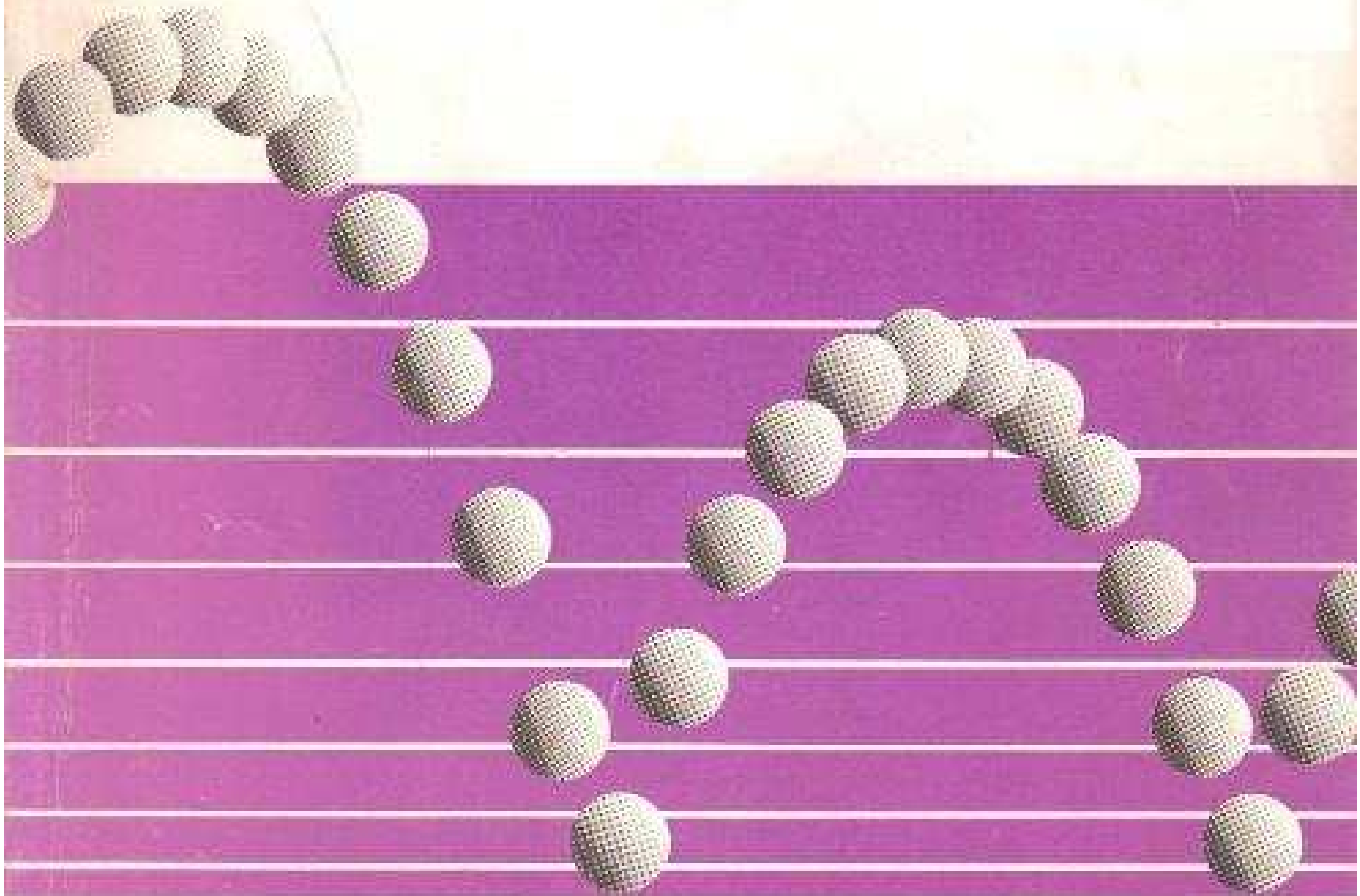
VX-2

Super college personal computer

Owners's manual



CASIO

GUIDELINES LAID DOWN BY FCC RULES FOR USE OF THE UNIT IN THE U.S.A. (not applicable to other areas).

NOTICE

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

FCC WARNING

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Proper connectors must be used for connection to host computer and/or peripherals in order to meet FCC emission limits.

Connector	FA-6	This unit to Personal computer
		This unit to Data recorder
		This unit to Graphic printer
		This unit to CASIO FX-850P/FX-880P
		This unit to CASIO FP-100
		This unit to CASIO FP-40 via SB-43

PERSONAL COMPUTER
VX2
OWNER'S MANUAL

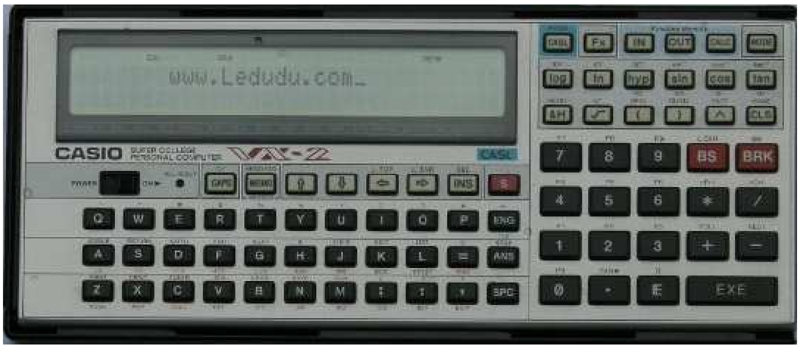
CASIO®

PRECAUTIONS

This computer is a product of CASIO's high level of electronics engineering, testing, and quality control. The following points should be carefully noted to allow this unit to provide the years of trouble free operation for which it is designed.

- This unit is constructed of precision electronic components and should never be disassembled, dropped, or otherwise subjected to strong impact. Strong shocks can cause termination of program execution or alteration of the unit's memory.
- Do not use or store this unit in areas subjected to high temperatures, humidity or dust.
- Display response may become slow or fail completely at extremely low temperatures. Normal operation should resume after the unit reaches normal temperature.
- The connectors of this unit are designed exclusively for connection of the specified FA-6 expansion units only.
- The display may become dim when the buzzer sounds, but this does not indicate malfunction and is no cause for worry.
- Batteries should be replaced as soon as possible after weakened batteries are indicated by a dim display during normal operation.
- Replace batteries at least once every two years even if the unit is not used during this period. Dead batteries left in the unit may cause serious damage due to fluid leakage and should be removed as soon as possible.
- Keep the connector of the unit covered with the connector cap whenever the unit is not connected to an expansion unit, and avoid touching the connector.
- Strong static electrical charge may cause alteration of memory contents or key operation failure. If this situation should occur, remove the batteries and load them again.
- Always ensure that the power supply of this unit is switched OFF before connecting peripheral devices.
- Never use thinner, benzine, or other volatile agents for cleaning the exterior of the unit. Use a soft cloth dipped into a mild solution of water and a neutral detergent, and wring the cloth out completely.
- Do not switch the power of the unit OFF during program execution or during calculations.
- When a malfunction occurs, contact the store where the computer was purchased or a nearby dealer.
- Before seeking service, please read this manual again, check the power supply, check the program for logic errors, etc.

