=N*20+120:Y=SIN(N)*50+45
50  PSET  (X,Y),1
60  NEXT  N
```

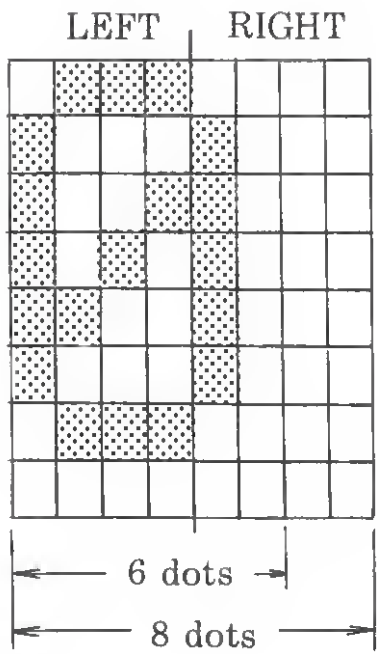


PATTERN

Using PATTERN statements, characters and graphic characters can be generated. Let's rewrite the text mode characters.

PATTERN C# Character code (32–255 or &H20–&HFF),
"Character–string expression"

"Character–string expression"
(hexadecimal)



LEFT	RIGHT
0 1 1 1	0 0 0 0
1 0 0 0	1 0 0 0
1 0 0 1	1 0 0 0
1 0 1 0	1 0 0 0
1 1 0 0	1 0 0 0
1 0 0 0	1 0 0 0
0 1 1 1	0 0 0 0
0 0 0 0	0 0 0 0

LEFT	RIGHT
7	0
8	8
9	8
A	8
C	8
8	8
7	0
0	0

Black location = 1 , White location = 0

Let's assign the above graphic characters to the space key.

```
20 PATTERNC #&H20 , "7088898A8C8887000"
RUN
```

Now, press the space key. You notice that "0" is displayed. This is because "0" was printed in the space. Push the [RESET] button to return the character to its normal pattern. As above, characters can be easily generated. Pattern characters to be displayed on the graphic screen are also printed in the same manner.

Make sure that application procedures are properly understood.

C# ····· Text mode assignment

Character code: In the case of hexadecimal numerals, input numerics from &H 20 to &HFF. In the case of decimal numerals, input numerics from 32 to 255.

Character string expression:

Characters and patterns can be drawn by painting 8 × 8 dots in black.

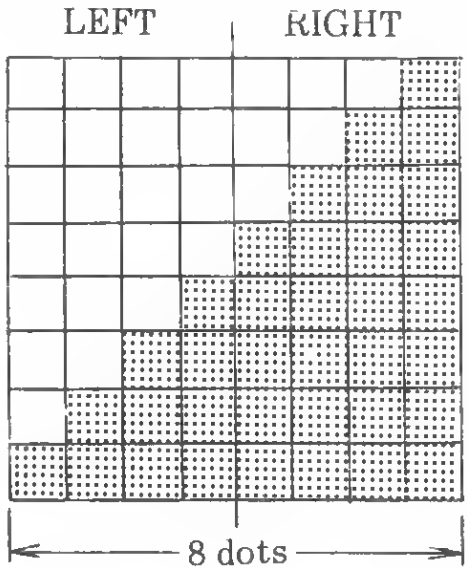
Assign 1 and 0 for respective columns, with black dots assigned as 1 and white dots as 0.

 becomes 01110000. This is separted from the center into two portions and then converted into hexadecimal numerals.

S# : Assignment of graphic screen

Sprite name: Numbers from 0 to 255 are assigned.

Character string: Entering to be same as the text mode.



0 0 0 0	0 0 0 1	0	1
0 0 0 0	0 0 1 1	0	3
0 0 0 0	0 1 1 1	0	7
0 0 0 0	1 1 1 1	0	F
0 0 0 1	1 1 1 1	1	F
0 0 1 1	1 1 1 1	3	F
0 1 1 1	1 1 1 1	7	F
1 1 1 1	1 1 1 1	F	F