Contents



PART 6

BASIC PROGRAMMING

Standard BASIC is employed as the programming language for this unit, and this section covers application of the BASIC language.

6-1 FEATURES OF BASIC

1. BASIC is much easier to use than other programming languages such as FORTRAN, making it suitable even for novices.
2. Writing programs is also easier because program creation, editing and execution are all performed by interacting with the computer itself.

The following functions are also available:

1. High-precision calculations are made possible by display of numeric values with 10-digit mantissas and 2-digit exponents (13-digit mantissa and 2-digit exponent for internal operations).

2. A wide selection of built-in functions makes operation easier.

① Standard mathematical functions

SIN COS TAN ASN ACS ATN LOG LN EXP SQR ABS SGN
INT FIX FRAC PI ROUND RAN# DEG

② Powerful string handling functions

CHR$ STR$ MID$ LEFT$ RIGHT$ HEX$ DMS$ ASC VAL LEN

③ High level mathematical functions

POL REC NCR NPR HYPSIN HYPCOS HYPTAN HYPASN HYPACS
HYPATN CUR

3. 10 independent program areas

Up to ten programs can be stored independently in memory at the same time (P0 ~ 9).

4. Extended variable names

Variable names up to 15 characters long can be used, making it possible to use names that makes contents easy to understand.

5. Powerful debugging function

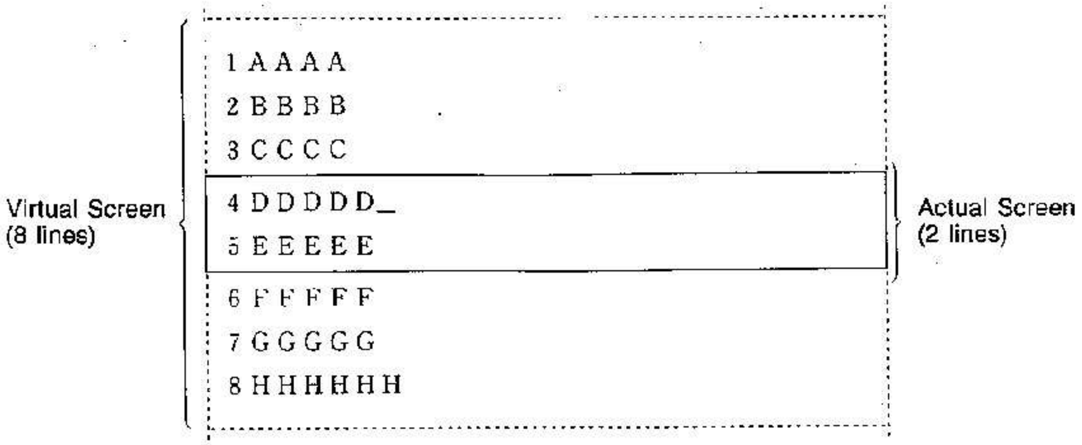
A TRON command displays the number of the program line currently being executed, making it possible to easily trace execution and locate mistakes in programming.

6. Powerful screen editor

Programs can be easily modified and corrected on the screen.

7. Virtual screen function

Though the actual physical display of the unit has a 32-column × 2-line capacity, the virtual screen is 32 columns × 8 lines. The virtual screen can be easily scrolled using the cursor keys.



8. Expanded file management
Such standard BASIC commands as OPEN, CLOSE, INPUT# and PRINT# are all available for data file reading and writing.

6-2 BASIC PROGRAM CONFIGURATION

6-2-1 BASIC Program Format

The following is a typical BASIC program which calculates the volume of a cylinder.

EXAMPLE:

```
10  REM CYLINDER
20  R=15
30  INPUT "H=" ; H
40  V=PI*R^2*H        (PI indicates π)
50  PRINT "V=" ; V
60  END
```

As can be seen, the BASIC program is actually a collection of lines (six lines in the above program). A line can be broken down into a line number and a statement.

$\underbrace{20}_{\text{Line number}} \quad \underbrace{R=15}_{\text{Statement}}$

Computers execute programs in the order of their line numbers. In the sample program listed above, the execution sequence is 10, 20, 30, 40, 50, 60. Program lines can be input into the computer in any sequence, and the computer automatically arranges the program within its memory in order from the smallest line number to the highest. Lines can be numbered using any value from 1 through 65535.

```
20  R=15                          10  REM CYLINDER
40  V=PI*R^2*H                    20  R=15
60  END                    ➡      30  INPUT "H=" ; H
10  REM CYLINDER                  40  V=PI*R^2*H
30  INPUT "H=" ; H                50  PRINT "V=" ; V
50  PRINT "V=" ; V                60  END
```

Input sequence — Memory contents

Following the line number is a statement or statements which actually tell the computer which operation to perform. The following returns to the sample program to explain each statement in detail

10 REM CYLINDER.............	REM stands for "remarks". Nothing in this line is executed.
20 R=15.............................	Assigns the constant 15 (radius) to variable R.
30 INPUT "H=" ; H...........	Displays H ? to prompt a value input for height.
40 V=PI*R^2*H.............	Calculates volume (V) of cylinder.
50 PRINT "V=" ; V...........	Prints result of line 40.
60 END................................	Ends program.

As can be seen, a mere six lines of programming handles quite a bit of data. Multiple BASIC program lines can also be linked into a single line using colons.

EXAMPLE:

```
100  R=15:A=7:B=8
```

Such a program line is known as a "multistatement".

Details concerning the other operations contained in the above program can be found in other sections of PART 6.

6-3 BASIC PROGRAM INPUT

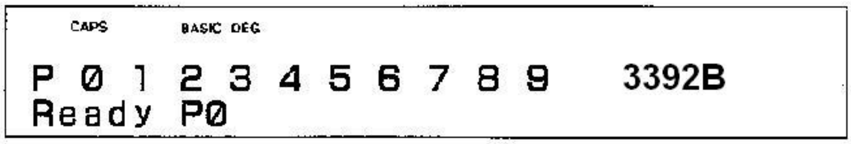
6-3-1 Preparation

First switch the power of the computer ON. At this time, the display should appear as illustrated below.

```
   CAPS   CAL     DEG

_
```

This is the CAL mode, so the operation [MODE] [1] should first be performed to allow input of BASIC programs. The display should now appear as illustrated below.