```
10 SCREEN 2,2:CLS
20 PATTERNS#0,"0103070F1F3F7FFF"
30 SPRITE 0,(10,0),0,1
RUN
◢ ← This mark is displayed or SCREEN 2.
```

0111 is 7 and 0000 is 0 and this results in "70".

Try to generate characters as per the above.

HOW TO DRAW PATTERNS

To begin with, divide graph sheets into 8x8 locations and paint a dot (location) to generate a pattern.

The painted location is assigned as I and the blank one, as 0. Arrange 0 and 1 numerals beside locations. Divide the eight numerals in a row into two equal parts from the center, each part being four digits.

The four digit numerals represent binary numerals.

These are converted into hexadecimal numerals of two digits.

Refer to the comparison table of decimal binary and hexadecimal numerals.

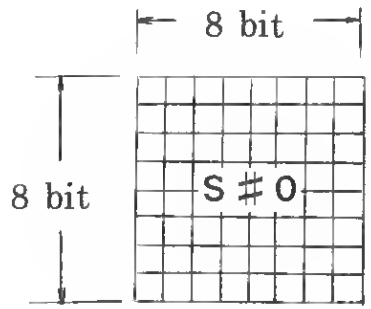
The eight sets of numerics converted into hexadecimal numerals are substituted in " " as character string variables. In this way characters are expressed by 8 × 6.

Decimal Numerals	Binary Numerals	Hexadecimal Numerals	Decimal Numerals	Binary Numerals	Hexadecimal Numerals
0	0000	0	9	1001	9
1	0001	1	10 (Carry)	1010	A
2	0010 (Carry)	2	11	1011	B
3	0011	3	12	1100	C
4	0100	4	13	1101	D
5	0101	5	14	1110	E
6	0110	6	15	1111	F
7	0111	7	16	10000	10 (Carry)
8	1000	8			

MAG

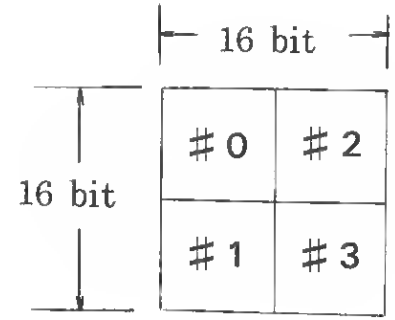
The MAG statement assigns the magnitude of graphic characters to be drawn on sprite planes by PATTERN statement.

MAG 0



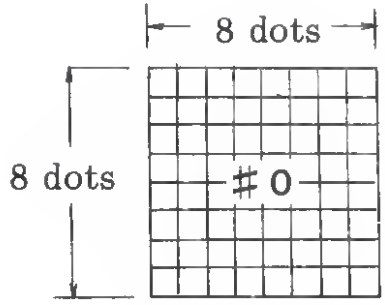
1 bit = 1 dot

MAG 1



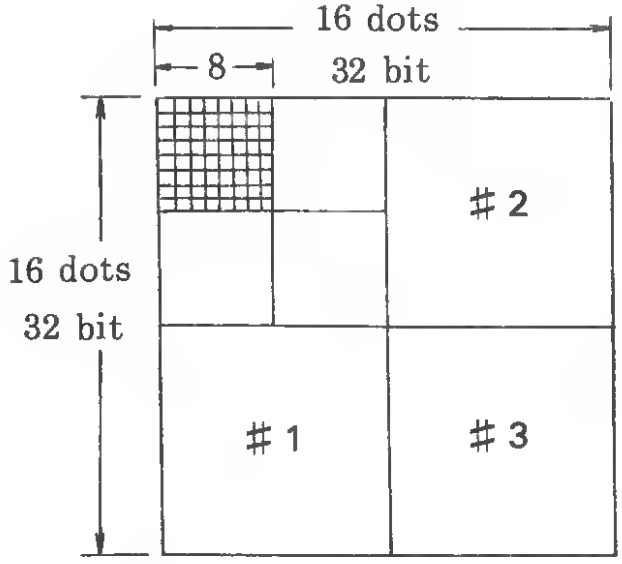
Four of MAG 0 are combined to draw the pattern.

MAG 2



2 bit × 2 bit is deemed as 1 dot.

MAG 3



Patterns are drawn by combining four MAG 2.

In MAG 0, patterns are drawn within 8×8 dot locations with 1 bit as 1 dot.

In MAG 1, patterns can be drawn within 16×16 dot locations by combining four locations, i.e., #0～3, #4～#7, ... , #252～#255.

In MAG 2, patterns are drawn within 8×8 dot locations with 2 bit × 2 bit as one dot. The bit number is 16 bit × 16 bit.

In MAG 3, patterns can be drawn by combining four locations as assigned in MAG 2. In this case, the bit number will be 32 bit × 32 bit.

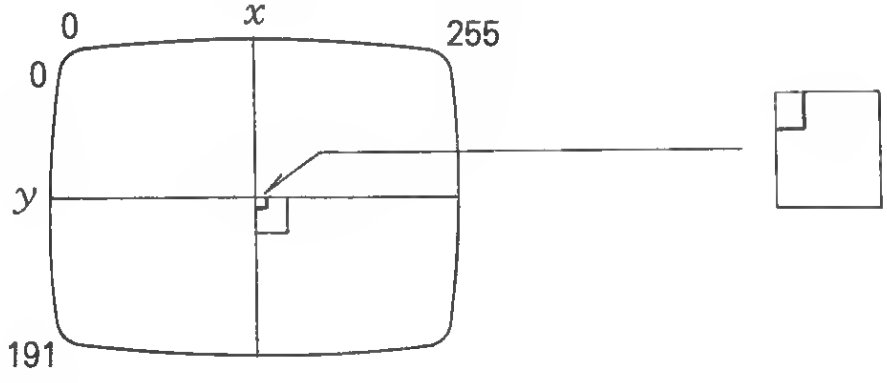
SPRITE

MAG statement, PATTERN statement and SPRITE statement are absolutely necessary when using sprite functions.

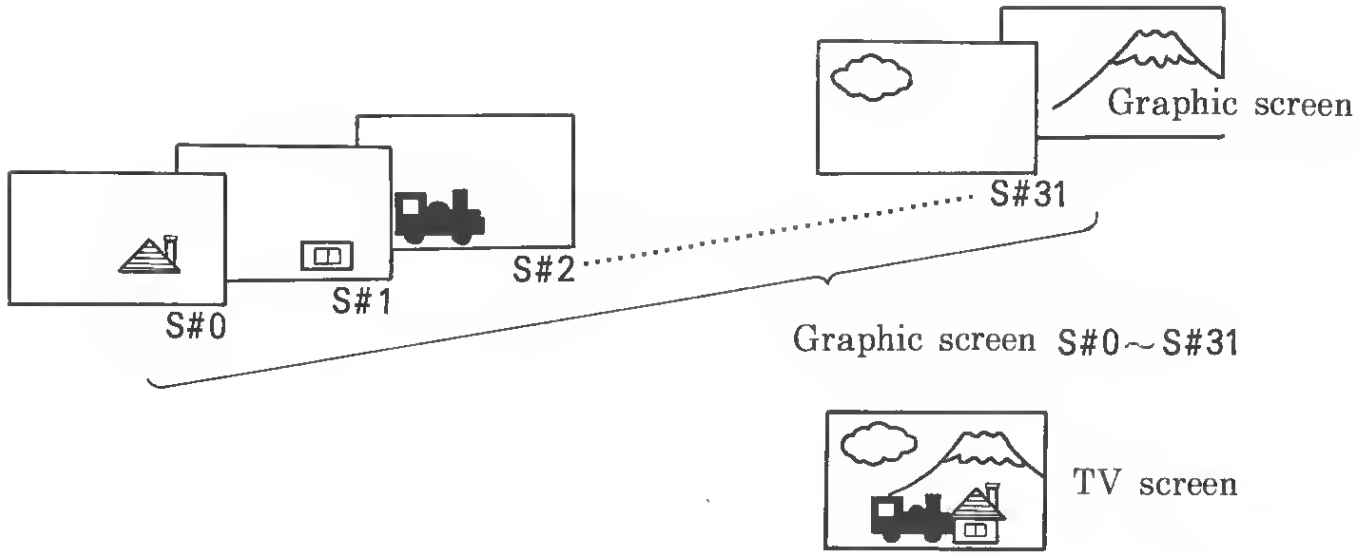
0～31 Coordinate
SPRITE Graphic screen No., (x,y), Sprite name, Color

There are 32 graphic screens (0～31) and the SPRITE statement assigns the number of the graphic screen on which the sprite is to be drawn.

Graphic screen No. 0 takes the foremost position and as the number increases, the graphic screen's position becomes progressively more and more in the background. When graphic screens are intersected, the one with a smaller number takes precedence. Coordinates used are 0～255 (x) and 0～191 (y). The upper left coordinates define the position assigned by the PATTERN statement.



SPRITE name refers to the S# name defined by the PATTERN statement. If there are clearances in a pattern drawn by the PATTERN statement, the screen behind the preceding one will be seen through the clearance. Taking advantage of this situation, deep and solid patterns can be generated.



Note: At the maximum, 4 patterns of the graphic screen can be displayed on the horizontal line. Where more than 4 patterns are in a row horizontally, the 4 patterns which have the highest priority will be displayed.

Chapter 6. Mathematical Function – 2

The computer excels in calculations. Various functions including trigonometric function are built into the computer in order to increase the calculation function.

ABS (X)

Function: Gives absolute value of expression X
Form: ABS(X)
How to use:

```
PRINT ABS(-5) [CR]  5
PRINT ABS(3*(-6)) [CR]  18
```

RAD

Function: Angular degrees are converted into radians.
Form: RAD (X)
How to use: 0°, 15°, 30°, 45° and 60° are converted into radians.

```
10 FOR I=0 TO 60 STEP 15
20 X=RAD(I)
30 PRINT "RAD(";I;"°)=";X
```

```
40 NEXT I
RUN
RAD(0°)=0
RAD(15°)=.261799387 8
RAD(30°)=.523598775 6
```

DEG

Function: Radians are converted into degrees.
Form: DEG (X) X refers to radian.
How to use: PRINT DEG (0.26) [C R] 14.896902673

PI

Function: The ratio of the circumference of a circle to its diameter is defined.
Form: PRINT PI [C R] (3.1415926536)
How to use:

```
10  INPUT"RADIUS";A
20  S=A^2*PI
30  PRINT"AREA OF CIRCLE";S
RUN
RADIUS  5
AREA OF CIRCLE  78.5398163 33
```

SIN (Sine)

Function: Defines the values of trigonometric function and sine.
Form: SIN (X) Argument (X) refers to radian.
How to use:

```
10  FOR TH=0 TO 90 STEP 30
20  S=SIN(RAD(TH))
30  PRINT TH;TAB(10);S
40  NEXT TH
RUN
0           0
30          .5
60          .866025403 79
90          1
```

COS (Cosine)

Function: Defines the values of trigonometric function and cosine.
Form: COS (X) Argument (X) radian
How to use:

```
10 FOR X=0 TO 90 STEP 30
20 A=COS(RAD(X))
30 PRINT X;TAB(10);A
40 NEXT X
RUN
 0          1
 30         .866025403 79
 60         .500000000001
 90         0
```

TAN (Tangent)

Function: Defines the values of trigonometric function and sine.
Form: TAN (X) Argument (X) refers to radian.
How to use:

```
10  INPUT "DEGREE";A
20  X=TAN(RAD(A))
30  PRINT "TAN(";A;"°)=";X
RUN
DEGREE  30
TAN(30°)=    .577350269 19
```

ASN (Arc Sine)

Function: Obtains the 0 value (degree) of SIN 0
Form: ASN (X) (where X is −1∼1)
How to use:

```
10  X=ASN(.5)
20  Y=DEG(X)
30  PRINT  Y
RUN
 30
```

ACS (Arc cosine)

Function: Obtains the 0 value (degree) of COS 0
Form: ACS (X) (Where X is −1∼1)
How to use:

```
10  X=ACS(-1)
20  Y=DEG(X)
30  PRINT  Y
RUN
  180
```

ATN (Arc Tangent)

Function: Obtains the value of the arc tangent.
Form: ATN (X)
How to use: Values to be obtained range between $-\frac{\pi}{2}$ to $\frac{\pi}{2}$

```
10  X=ATN(1)
20  PRINT  X
RUN
  .785398163 4
```

LTW

Function: Obtains common logarithm with 2 as a base.
Form: LTW (X)
How to use: Same as in LOG.

LGT

Function: Obtains the common logarithm of the value with 10 as a base.
Form: LGT (X)
How to use: Obtains the common logarithm of 10, 100 and 1,000.

```
10  N=1
20  N=N*10
30  X=LGT(N)
40  PRINT"LGT(";N;")=";X
50  IF  N<1000  THEN  20
RUN
LGT(10)=1
LGT(100)=2
LGT(1000)=3
```

EXP

Function: Obtains raising to power of the natural logarithm with e as a base.
Form: EXP(X)
How to use: e^1 .e^2 . and e^3 are obtained respectively.

```
10  FOR  I=1  TO  3
20  X=EXP(I)
30  PRINT"EXP(";I;")";X
40  NEXT  I
RUN
EXP(1)=2.718281828 4
EXP(2)=7.389056098 7
EXP(3)=20.08553692 3
```

SGN (Sign)

Function: SGN Fn assigns value signs.

When x value is	negative	−1
〃	0	0
〃	positive	1

Form: SGN (X)
How to use:

```
10  FOR  I=-2  TO  2
20  N=SGN(I)
30  PRINT"SGN(";I;")=";N
40  NEXT  I
RUN
  SGN(-2)=-1
  SGN(-1)=-1
  SGN(0)=0
  SGN(1)=1
  SGN(2)=1
```

LOG

Function: Obtains the natural logarithm of value with e as a base.
Form: LOG (X)
How to use:

```
10  FOR  J=1  TO  3
20  X=LOG(J)   ← Argument J is a positive value.
30  PRINT"LOG(";J;")=";X
40  NEXT  J
```

```
RUN
LOG(1)=2.67468532E-11
LOG(2)=  .693147180 57
LOG(3)=1.098612288 6
```

SQR

Function: Obtains the square root of the value.
Form: SQR (X)
How to use: $\sqrt{2}$ and $\sqrt{3}$ are obtained as follows.

```
10  INPUT"NUMERAL";A
20  X=SQR(A)
30  PRINT"ROOT";A;"=";X
40  GOTO  10
RUN
NUMERAL  2
ROOT=1.414213562 4
NUMERAL  3
ROOT=1.732050807 6
```

HEX$

Function: Decimal numerals are converted into hexadecimal numerals.
Form: HEX$ (X)
How to use: Values convertible into hexadecimal numerals range from −32768～32767.
Values −10, −5, 0, 5, 10 and 15 are converted into hexadecimal numerals.

```
10 FOR S=-10 TO 15 STEP 5
20 X$=HEX$(S)
30 PRINT S;"=";X$
40 NEXT S
RUN
-10=FFF6
-5=FFFB
0=0
5=5
10=A
15=F
```

INP

Reads out the content of I/O Areas. This function is the opposite of OUT.

Assigns I/O port No. and reads out the data which is on the port.

How to apply:

```
10  A=INP(&HBE)
20  PRINT  A
RUN
  32
```

In line No. 10, the data of the I/O port No. BE (hexadecimal numerals) is read out to variable A. In this case, the results may vary depending on the situation of the computer. The I/O port numbers which were defined in the system in advance are integers 0～255 (&HOO～&HFF). Situations of outside input devices including Joysticks can be recognised by the above command.

DEF FN

This is a function which computer operting personnel define arbitrarily.

```
  5  REM  AREA  OF  CIRCLE
10  DEF  FNS(R)=R*R*3.14159
20  INPUT"RADIUS=";A
30  Z=FNS(A)
40  PRINT
50  PRINT"AREA=";Z
60  END
RUN
RADIUS=10
AREA=314.15
```

The ratio of the circumference of a circle to its diameter.

DEF FNS (R) =R* R*3.14159

For the above formula, the right side expression can be defined as the function of the left side expression FNS (R). When entering the radius, the Fn (function) which was defined by line No. 30 is called for calculation.

(R) is only a dummy argument, the number returned will be a function of whatever number or variable that is in the brackets when the function is called.

Frequency Table

SCALES	f 1	f 2	f 3	f 4	f 5	f 6
C		131	262	523	1047	2094
C♯ , D♭		139	277	554	1109	2218
D		147	294	587	1175	2350
D♯ , E♭		156	311	622	1245	2490
E		165	330	659	1319	2638
F		175	349	698	1397	2794
F♯ , G♭		185	370	740	1480	2960
G		196	392	784	1568	3136
G♯ , A♭		208	415	831	1661	3322
A	110	220	440	880	1760	3520
A♯ , B♭	117	233	466	932	1864	
B	123	247	494	988	1976	

Frequency unit Hz.

BEEP

This is used for producing a short sound in programs.

BEEP	Makes beep sound.
BEEP 0	Stops beep sound.
BEEP 1	Continues beep sound.
BEEP 2	Makes beep beep sound.

Example :

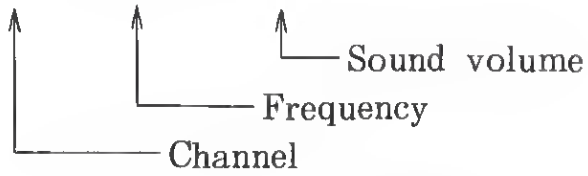
```
10 A$="SEGA PERSONAL COMPUTER"
20 FOR I=1 TO LEN(A$)
30 PRINT MID$(A$,I,1);
40 BEEP
50 FOR J=0 TO 100:NEXT J,I
60 END
```

SOUND

SC−3000 has a synthesizing function.

Example