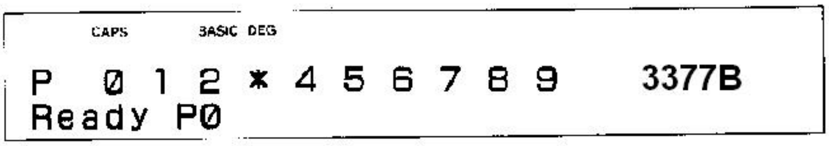
```
   CAPS      BASIC DEG

P 0 1 2 3 4 5 6 7 8 9    3392B
Ready P0
```

Note that the indicator CAL has been replaced by BASIC to indicate the BASIC mode. This is the mode used for BASIC program input. The other indicators on the display in the BASIC mode have the following meanings.

P	: Program area
0~9	: Program area numbers. The numbers of program areas which already contain programs are replaced by asterisks.

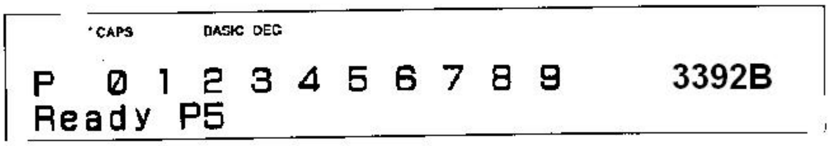
EXAMPLE:
Program stored in area 3


CAPS BASIC DEG
P 0 1 2 * 4 5 6 7 8 9 3377B
Ready P0

3392B	:	Capacity (number of bytes) remaining in area for writing programs and data (free area). The value will be 3392B when there are no programs or data stored in memory, with this value decreasing as storage space is used.
Ready P0	:	Current program area=area 0. The current program area can be switched by pressing [SHIFT] followed by the desired program area.

EXAMPLE:
Switching to program area 5

[SHIFT] [P5]

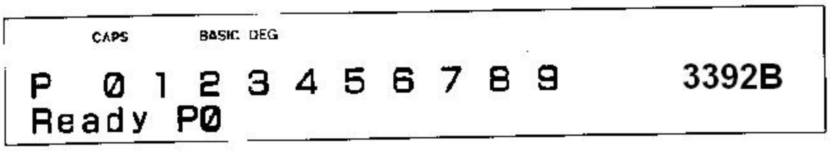

CAPS BASIC DEG
P 0 1 2 3 4 5 6 7 8 9 3392B
Ready P5

Previously stored programs can be deleted using one of two different procedures.

NEW	:	Deletes program stored in current program area only.
NEW ALL	:	Clears all programs stored in memory.

EXAMPLE:

NEW ALL [EXE]


CAPS BASIC DEG
P 0 1 2 3 4 5 6 7 8 9 3392B
Ready P0

This operation clears all programs stored in memory and returns to current program area to 0.

6-3-2 Program Input

The following input procedure inputs the sample program for calculation of the volume of a cylinder.

[1] [0] [R] [E] [M] [SPC] [C] [Y] [L] [I] [N] [D] [E] [R] [EXE]
[2] [0] [R] [=] [1] [5] [EXE]
[3] [0] [I] [N] [P] [U] [T] [SHIFT] ["] [H] [=] [SHIFT] ["] [;] [H] [EXE]
[4] [0] [V] [=] [P] [I] [*] [R] [^] [2] [*] [H] [EXE]
[5] [0] [P] [R] [I] [N] [T] [SHIFT] ["] [V] [=] [SHIFT] ["] [;] [V] [EXE]
[6] [0] [E] [N] [D] [EXE]

Note that the [EXE] key is pressed at the end of each line. A program line is not entered into memory unless the [EXE] key is pressed.

ONE-KEY INPUT
The one-key BASIC commands help to make program input even easier.

EXAMPLE:
Line 30 input.

[3] [0] [SHIFT] INPUT [SHIFT] " [H] [=] [SHIFT] " [;] [H] [EXE]

6-3-3 Program Editing

The procedure used for making corrections or changes to a program depends upon what step of program input the changes are to be made.

① Changes in a line before [EXE] is pressed
② Changes in a line after [EXE] is pressed
③ Changes within a program already input
④ Changes within a program following the EDIT command

1. Changes in a line before [EXE] is pressed

EXAMPLE:
20 E = 15 mistakenly input for 20 R = 15

```
10 REM CYLINDER
20 E=15_
```

[⇦] [⇦] [⇦] [⇦]

```
10 REM CYLINDER
20 E=15
```

(Move cursor to E)

[R]

```
10 REM CYLINDER
20 R=15
```

(Input correct character)

[EXE]

```
20 R=15
_
```

(Editing complete)

Note that once the desired changes are made, the [EXE] key must be pressed to store the entered line into memory.

2. Changes in a line after [EXE] is pressed

EXAMPLE:
40 V = P1 ∗ R ^ 2 ∗ H mistakenly input for 40 V = PI ∗ R ^ 2 ∗ H

```
40 V=P1*R^2*H
_
```

[⇧] [⇨] [⇨] [⇨] [⇨] [⇨] [⇨]

```
40 V=P1*R^2*H
```

(Move cursor to 1)

[I]

```
40 V=PI*R^2*H
```

(Input correct character)

[EXE]

```
40 V=PI*R^2*H
_
```

(Editing complete)

Again, the [EXE] key must be pressed to store the corrected line into memory after changes are made.
Procedures 1 and 2 show the procedures for simple exchanges of one character for another.

Characters can also be inserted and deleted using the following procedures.

i) Insert

40 V=PI*R2*H mistakenly input for 40 V=PI*R^2*H

	40 V=PI*R2*H_	
⇦ ⇦ ⇦	40 V=PI*R2*H	(Move cursor to location of insertion)
INS	40 V=PI*R_2*H	(Open one space)
^ EXE	40 V=PI*R^2*H _	(Input correct character and press EXE)

ii) Delete

40 V=PI*RR^2*H mistakenly input for 40 V=PI*R^2*H

	40 V=PI*RR^2*H_	
⇦ ⇦ ⇦ ⇦ ⇦	40 V=PI*RR^2*H	(Move cursor to character to be deleted)
SHIFT DEL	40 V=PI*R^2*H	(Delete character)
EXE	40 V=PI*R^2*H _	(Editing complete)

The BS key works rather similarly to the SHIFT DEL operation. The difference between the two operations is as follows.

Difference Between SHIFT DEL and BS

- SHIFT DEL

Deletes the character at the current cursor location and shifts everything to the right of the cursor one space to the left.

A B C D E F G→SHIFT DEL→A B C E F G
↑Cursor

- BS

Deletes the character to the left of the current cursor location and shifts everything from the cursor position right one space to the left.

A B C D E F G→BS→A B D E F G

3. Changes within a program already input

The LIST command displays the program stored in the current program area from beginning to end.

LIST EXE

```
10  REM  CYLINDER
20  R=15
```

⋮

```
60  END
Ready  P0
```

The last line of the program is displayed when the LIST operation is complete.

⇧ ⇧ ⇧ ⇧ ⇧ ⇧

```
10  REM  CYLINDER
20  R=15
```

The 8-line virtual screen of the computer now makes it possible to use the cursor keys to scroll to preceding lines not shown on the display (see page 7).

(8-line virtual screen)

```
Ready P0
10   REM CYLINDER
20   R=15
30   INPUT "H=" ; H
40   V=PI*R^2*H
50   PRINT "V=" ; V
60   END
Ready P0
```

2-line display
⇧ ⇩ keys move display

When a program greater than eight lines is stored in memory, the LIST operation should be performed by specifying the line numbers to be displayed.

EXAMPLE:
Displaying from line 110 to line 160 on the virtual screen.
LIST 110 — 160 EXE

(8-line virtual screen)

```
LIST 110—160
110   A=1
120   FOR I=1 TO 100
130   B=LOG ( I )
140   PRINT B
150   NEXT I
160   E=A*B
Ready P0
```

2-line display

Changes can be made in a program displayed by the LIST operation by using the same procedures outlined for 1 and 2 above.

* The BRK key can be used to terminate the LIST operation. The STOP key suspends the operation, and listing can be resumed by pressing EXE.

4. Changes within a program following the EDIT command

The EDIT command makes it easier to edit or review programs already stored in memory.

EDIT [EXE]

```
CAPS   BASIC DEG                    EDIT
10 REM CYLINDER
20 R=15
```

From this display, [⇩] (or [EXE]) advances to the following line, while [⇧] (or [SHIFT] [EXE]) returns to the previous line.

[⇩] [⇩]

```
30  INPUT"H=";H
40  V=PI+R^2*H
```

(Displays lines 30 and 40)

Here, a correction will be made in line 40.

[⇩]

```
40  V=PI+R^2*H
50  PRINT"V=";V
```

(Displays line 40 at upper line of display)

[⇨]

```
40  V=PI+R^2*H
```

(Enables program editing)

[⇨] [⇨] [⇨] [⇨] [⇨] [⇨] [⇨] [✱] [EXE]

```
40  V=PI*R^2*H
50  PRINT"V=";V
```

(Correction)

[BRK]

```
Ready P0
```

([BRK] key exits EDIT mode)

6-4 BASIC PROGRAM EXECUTION

6-4-1 Program Execution

Once a BASIC program is stored in memory, it can be executed using one of the two following procedures.

1. Using [SHIFT] (program area) in CAL mode

   EXAMPLE: [SHIFT] [9]
   Executes the program in program area 9.

2. Entering RUN command in BASIC mode

   EXAMPLE: RUN [EXE]
   Executes the program in the current program area.

Execute the program input in the previous section to determine the volume of a cylinder with a height of 10 (radius fixed as 15).

RUN [EXE]

```
RUN
H=?_
```

(Executes program)

10 [EXE]

```
H=?10
V= 7068.583471
```

[EXE]

```
V= 7068.583471
Ready P0
```

(Corresponding volume displayed when height is entered.)

Display of the volume when this program is executed causes the STOP symbol to appear in the upper right of the display. This symbol indicates that program execution has been suspended because of execution of the PRINT command. Program execution can be resumed at this time by pressing the EXE key again. Ending a PRINT statement with a semicolon causes execution to continue when the PRINT statement is executed, causing the display of the next PRINT statement to appear immediately following the previous display.

EXAMPLE 1:

```
10  PRINT "BASIC  "
20  PRINT "PROGRAM"
30  END
```

Execution of this program results in the following display.

RUN EXE

```
RUN
BASIC
```

EXE

```
BASIC
PROGRAM
```

EXE

```
PROGRAM
Ready P0
```

EXAMPLE 2:

```
10  PRINT "BASIC  ";
20  PRINT "PROGRAM"
30  END
```

Including a semicolon at the end of the first PRINT statement produces the following display.

RUN EXE

```
RUN
BASIC PROGRAM
```

EXE

```
BASIC PROGRAM
Ready P0
```

6-4-2 Errors

At times, the results produced by a program are not what is expected. Such irregular executions can be broadly divided under two major classifications.

① Executions that produce errors
  Simple programming errors
  Program logic errors
② Irregular execution that do not produce errors
  Mostly program logic errors

1. Executions that produce errors

i) Simple programming errors

This is the most common type of program error and is generally caused by mistakes in program syntax. Such errors result in the following message being displayed:

SN error P0—10

This message indicates that a syntax error has been detected in line 10 of the program stored in program area 0. The indicated program line should be checked and corrected to allow proper execution.

ii) Program logic errors

This type of error is generally caused by such illegal operations as division by zero or LOG(0). Such errors result in the following message being displayed:

MA error P0—10

As before, this message indicates that a mathematical error has been detected in line 10 of the program stored in program area 0. In this case, however, program lines related to the indicated program line as well as indicated program line itself should be examined and corrected. When an error message is displayed, the following two operations can be used to display the line number in which the error was detected.

```
{ LIST 10 EXE
{ EDIT 10 EXE

{ LIST . EXE
{ EDIT . EXE
```

* The periods contained in **LIST .** and **EDIT .** instruct the computer to automatically display the last program line executed.

* Change to the BASIC mode if a BASIC program was executed in the CAL mode.

2. Irregular execution that do not produce errors

Such errors are also caused by a flaw in the program, and must be corrected by executing the LIST or EDIT command to examine the program to detect the problem. The TRON command can also be used to help trace the execution of the program.
Entering **TRON** EXE causes the TR symbol to appear on the display to indicate that the trace mode is ON. Now executing a BASIC program displays the program area and line number as execution is performed, and halts execution until EXE is pressed. This allows confirmation of each program line, making it possible to quickly identify problem lines. Executing **TROFF** EXE cancels the trace mode.

6-5 COMMANDS

Though there are a variety of commands available in BASIC for use in programs, the nine fundamental commands listed below are the most widely used.

The following program reads data items contained within the program itself, with the number of data items read being determined by input from an operator. The operator may input any value, but note that values greater than 5 are handled as 5 (because there are only 5 data items in line 180). The program then displays the sum of the data read from line 180, followed by the square root and cube root of the sum. Program execution is terminated when a zero is entered by the operator.

```
10 S=0...............................Clears current total assigned to S
20 RESTORE...........................Specifies read operation should begin with first data item
30 INPUT N...........................Input of number of data items to be read
40 IF N>5 THEN N=5...................Input of value greater than 5 should be treated as 5
50 IF N=0 THEN GOTO 130..............Jump to line 130 when input is zero
60 FOR I=1 TO N ──────────────┐
70 READ X   Data read         │
                              ├──This section repeated the number of times specified by
80 S=S+X    Sum of data calculation  operator input in line 30
90 NEXT I ────────────────────┘
```

```
100  GOSUB  140...................................Branch to subroutine starting from line 140
110  PRINT S ; Y ; Z..............................Displays contents of variables S, Y, Z
120  GOTO 10......................................Jump to line 10
130  END..........................................Program end
140  REM  SQUARE ROOT/CUBE ROOT...Remarks
150  Y=SQR  S.....................................Square root calculation
160  Z=CUR S......................................Cube root calculation
170  RETURN.......................................Return to the statement following the statement which
                                                  called the subroutine
180   DATA 9, 7, 20, 28, 36.......................Data read by READ statement in line 70
```

① REM

The REM command (line 140) is actually short for the word "remarks". The computer disregards anything following a REM command, and so it is used for such purposes as labels in order to make the program list itself easier to follow. Note that a single quotation mark (SHFT ') can be used in place of the letters "REM".