```
SOUND  1 , 1 0 0 0 , 1 5
```



Sound of 1000 Hz is produced.

(Channel)

Only one sound is produced from one channel. Six channel assignments (0～5) are possible. (Sound up to treble chord can be produced.)

0	:	Silences noise.
		Example SOUND 0
1～3	:	Sound from 110 Hz is produced.
4	:	Selection of white noise.
5	:	Selection of synchronous noise.

(Frequency)
When channels 1～3 are assigned, frequency (Hz.) is entered.
When channel 4 or 5 is assigned :

0～2 : Frequencies of 3 defined steps are assigned.
3 : Frequency is assigned by channel 3.

(Sound Volume)

0 : Silences noise.
1 : Minimum sound volume.
≀
15 : Maximun sound volume.

By the above, effect (sound) for games, etc. can be produced, and melodies can be heard in accordance with the following table.

OUT

Data is output to the output port by this statement.

The output port No. is defined in the system in advance to output data to the outside.

Output port numbers are integers from 0 to 255 (& HOO & HFF).

VDP data register	&H BE
Command register	&H BF
Sound generator	&H 7F

POKE, PEEK, CALL

Programs when entered by BASIC are memorized in the MEMORY in their respective order. In addtion, data and the machine language can be printed in the specific memory.

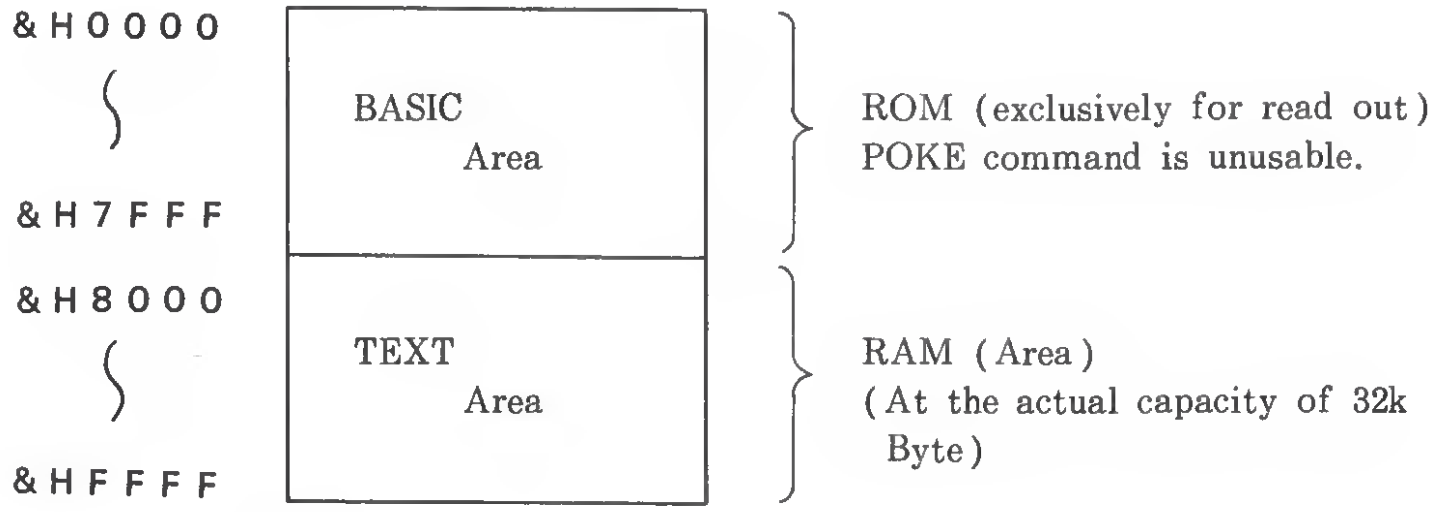
POKE	Address	Data
POKE command	POKE &H 9000,	65

The address covers from &H 8000 (−32768) to &HFFFF.
DATA are integers from 0 to 255.
Note that the address varies depending on the used quantity of MEMORY, BASIC version and types.

PEEK Function

Address
Read out command A = PEEK (&H9000)
Reads out the content of assigned address memory.

Main memory map



DATA conversion program