② INPUT

The INPUT command (line 30) is used to allow input from the computer's keyboard during program execution. The data input are assigned to a variable immediately following the INPUT command. In the above example, input numeric data are assigned to the variable N. Note that a string variable must be used for string input.

EXAMPLE:

10 INPUT A$ (string input)

③ PRINT

The PRINT command (line 110) is used to display data on the computer's display. In this example, this command is used to display the results of the sum, square root, and cube root calculations. When the data are displayed, the STOP symbol appears and program execution is suspended. Execution can be resumed by pressing the EXE key.

④ END

The END command (line 130) brings execution of the program to an end, and can be included anywhere within a program.

⑤ IF ~ THEN ~

The IF/THEN command (lines 40 and 50) is used for comparisons of certain conditions, basing the next operation upon whether the comparison turns out to be true or false. Line 40 checks whether or not value assigned to N is greater than 5, and assigns a value of 5 to N when the original value is greater. When a value of 5 or less is originally assigned to N, execution proceeds to the next line, with N retaining its original value. Line 50, checks whether or not the value assigned to N is zero. In the case of zero, program execution jumps to line 130, while execution proceeds to the next line (line 60) when N is any other value besides zero.

* Line 50 can also be abbreviated as follows:

50 IF N=0 THEN 130

* Program areas can also be specified as jump destinations:

IF A = 1 THEN GOTO #2 (Program stored in program area 2 executed when A equals 1)

⑥ GOTO

The GOTO command (lines 50 and 120) performs a jump to a specified line number or program area. The GOTO statement in line 120 is an unconditional jump, in that execution always returns to line 10 of the program whenever line 120 is executed. The GOTO statement in line 50, on the other hand, is a conditional jump, because the condition of the IF~THEN statement must be met before the jump to line 130 is made.

* Program area jumps are specified as GOTO #2 (to jump to program area 2).

⑦ FOR/NEXT

The FOR/NEXT combination (lines 60 and 90) forms a loop. All of the statements within the loop are repeated the number of times specified by a value following the word "TO" in the FOR statement. In the example being discussed here, the loop is repeated N number of times, with the value of N being entered by the operator in line 30.

⑧ READ/DATA/RESTORE

These statements (lines 70, 180, 20) are used when the amount of data to be handled is too large to require keyboard input with every execution. In this case, data are included within the program itself. The READ command assigns data to variables, the DATA statement holds the data to be read, and the RESTORE command is used to specify from which point the read operation is to be performed.

In the sample program here, the READ command reads the number of data items specified by the input for variable N. Though the DATA statement holds only five data items, the RESTORE command in line 20 always returns the next read position to the first data item, the READ statement never runs out of data to read.

⑨ GOSUB/RETURN

The GOSUB/RETURN commands (lines 100 and 170) are used for branching to and from subroutines. Subroutines (lines 140 through 170) are actually mini programs within the main program, and usually represent routines which are performed repeatedly at different locations within the main program. This means that GOSUB/RETURN makes it possible to write the repeated operation once, as a subroutine, instead of writing each time it is needed within the main program.

EXAMPLE:

```
120  GOSUB 1000
       ⋮
370  GOSUB 1000
       ⋮
```

Execution of the RETURN statement at the end of a subroutine returns execution of the program back to the statement following the GOSUB command. In this sample program, execution returns to line 110 after the RETURN command in line 170 is executed.

* GOSUB routines can also be used to branch to other program areas, as in GOSUB #3 (branches to program area 3). Note, however, that a return must be made back to the original program area using the RETURN command before an END command is executed.

* See PART 10 COMMAND REFERENCE for further details on BASIC commands.

6-6 OPERATORS

The following are the operators used for calculations which involve variables.

Operators	Arithmetic operators	Signs	+, −
		Addtion	+
		Subtraction	−
		Multiplication	*
		Division	/
		Power	^
		Integer division	¥
		Integer remainder of integer division	MOD
	Relational operators	Equal to	=
		Does not equal	<>, ><
		Less than	<
		Greater than	>
		Less than or equal to	=<, <=
		Greater than or equal to	=>, >=
	Logical operators	Negation	NOT
		Logical product	AND
		Logical sum	OR
		Exculsive OR	XOR
	String operator		+

1. Arithmetic Operators (+, −, *, /, ^ ,¥, MOD)

- Fractions are truncated in ¥ and MOD calculations, when the operands on both sides of the operator are not integers.
- In ¥ and MOD calculations, the division is performed with the absolute values of both operands. In integer division (¥), the quotient is truncated to an integer. With the MOD operator, the remainder is given the sign of the dividend.

EXAMPLES:

−15¥7 = −2

−15 MOD7 = 1

$$-15\div 7=\underbrace{-2}_{-15\text{¥}7}\cdots\cdots\underbrace{-1}_{-15\text{MOD}7}$$

- The length of an expression is limited to 255 characters.

2. Relational Operators (=, <>, ><, <, >, =<, <=, =>, >=)

Relational operations can be performed only when the operators are both strings or numeric values.

With strings, character codes are compared one-by-one from the beginning of the strings. This is to say that the first position of string A is compared with the first position of string B, the second position of string A with the second position of string B, etc. The result of the comparison is based upon the character codes of the first difference between the strings detected, regardless of the length of the strings being compared.

EXAMPLES:

STRING A	STRING B	RESULT	
ABC	ABC	A=B	
ABC	ABCDE	A<B	
ABC	XYZ	A<B	(character code for A less than that for X)
XYZ	ABCDE	A>B	(character code for X greater than that for A)

A result of −1 is returned when the result of a relational operation is true (conditions met), while 0 is returned when the result is false (conditions not met).

EXAMPLE:

```
10  PRINT 10>3.................................−1 returned because 10>3 is true
20  PRINT 7<1...................................0 returned because 7<1 is false
30  PRINT "ABC" = "XYZ"............0 returned because ABC = XYZ is false
40  END
```

3. Logical Operators

The operands of logical operations are truncated to integers and the operation is performed bit-by-bit to obtain the result.

Negation

X	NOT X
0	1
1	0

Logical product

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

Logical sum

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

Excluxive OR

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

4. String Operators (+)

Strings may be concatenated using a + sign.
The result of the operation (including intermediate results) may not exceed 255 characters.

EXAMPLE:

```
A$="AD"+"1990"
```

The above example results in the string "AD1990" being assigned to variable A$.