```
10  REM  ***  DATA CONVERSION
20  INPUT  "DATA=" ; D
```

```
30  IF  D=>256  THEN  GOTO  20
40  POKE&H9000,D
50  A=PEEK(&H9000)
70  B$=CHR$(A)
80  PRINT  A;"=";B$
90  GOTO  20
RUN
DATA=65
  65=A
DATA=
```

In this program, entered values are converted into symbols.
When exceeding 256, the value should be entered again.

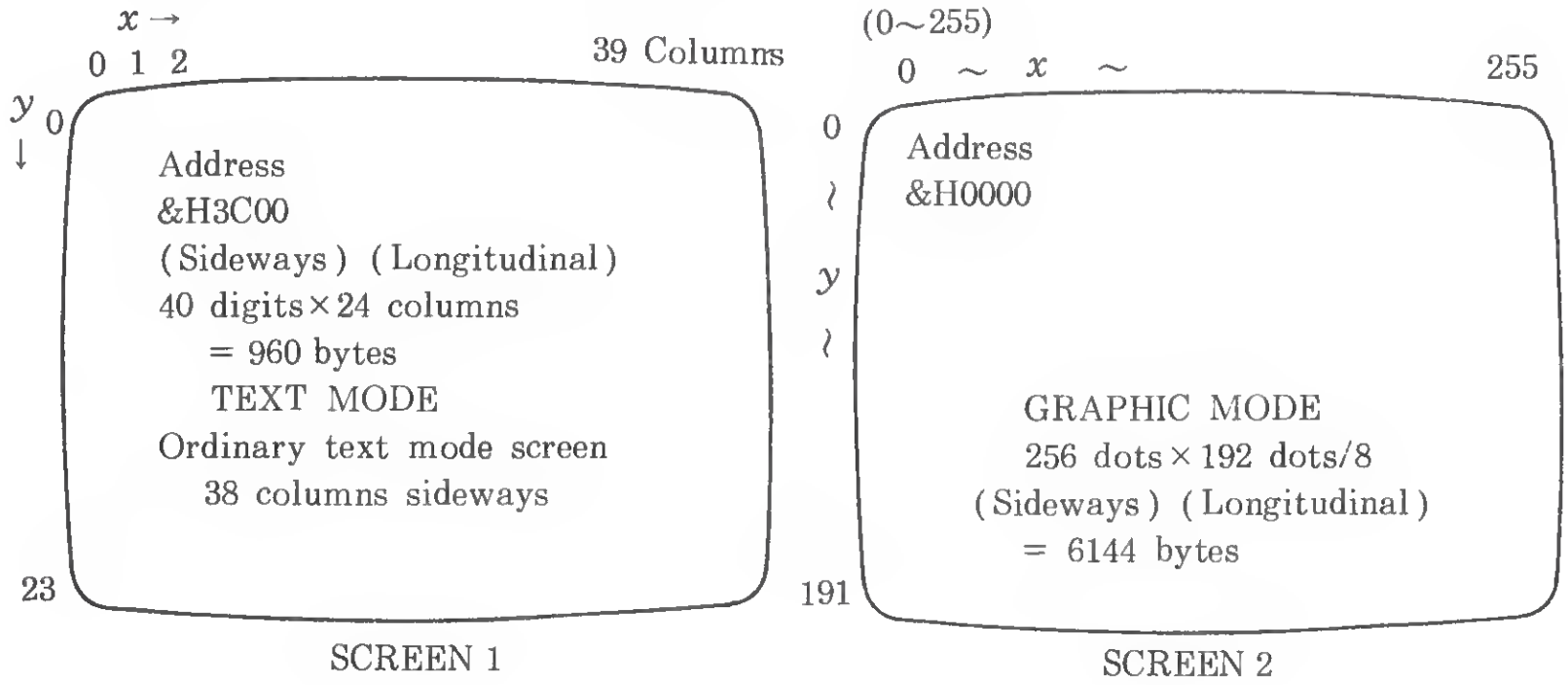
No symbol output means that there is no symbol corresponding to the value available.

CALL

This calls the address printed by the machine language.
Differing from the BASIC language, the machine language should be mastered separately from the former (BASIC).

The incorrect use of the machine language may damage the program. So take care. You could learn the machine language at some other opportune time.

VPOKE ADDRESS ASCII DATA (Text mode)



Part of VRAM MAP

The address calculations on the text screen are carried out as follows.

$$\text{Address (text)} = y * 40 + x + \&H3C00$$
$$\text{where } (x = 0 \sim 39,\ y = 0 \sim 23)$$

For the data to be sent, the ASCII code of the corresponding character is applicable (0~255 in decimal numerals and 0~&HFF in hexadecimal numerals).

Note : As shown in the left figure above, the horizontal axis is deviated by 2 columns as compared to the ordinary text screen. Thus, the display position defined by CURSOR statement deviates from that defined by VPOKE, by about 2 locations in the horizontal direction.
(See page 146.)

VPOKE ADDRESS, DATA (For graphic screen)

Graphic address calculations are carried out as follows.

$$\text{Graphic address} = \text{INT}(y/8) * 256 + \text{INT}(x/8) * 8 + y \text{ MOD } 8$$
$$\text{where } (y \text{ is } 0 \sim 191,\ x \text{ is } 0 \sim 255)$$

The address derived from the above calculations is the beginning address of 8 bits (dots) in the assigned horizontal direction. The assigned address is the x−INT (x/8) bit location counting from the left of the beginning address.

The data to be sent are hexadecimal or decimal numerals displayed by the bit pattern in a horizonal row.

Example

 ⇨ &H93 (147)

Similarly, the color table address for graphic color assignment is derived from the addition of &H 2000 to the above address. The data to be sent are natural numbers (0 - 255) of 1B (1 Byte). The upper 4 bits of these numbers converted into binary data are the assigned color number, and the lower 4 bits, the background color number. (The addresses of the graphic pattern generator table and color table respectively corresponds at 1 : 1).

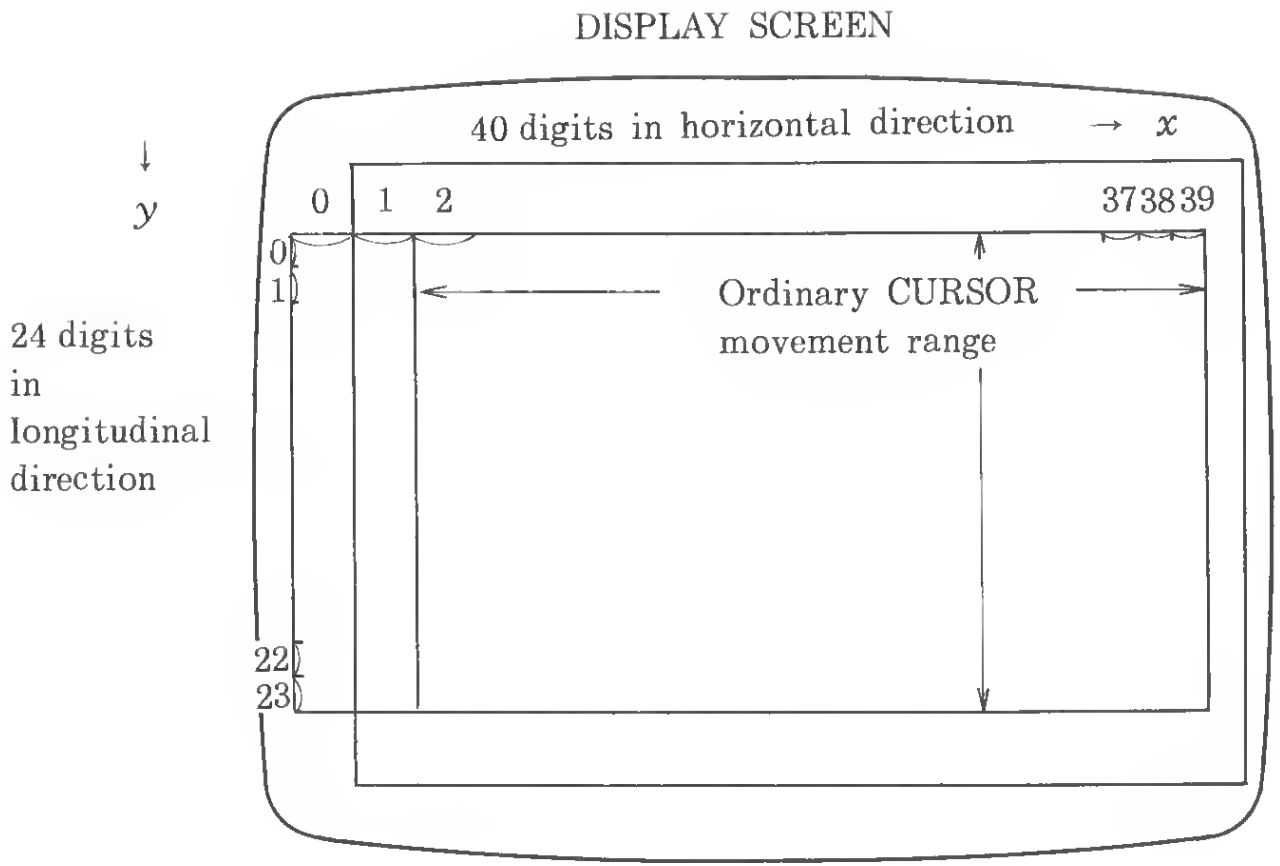
Graphic color table address

$$= INT(y/8) * 256 + INT(x/8) * 8 + y \ MOD \ 8 + \&H2000$$

Where y is $0 \sim 191$
x is $0 \sim 255$

Color data = Assigned color No. ($0 \sim 15$) * 16 + background color No. ($0 \sim 15$)

SCREEN


DISPLAY SCREEN
40 digits in horizontal direction
x
y
0
1
2
37
38
39
0
1
22
23
Ordinary CURSOR
movement range
24 digits
in
longitudinal
direction

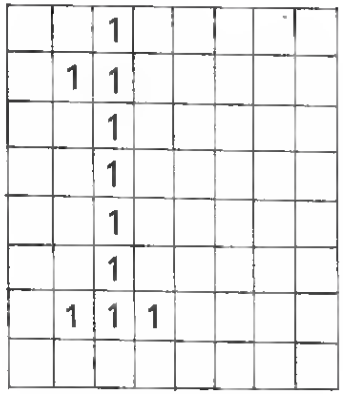
VPEEK

Use VPEEK with reference to VPOKE address. Program to read the content of the pattern generator table in VRAM.

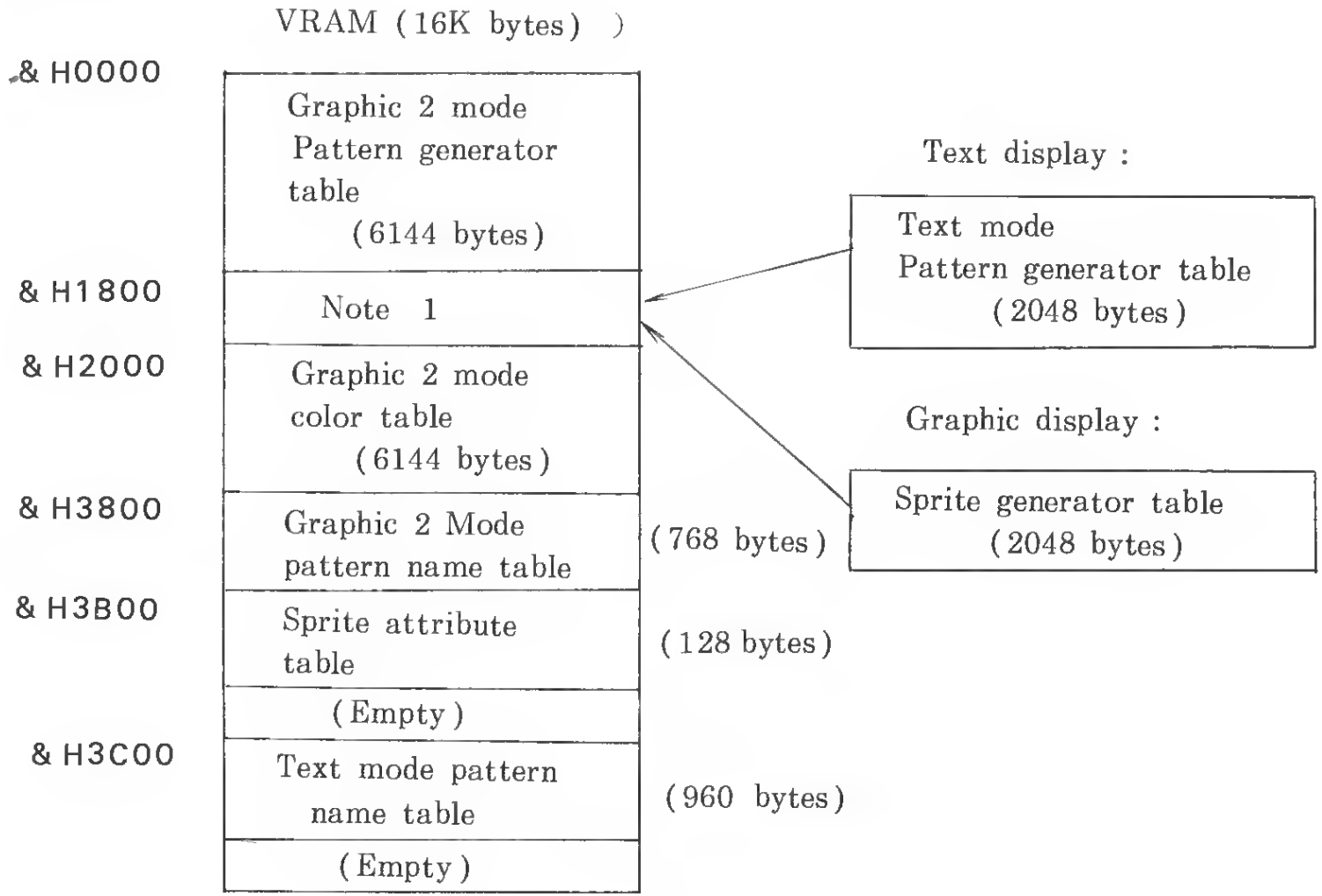
Example

```
10  AD=&H1800+&H31*8     :REM The beginning address
20  FOR A=AD TO AD+7           of REM "1" pattern
30  DA=VPEEK(A)
40  PRINT HEX$(DA)
50  NEXT A
```

```
20
60
20
20
20
20
70
00
```



VRAM MAP


VRAM (16K bytes)
&H0000
Graphic 2 mode Pattern generator table (6144 bytes)
&H1800
Note 1
&H2000
Graphic 2 mode color table (6144 bytes)
&H3800
Graphic 2 Mode pattern name table
(768 bytes)
&H3B00
Sprite attribute table
(128 bytes)
(Empty)
&H3C00
Text mode pattern name table
(960 bytes)
(Empty)
Text display :
Text mode Pattern generator table (2048 bytes)
Graphic display :
Sprite generator table (2048 bytes)

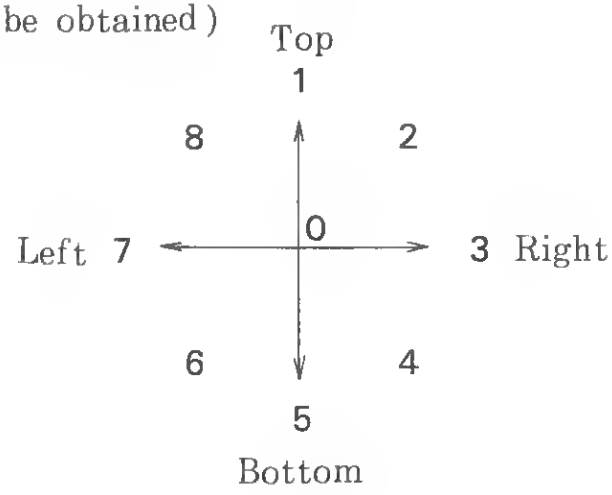
Note : The contents of the table for 2K Bytes of &H1800 &H1FFF varies depending on the display mode (text/graphic).

The table contents of the mode on the unselected side is SAVEed in the MEMORY (RAM).

STICK (n)

(Value to be obtained)

Parameter : 1 = Joystick 1
2 = Joystick 2



STRIG (n)

(Value to be obtained)

Parameter : 1 : Joystick 1
2 : Joystick 2

0 : off
1 : Trigger (left) ON
2 : Trigger (right) ON
3 : Trigger (left, right) ON

STICK, STRIG

Program to find out the situation of connected JOYSTICK.