5. Order of Operations

Arithmetic, relational and logical operations are performed in the following order of precedence:

1. (,)
2. Scientific function
3. Power
4. Sign (+, −)
5. ∗, /, ¥, MOD
6. Addition and subtraction
7. Relational operators
8. NOT
9. AND
10. OR, XOR

Operations are performed from left to right when the order of precedence is identical.

6-7 CONSTANTS AND VARIABLES

6-7-1 Constants

The following shows the constants included in the sample program on page 46:

PROGRAM	CONSTANTS
`20 R=15`	15
`30 INPUT "H=" ; H`	"H="
`40 V=PI*R^2*H`	2
`50 PRINT "V=" ; V`	"V="

Of these, 15 and 2 are numeric constants, while "H=" and "V=" are string constants.

Numeric Constants

• Numeric Notation

① Decimal notation
② Hexadecimal notation

• Numeric Value Precision

① Internal numeric operations
12-digit mantissa, 2-digit exponent (PI = 11 digits: 3.1415926536; displayed in 10 digits: 3.141592654)
② Results
10-digit mantissa, 2-digit exponent (exponent rounded to 10 digits)
③ Number of characters per line
255 characters per line
④ Result Display

Integers less than 1×10^{10}	: Integer display
Decimal portion less than 11 digits	: Decimal display
Other	: Exponential display
Display rounding	: Results are rounded off at the 10th digit and displayed.

String Constants

Strings within quotation marks (i.e. "ABC", "H=")
Closing quotation marks at the end of a line may be omitted (10 PRINT "TEST" can be written 10 PRINT "TEST)
Multiple strings can be connected with a "+" sign.

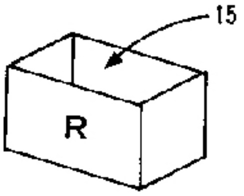
6-7-2 Variables

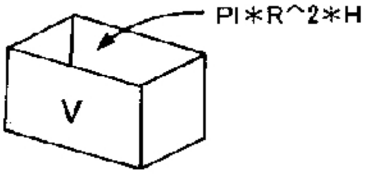
Numeric Variables

The following shows the numeric variables included in the sample program on page 46:

PROGRAM	NUMERIC VARIABLES
20 R=15	R
30 INPUT "H=" ; H	H
40 V=PI*R^2*H	V

Numeric variables are so named because their contents are handled as numbers. Numeric variable names can be up to 15 characters long, and are used within programs to store calculation results or constants in memory. In the sample program, the value 15 is stored in H, while V, which is the result of the calculation, holds the value which represents the volume of the cylinder. As can be seen, assignment to a variable is performed using the "=" symbol. This differs from an equal sign in that it declares that what is to the right should be assigned to what is to the left. Actually, a variable can be thought of as a kind of box as illustrated below:





String Variables

Another type of variable is known as a string variable, which is used to store character string data. String variable names are indicated by "$" following the name.

EXAMPLE:

```
10  A$="AD"..................................Assigns "AD" to string variable A$.
20  INPUT "YEAR=" ; B$.............Assigns keyboard input to variable B$.
30  C$=A$+B$..............................Assigns combination of A$ and B$ to C$.
40  PRINT C$.................................Displays contents of C$.
50  END
```

In the above example program, entering a year such as 1990 in line 20 results in a display of AD1990 in line 40.

* With string variables, "+" can be used to connect two strings.

* Note here that strings cannot be assigned to numeric variables such as A, and numeric values cannot be assigned to string variables such as A$.

Array Variables

Both numeric variables and string variables can store only one data item per variable. Because of this, large amounts of data are better handled using array variables (usually referred to as simply "arrays"). Before an array variable can be used within a program, a DIM statement must appear at the beginning of the program to "declare" to the computer that an array variable is to be employed.

EXAMPLE:

Declare array variable A for storage of 21 data items.

```
10  DIM A (20)
```

* The above format is used to declare "ARRAY VARIABLE NAME (NUMBER OF ELEMENTS)".

* A declared value of 20 makes it possible to store 21 data items (see page 63 for details).

EXAMPLE:

Find the sum (X) and the sum of the squares (Y) for the following 10 data items:

10, 12, 9, 11, 13, 14, 11, 12, 9, 10

The following program would be required to perform the calculation if only simple numeric variables are used:

```
10  A1=10 : A2=12 : A3=9 : A4=11 : A5=13          } Assigns values to each variable
20  A6=14 : A7=11 : A8=12 : A9=9 : A10=10          }
30  X=A1+A2+A3+A4+A5+A6+A7+A8+A9+A10......Calculates sum
40  Y=A1^2+A2^2+A3^2+A4^2+A5^2+A6^2+
        A7^2+A8^2+A9^2+A10^2.............................Calculates sum of squares
```

The program becomes much simpler when an array is used.

```
10  DIM A (10)....Declares array
20  A (1) =10 : A (2) =12 : A (3) =9 : A (4)=11 : A (5)=13    } Assigns values to array
30  A (6) =14 : A (7) =11 : A (8)=12 : A (9)=9 : A (10)=10   }
40  X=0 : Y=0
50  FOR I=1 TO 10              }
60  X=X+A ( I ) : Y=Y+A ( I )^2  }  Calculates sum and sum of squares
70  NEXT I                     }
```

At first glance, the array may appear to be rather troublesome to use, but it actually makes programming simpler when large volumes of data are being assigned.

EXAMPLE:
100 data items

Numeric variables

```
 10  A1 = 61 : A2 = 38 : A3 = 90 : A4 = 37 : A5 = 99        )
 20  A6 = 12 : A7 = 17 : A8 = 94 : A9 = 39 : A10 = 75        |
 30  A11 = 24 : A12 = 84 : A13 = 46 : A14 = 18 : A15 = 55    |
               A18 = 51 : A19 = 91 : A20 =                   |
150  A71 = 31 : A77 = 69 :                          = 30     |
160  A76 = 40 : A77 = 69 : A78 = 51 : A19 = 91 : A20 = 30    } Assigns values to variables
170  A81 = 91 : A82 = 46 : A83 = 23 : A84 = 37 : A85 = 84    |
180  A86 = 65 : A87 = 23 : A88 = 98 : A89 = 51 : A90 = 30    |
190  A91 = 57 : A92 = 78 : A93 = 16 : A94 = 39 : A95 = 46    |
200  A96 = 59 : A97 = 24 : A98 = 32 : A99 = 74 : A100 = 47   )
210  X = A1 + A2 + A3 + ........... + A49 + A50     } Calculates sum
220  X = X + A51 + A52 + .......... + A99 + A100    }
230  Y = A1 ^ 2 + A2 ^ 2 + ......... + A39 ^ 2 + A40 ^ 2          )
240  Y = Y + A41 ^ 2 + A42 ^ 2 + .......... + A79 ^ 2 + A80 ^ 2   } Calculates sum of squares
250  Y = Y + A81 ^ 2 + A82 ^ 2 + .......... + A99 ^ 2 + A100 ^ 2  )
```