```
10 REM JOY STICK TEST
20 B$="SHOOT"·CLS
30 P1=STICK(1):P2=STICK(2)
40 S1=STRIG(1):S2=STRIG(2)
50 F1$="":F2$=""
60 IF P1=1 THEN F1$="UP    "
70 IF P1=3 THEN F1$="RIGHT "
80 IF P1=5 THEN F1$="DOWN  "
90 IF P1=7 THEN F1$="LEFT  "
100 IF P2=1 THEN F2$="UP    "
110 IF P2=3 THEN F2$="RIGHT "
120 IF P2=5 THEN F2$="DOWN  "
130 IF P2=7 THEN F2$="LEFT  "
140 IF S1>0 THEN F1$=F1$+B$+STRING$(S1)
150 IF S2>0 THEN F2$=F2$+B$+STRING$(S2)
160 CURSOR 1,10:PRINT"PLAYER 1";F1$
170 CURSOR 1,15:PRINT"PLAYER 2";F2$
180 GOTO 20
```

APPENDIX

Variables and Arrays

Numeric variables	·····	A, B, ····· Z, AA, AB, ····· ZZ A 0, A 1 ····· A 9
Numeric array	·····	(Subscript, ·····) Up to 3 – DIM A (15), B (5, 5), AC (3, 3, 3)
String variables	·····	A $, A B $, A 1 $
String array	·····	(Subscript, ·····) Up to 3 – DIM A $ (15), B $ (5, 5), AC $ (3, 3, 3)

- For variable names, the first character is an alphabetic character and then after, alphabetic characters or numerics. Although any number of characters is acceptable, separation is made by the beginning 2 characters.

- The names of variables and arrays may be the same.

(Range of numeric variables and arrays)

±9.99999999999E−99

±9.99999999999E+99

(Range of string variables and array)

Character length 0 ~ 31

CONSTANT

Numeric constant

Integer form	Example	3, −2, 99926768
Decimal form	Example	0.2, .3, −5.3, 86.0
Exponent form	Example	3, E99, −6E3, 0.3E+5, 4E−82
Hexadecimal form	&H Hexadecimal value ·····	0000 ~ FFFF
	Example &H64 ·····	same as 100
	&HFFFF ·····	same as −1

String constant

Use double quotation to indicate " enclosed by "

Example
" ABC " → Character ABC
" " → → Character NULL
" " " " → Character "
" A3 " " 64 " → Character A3 " 64

Contents	Limitation
Characters taken into the inside from the screen.	256 characters
Character numbers usable for actual text image by reserved words converted from line buffer.	256 characters
Character numbers which can be handled as character string.	255 characters
Level number such as operator priority, etc.	32 levels
Area for string operation	300 characters
FOR ~ NEXT nesting level number	16 levels
GOSUB, RETURN nesting level number	8 levels

CHARACTER CODE

32	SP	48	0	64	ⓐ	80	P	96	`	112	p	128		144	
33	!	49	1	65	A	81	Q	97	a	113	q	129		145	
34	"	50	2	66	B	82	R	98	b	114	r	130		146	
35	#	51	3	67	C	83	S	99	c	115	s	131		147	
36	$	52	4	68	D	84	T	100	d	116	t	132		148	
37	%	53	5	69	E	85	U	101	e	117	u	133		149	
38	&	54	6	70	F	86	V	102	f	118	v	134		150	
39	▼	55	7	71	G	87	W	103	g	119	w	135		151	
40	(	56	8	72	H	88	X	104	h	120	x	136		152	
41	)	57	9	73	I	89	Y	105	i	121	y	137		153	
42	*	58	:	74	J	90	Z	106	j	122	z	138		154	
43	+	59	;	75	K	91	[	107	k	123	{	139		155	
44	,	60	<	76	L	92	¥	108	l	124	:	140		156	
45	−	61	=	77	M	93	]	109	m	125	}	141		157	
46	.	62	>	78	N	94	^	110	n	126	~	142		158	
47	/	63	?	79	O	95	π	111	o	127		143		159	

160	Â	176	Ĭ	192	Ù	208		224		240	
161	Ǎ	177	Ì	193	Ü	209		225		241	
162	Á	178	Í	194	Ū	210		226		242	
163	À	179	Ï	195	α	211		227		243	
164	Ä	180	Î	196	β	212		228		244	
165	Å	181	Ī	197	θ	213		229		245	♠
166	Ã	182	Ô	198	Λ	214		230		246	♥
167	Ā	183	Ǒ	199	μ	215		231		247	♦
168	Ê	184	Ǫ	200	Σ	216		232		248	♣
169	Ě	185	Ó	201	ϕ	217		233		249	☺
170	Ë	186	Ò	202	Ω	218		234		250	
171	Ē	187	Ö	203	Ç	219		235		251	
172	É	188	Õ	204	¿	220		236		252	
173	È	189	Û	205	¡	221		237		253	
174	Ñ	190	Ǔ	206	$\not$	222		238		254	
175	Ň	191	Ú	207	£	223		239		255	

CHARACTER SET

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p			Â	Ǐ	Ù		▌	
1			!	1	A	Q	a	q	—	╳	Ǎ	Ì	Ü		▊	
2			"	2	B	R	b	r	┴	┼	Á	Í	Ū		▊	
3			#	3	C	S	c	s	┬	╱	À	Ï	α		▌	
4			$	4	D	T	d	t	┤	╲	Ä	Î	β		▏	
5			%	5	E	U	e	u	├	◢	Å	Ī	θ		▉	♠
6			&	6	F	V	f	v	┌	◣	Ã	Ô	∧		▏	♥
7			'	7	G	W	g	w	└	◥	Ā	Ǒ	μ		—	♦
8			(	8	H	X	h	x	┐	◤	Ê	Ǫ	Σ		▒	♣
9			)	9	I	Y	i	y	┘	▁	Ế	Ó	ϕ		▩	☺
A			*	:	J	Z	j	z	╭	▂	Ë	Ò	Ω		▟	☻
B			+	;	K	[	k	{	╰	▃	Ē	Ö	Ç		○	⌶
C			,	<	L	¥	l	:	╮	▄	É	Õ	¿		●	⌺
D			−	=	M	]	m	}	╯	▅	È	Û	ì			웃
E			.	>	N	^	n	~	↑	▆	Ñ	Ǔ	ſ			÷
F			/	?	O	π	o		←	▇	Ň	Ú	£			

└CONTROL CODE

Command, Statement and Built in function

Command

No.	Command	Functions
1	LIST	Displays programs on screen.
2	LLIST	Prints programs on PRINTER.
3	SAVE	Records programs on cassette tapes
4	VERIFY	Compares programs in MEMORY and those recorded on cassette tapes.
5	LOAD	Loads cassette tape programs on MEMORY.
6	RUN	Runs programs.
7	CONT	Continues discontinued programs.
8	NEW	Clears variables and programs.
9	DELETE	Clears programs partially.
10	AUTO	Generates line numbers automatically.
11	RENUM	Renumbers line numbers.

STATEMENT

NO.	Statement	Functions
1	REM	Comment
2	STOP	Stops programs. Continuable by CONT.
3	END	Completes program run.
4	LET	Input substitution. LET omittable.
5	PRINT or ?	Displays on display.
6	LPRINT or L?	Prints on Printer.
7	INPUT	Input from key.
8	READ	Reads data from "DATA" statements.
9	DATA	Shows data to be read from "READ" statements.
10	RESTORE	Assigns positions of "DATA" statement to be read from "READ" statements.
11	DIM	Declares arrays.
12	ERASE	Clears declared array.
13	DEF FN	Defines user function.
14	GOTO	Branches to assigned address.
15	GOSUB	Go to subroutine.
16	RETURN	Returns from subroutine.
17	ON–GOTO ON–GOSUB	Selects line numbers to be branched. Selects line numbers to be branched.
18	FOR–TO– STEP–	Repeats statements between FOR and NEXT for a set number of times. STEP omissible.
19	NEXT	Assigns positions of repeating by "FOR" statement.

NO.	Statement	Functions
20	IF−THEN	Conditional branch.
21	CONSOLE	Assigns the ranges of screen scroll, click sound and character set.
22	CLS	Clears screen.
23	SCREEN	Shifts the screen.
24	COLOR	Color assignment.
25	PATTERN	Changes character sprite PATTERN.
26	CURSOR	Assigns display positions.
27	POSITION	Assigns coordinates.
28	PSET	Displays dots.
29	PRESET	Erases by dots.
30	LINE	Draws lines.
31	BLINE	Erases by lines.
32	CIRCLE	Draws circles.
33	BCIRCLE	Erases by circles.
34	PAINT	Paints enclosed extent.
35	SPRITE	Assigns sprite position, color and pattern.
36	MAG	Assigns sprite magnitude.
37	SOUND	Produces effective sound.
38	BEEP	Produces beep sound.
39	HCOPY	Prints text on screen on to printer.
40	CALL	Branches to machine language subroutine.
41	POKE	Writes in memory.
42	OUT	Outputs to output port.
43	VPOKE	Writes data in video RAM.

FUNCTION

NO.	Function	Functions
1	ABS(x)	Finds the absolute value of x.
2	RND(x)	Generates random numbers
3	SIN(x)	Finds the sine of x.
4	COS(x)	Finds the cosine of x.
5	TAN(x)	Finds the tangent of x.
6	ASN(x)	Finds the arc sine of x.
7	ACS(x)	Finds the arc cosine of x.
8	ATN(x)	Finds the arc tangent of x.
9	LOG(x)	Finds the natural logarithm of x.
10	LGT(x)	Finds the common logarithm of x.
11	LTW(x)	Finds the logarithm of x. with 2 as a base.
12	EXP(x)	Finds e
13	RAD(x)	Converts degrees into radians.
14	DEG(x)	Converts radians into degrees.
15	PI	Specifies the ratio of the circumference of a circle to its diameter.
16	SQR(x)	Finds the square root of x.
17	INT(x)	Finds the greatest integer not exceeding x.
18	SGN(x)	Specifies the positive and negative codes of x.
19	ASC(s)	Specifies the first code of character-string s by numeric values.
20	LEN(s)	Specifies the number of character-string s.
21	VAL(s)	Converts character-string s into numeric values.
22	CRH$($x$)	Specifies the corresponding character and functions of x.

NO.	Function	Functions
23	HEX$($x$)	Specifies the hexadecimal number character-string.
24	INKEY$($x$)	Checks whether or not key was pressed. When key is pressed, the character is given. (Null) if it is not pressed.
25	LEFT$(s,$x$)	Substitutes the character-string covering from the left of the character–string s to x places.
26	RIGHT$(s,$x$)	Substitutes the character-string covering from the right of the character-string s to x places.
27	MID$(s.$x$.$y$)	Substitutes the character-string of length y from the x places of the left of the character-string. y is omissible and in this case, substitutes from x place character to the end character.
28	STR$($x$)	Converts x into the character-string which indicates x.
29	TIME$	Determines the time of the inside clock.
30	PEEK(x)	Specifies the content of the x address of memory.
31	INP(x)	Specifies the input content of input port.
32	FRE	Specifies memory area space for users.
33	SPC(x)	Used by print statement. Provides space.
34	TAB(x)	Used by print statement. Assigns display positions.
35	STICK(n)	Shows the n direction of joysticks.
36	STRIG(n)	Shows the trigger button condition of joystick n.
37	VPEEK(x)	Specifies the content of VRAM x address.

ERROR MESSAGE

1. Display Format

(1) When Command or Statement was directly entered, errors occurred:

? [Message] error

(2) When errors occur during text run;

? [Message] error in [line No.]

(3) When error due to Input Statement is found in key Input Data:

? [Message]

MESSAGE	DESCRIPTION
System	System error due to Basic Interpreter Program. Generally this occurrence is impossible.
N-formula too Complex	Numeric values are too complicated.
S-formula too Complex	Character-String is too complicated.
Overflow	Values and operation results exceed permissible range.
Division by Zero	The denominator in division is 0.
Function Parameter	Function parameter is unusual.
String too long	The length of Character-String exceeds 255.
Stack overflow	Excessive use of parentheses ( ). Patterns to PAINT are too complicated. User define function calls itself.
Out of memory	Memory is insufficient. Text. Variable. Array.
Number of Subscripts	Number of subscripts is unusual.
Value of Subscript	Value of subscript is improper.
Syntax	Syntax Error
Command Parameter	Command Parameter is unusal.
Line number over	In AUTO or RENUM, line No. Exceeds 65535.
Illegal line number	Line No. is improper.
Line image too long	Line image is too long, (RENUM, etc.)
Undefined line number	Line No. is undefined. (RENUM, GOTO, GOSUB, IF-THEN, RESTORE, RUN)
Type mismatch	The type of substituting side and that of substituted side do not match. (Values, strings)

MESSAGE	DESCRIPTION
Out of DATA	Reading by READ Statement was attempted but DATA of DATA statement is unavailable.
RETURN without GOSUB	RETURN statement was executed without GOSUB.
GOSUB nesting	GOSUB nesting exeeded 4 levels.
NEXT without FOR	For statement corresponding to NEXT is not available.
FOR nesting	FOR~NEXT nesting exceeded 4 levels.
Statement Parameter	Statement parameter is unusual.
Can't continue	Can't continue by CONT statement.
FOR variable name	FOR statement loop variable is not numeric variable. (Character string or array)
Array name	DIM statement parameter is not in array.
Redimensioned array	Dual array difinition was attempted.
Undefined Array	Erase of undefined array was attempted.
No Program	SAVE was attempted despite unavailability of program in text.
Memory writing	Memory writing error (At the time of LOAD)
Device not ready	Printer is not connected or in trouble.
Undefined Function	Undefined user function was called.
Verifying	Errors in comparison with tape programs.
Illegal direct	Direct statement run is impossible.

MESSAGE	DESCRIPTION
Redo from start	INPUT DATA of input statement is unusual. Redo input from the start.
Extra ignored	INPUT DATA of input statement is unusual. Extra data was entered. Extra data was ignored.
Unprintable	Errors other than the above.

SAMPLE PROGRAM

CHECKERED PAINT

```
 10  SCREEN 2 , 2 : CLS
 50  X=10 : Y=10 : XX=250 : YY=190
100  REM VERTICAL LINE
110  FOR I=1 TO 12
120  LINE ( X , Y ) - ( X , Y Y ) , 1
130  X=X+20
140  NEXT I
200  REM HORIZONTAL LINE
210  X=0
220  FOR I=1 TO 10
230  LINE ( X , Y ) - ( XX , Y ) , 1
240  Y=Y+20
250  NEXT I
300  REM PAINT
310  A=RND (1) *240
320  B=RND (1) *185
330  C=RND (1) * 15
340  PAINT (A , B) , C
350  GOTO 300
```

LINE

```
10  SCREEN2,2:CLS:COLOR,15
15  FOR I=0TO30
20  A=INT(RND(1)*16)
30  B=INT(RND(1)*128):C=INT(RND(1)*96)
40  D=INT(RND(1)*256):E=INT(RND(1)*192)
50  LINE(B,C)-(D,E),A,B
55  NEXTI
100 GOTO 10
```

LINEBF

```
10  SCREEN 2,2:CLS
20  A=INT(RND(1)*16)
30  B=INT(RND(1)*128):C=INT(RND(1)*96)
40  D=INT(RND(1)*256):E=INT(RND(1)*192)
50  LINE(B,C)-(D,E),A,BF
100 GOTO 20
```

CIRCLE 1

```
10 SCREEN2,2:CLS:COLOR15
20 FOR R=1TO96
30 C=INT(RND(1)*16)
40 CIRCLE(128,96),R,C,1,0,1,
50 NEXT R
60 GOTO 20
```

CIRCLE 2

```
10 SCREEN2,2:CLS:COLOR15
20 FOR R=1TO96 STEP 5
30 C=INT(RND(1)*16)
40 CIRCLE(128,96),R,C,1,0,1,
50 NEXT R
60 PAINT(0,0),5
```

CIRCLE 3

```
10 SCREEN2,2:CLS
20 X=INT(RND(1)*256)
30 Y=INT(RND(1)*192)
40 C=INT(RND(1)*16)
50 R=INT(RND(1)*20)
60 CIRCLE(X,Y),R,C,1,0,1
70 GOTO 20
```

BCIRCLE

```
10 SCREEN2,2:CLS:COLOR15
20 FOR R=1TO96 STEP 10
30 C=INT(RND(1)*16)
40 CIRCLE(128,96),R,C,1,0,1
50 NEXT R
60 FOR I=91 TO 1 STEP-10
70 BCIRCLE(128,96),I,,1,0,1
80 NEXTI
100 GOTO 20
```

Sprite Sample Program

```
 10  M=1
 20  SCREEN 2,2:CLS
 30  MAG M:C=RND(1)*13+1
 40  CURSOR 10,10:PRINT CHR$(17);"MAG";M
 50  FOR Y=0 TO 191 STEP 4
 60  PATTERN S#0,"00193F3C1C0D0F7B"
 70  PATTERN S#1,"0C0F0F0F07031B07"
 80  PATTERN S#2,"00CCFE9E9CD878EC"
 90  PATTERN S#3,"1AFAF8F0EC7C3800"
100  Y1=Y:GOSUB 200
110  PATTERN S#0,"00193F3C1C0D0F1B"
120  PATTERN S#1,"2C2F0F071B1F0E00"
130  PATTERN S#2,"00CCFE9E9CD878EF"
140  PATTERN S#3,"18F8F8F8F0606C70"
150  Y1=Y+2:GOSUB 200
160  NEXT Y
170  M=M+2:IF M>3 THEN M=1
180  GOTO 20
200  SPRITE 0,(120,Y1),0,C
210  SPRITE 0,(120,Y1+1),0,C
220  RETURN
```

PAINT

```
10 SCREEN2,2:CLS
20 FOR I=0 TO 255 STEP 16
30 LINE(I,0)-(I,191):NEXTI
40 FOR I=0 TO 191 STEP 16
50 LINE(0,I)-(255,I):NEXTI
60 C=INT(RND(1)*16)
70 X=INT(RND(1)*256):Y=INT(RND(1)*192)
80 PAINT(X,Y),C
90 GOTO 60
```

END

John Sands

SEGA®