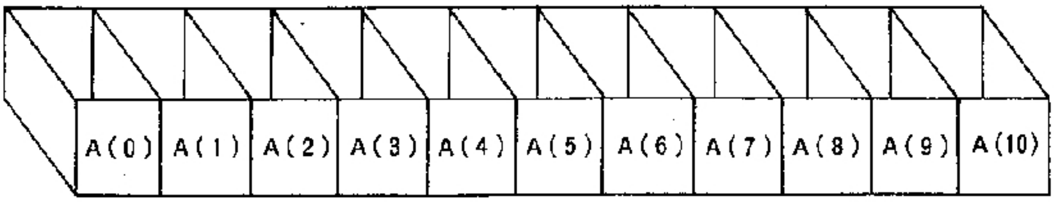
Array

```
 10  DIM A (100) }       Declares array
 20  FOR I = 1 TO 100 : READ A (I) : NEXT I }     Assigns values to array
 30  X = 0 : Y = 0
 40  FOR I = 1 TO 100                  }   Calculates sum and sum of squares
 50  X = X + A (I) : Y = Y + A (I)^ 2  }
 60  NEXT I
 70  DATA 61, 38, 90, 37, 99  )
 80  DATA 12, 17, 94, 39, 75  |
 90  DATA 24, 84, 46, 18, 55  |
100  DATA 46, 65, 51, 91, 30  |
              26, 11, 88, 78  } Data
220  DATA 47,                 |
230  DATA 91, 46, 28, 31, 64  |
240  DATA 65, 23, 98, 51, 30  |
250  DATA 57, 78, 16, 39, 46  |
260  DATA 59, 24, 32, 74, 47  )
```

A look at these programs reveals that an increase in data entails virtually no change in the portion which calculates the sum and sum of squares. The only changes would be in lines 10, 20, and 40, where the constant would be changed from 10 to 100.

Again, the concept of the array can be better grasped by thinking of them as boxes. Previously, a simple variable was described as a single box. Arrays, on the other hand, would be a series of numbered boxes which form a set.

EXAMPLE:
Array A (10)



As illustrated above, the array A(10) actually contains a total of eleven boxes, numbered from A(0) through A(10), with each box being capable of holding a different value. The actual term used to refer to a box is "element". Recalling a stored value is performed by simply specifying the corresponding element number.

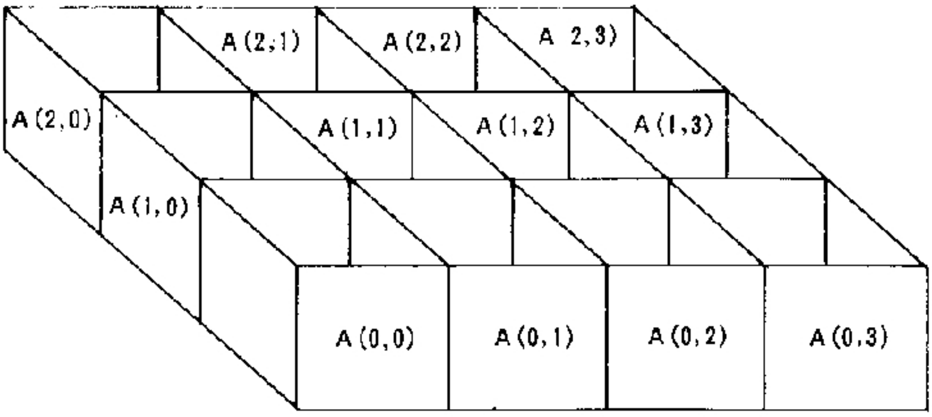
EXAMPLE:
Recall value stored in element 4 of array A

Y=(4)
or
X=4 : Y=A (X)

The value which specifies an element in an array (4 above) is called a subscript.

Until now, the only arrays covered have been those formed by a single line of elements or "boxes". These are known as "one-dimensional" arrays. Arrays may also contain more than one dimension with elements connected vertically and horizontally into two-dimensional and three-dimensional arrays.

EXAMPLE:
DIM A (2, 3)



The declaration in this example sets up an array of three lines and four columns, making it capable of storing 12 different values.

Numeric arrays and string arrays
As with simple variables, arrays can also be declared to hold strings by using the "$" symbol following the array variable name. Again remember, numeric values cannot be assigned to string arrays and strings cannot be assigned to numeric arrays.

EXAMPLE:

The following procedure is used to declare an array and store the data for five individuals and their points scored during a certain game.

String array N$(5) declared for names
Numeric array P(5) declared for points

```
10  DIM N$ (5), P (5)......................Declaration of arrays to store names and points
20  FOR I=1 TO 5
30  READ A$, X
40  N$ ( I )=A$..............................Stores names to string array
50  P ( I )=X..................................Stores points to numeric array
60  NEXT I
70  END
80  DATA SMITH, 70, BROWN, 68, JONES, 87, CARTER, 80, MILLS, 74
```

6-7-3 Summary

Variable Types

The three following types of variables are available for use with this unit.

1. Numeric variables (up to 12-digit mantissa)		A, a, NUMBER, POINTS
2. String variables (up to 255 characters)		A$, STRING$
3. Array variables	Numeric array	A (10), XX (3, 3, 3)
	String array	A$ (10), ARRAY$ (2, 2)

Variable Names

- Variabel names can consist of upper, lower case or numeric characters, but a numeric character cannot be used in the first position of the variable name (i.e. 1AE, 3BC$ are illegal).
- Reserved words (see page 400) cannot be used as the leading characters of a variable name (i.e. RUNON, LIST1$ are illegal).
- The maximum length of a variable name is 15 characters.

Arrays

1. Arrays are declared by DIM statements.
2. Elements described by subscripts which are integers greater than 0. Fractions are disregarded.
3. The number of dimensions is limited by stack capacity.
4. The maximum value of subscripts is limited by memory capacity.

Variable/Array Application

1. Variables and arrays can be used jointly by all program areas.
2. Arrays cannot be used unless first declared using the DIM statement.

Counting Bytes Used by Variables

The following outlines the number of bytes reserved when a variable appears the first time within a program.

- **Numeric Variables**

(variable name length + 12) bytes in variable area

- **String Variables**

(variable name length + 4) bytes in variable area and (string length + 1) bytes in string area
Areas are reserved for array variables when the array is declared by the DIM statement.

- **Numeric Array Variables**

(variable name length + 4) + (array size × 8) + (dimension × 2 + 1) bytes in variable area

EXAMPLE:

```
DIM XYZ (3, 3, 5, 2)
```

Name : 3
Size : 4 × 4 × 6 × 3 = 288
Dimension : 4
Calculation : (3 + 4) + (288 × 8) + 4 × 2 + 1 = 2320 bytes

- **String Array Variables**

(variable name length + 4) + (array size) + (dimension × 2) bytes in variable area.
The lengths of individual strings are required in the variable area when strings are assigned to the array.

EXAMPLE:

```
10  DIM AB$(3, 3)
20  AB$ (0, 0)= "*****"
```

Name : 2
Size : 4 × 4 = 16
Dimension : 2
Calculations : (2 + 4) + 16 + (2 × 2) + 5 bytes

Calculating Program Length

The following shows points which must be considered when calculating memory requirements for programs.

Line numbers : 2 bytes per line number, regardless of number length (1 ~ 65535)
Commands : 2 bytes per command
Functions : 2 bytes per function
Numeric/alphabetic characters : 1 byte per character (spaces also counted as characters)
EXE key : 1 byte per EXE key operation at end of program line (for storage of line)
1 byte added to sum of the above

EXAMPLE:

```
10  A = SIN X
```

2 (line number) + 1 (space following line number) + 1 (A) + 1 (=) + 2 (SIN) + 1 (space) + 1 (X) + 1 (EXE) + 1 = 11

This calculation indicates that a total of 11 bytes are required for storage of the above program.
* The space following the line number is added automatically.

6-8 PROGRAM SAVE AND LOAD

The following save and load procedures can only be performed when the FA-6 interface unit is used.

6-8-1 Program Save

Programs stored in the memory of the unit are protected by the memory back up battery even when the power of the unit is switched OFF. The entire contents of the memory, however, are deleted whenever both the main power supply batteries and memory back up batteries are removed from the unit at the same time, or when the NEW ALL command is executed. Program area contents can be stored onto standard cassette tapes to protect against loss of important data, or to make room for further programming when all program areas are full. The following two commands are available for such save operations.

SAVE : Saves contents of current program area.
SAVE ALL : Saves entire contents of all program areas.

EXAMPLE:
Executing SAVE in this case saves the contents of program area P0, while SAVE ALL would save the contents of program areas P0 through P9.

SAVE EXE

```
SAVE
```

(Saves program in program area 0)

```
SAVE
Ready P0
```

(Save complete)

Filenames up to eight characters long can also be assigned to programs stored on cassette tapes using the SAVE and SAVE ALL commands.

SAVE "BASIC" EXE

```
SAVE 'BASIC'
```

(Saves program under filename "BASIC")

```
SAVE"BASIC"
Ready P0
```

(Save complete)

6-8-2 Program Verify

The VERIFY command makes it possible to verify whether or not the program saved using SAVE or SAVE ALL was copied correctly to the cassette tape.

EXAMPLE:
Verify correct save of the program BASIC

VERIFY "BASIC" EXE

```
VERIFY"BASIC"
```

(Verification of saved program)

```
BASIC      B
```

(Finds specified program and verifies)

```
BASIC      B
Ready P0
```

(Verification complete)

If the Ready prompt does not appear after some time, check whether or not the filename entered with the VERIFY command is correct. If it is correct, adjust the volume level of the cassette recorder being used and repeat the verification procedure.

```
P0 error     o
Ready P0
```

The error message illustrated above indicates that the program was not saved correctly. In this case, check the following items:

- Verify the program again, this time appending "CAS1:" before the filename (VERIFY "CAS1 : BASIC" in the above example).
- Ensure that connections between the computer and cassette tape recorder are correct and secure.
- Ensure that the volume level of the recorder is set to in the vicinity of its maximum.
- Check whether the cassette tape is damaged.
- Check whether the recorder heads are soiled.

Note also that an error will be generated if a program exists on the tape with the same name as that currently present in computer memory, but the contents of the two programs are different.

* The VERIFY command automatically determines whether the program being checked was saved using the SAVE or SAVE ALL command.

6-8-3 Program Load

Programs stored on cassette tapes using the SAVE and SAVE ALL commands can be loaded into the computer using the LOAD and LOAD ALL commands.

EXAMPLE:
Load the program "BASIC" from cassette tape into memory

LOAD EXE

```
LOAD
```
(Program load command)

```
BASIC       B
```
(Program filename)

```
Ready P0
```
(Load complete)

Note that executing the LOAD and LOAD ALL commands while programs are already stored in memory deletes the current memory contents.

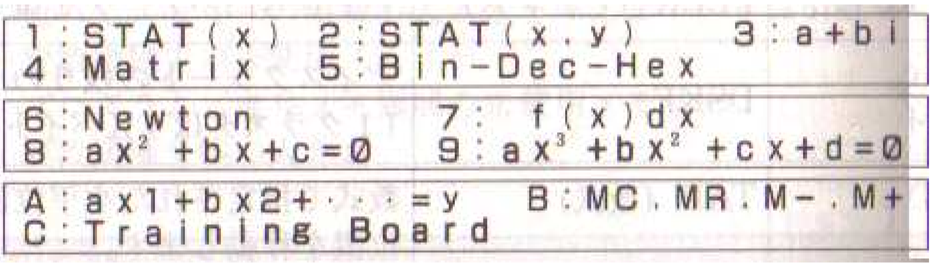
The LOAD ALL command can be used to load programs to all of the program areas (P0 ~ P9). Specifying a filename in the LOAD and LOAD ALL commands causes the unit to search for the specified filename for loading into memory. The following table shows the relationship between the LOAD, LOAD ALL, SAVE and SAVE ALL commands.

	LOAD	LOAD "filename"	LOAD ALL	LOAD ALL "filename"
SAVE	○	×	×	×
SAVE "filename"	○	○	×	×
SAVE ALL	×	×	○	×
SAVE ALL "filename"	×	×	○	○

NOTE:
See PART 7 PERIPHERAL DEVICES for details on using the SAVE and LOAD commands.

PART 11 SCIENTIFIC LIBRARY ....

This chapter deals with the  key providing the following functions :



```
1:STAT(x)  2:STAT(x.y)    3:a+bi
4:Matrix   5:Bin-Dec-Hex

6:Newton          7:  f(x)dx
8:ax² +bx+c=0     9:ax³ +bx² +cx+d=0

A:ax1+bx2+···=y   B:MC.MR.M-.M+
C:Training Board
```

1 Single statistics

2 Multiple statistics

3 Complex numbers

4 Matrix operations

5 Binary – Decimal - hexadecimal

6 Numeric solutions of an equation : The Newton method

7 ???

8 Quadratic equations

9 Cubic equations

A Simultaneous equations: Gauss-Jordan elimination

B Memory calculations

1 SINGLE VARIABLE STATISTICS

Determines the following statistics and determines the deviation value for input of n data items.

Number of data items	CNT : n	
Sum of data	SUMX : Σx	
Sum of squares of data	SUMX2 : Σx^2	
Mean of data	MEANX : $\Sigma x/n$	
Population standard deviation of data	SDXN : $x\sigma n$	$\sqrt{\dfrac{n\Sigma x^2-(\Sigma x)^2}{n^2}}$
Sample standard deviation of data	SDX : $x\sigma n_{-1}$	$\sqrt{\dfrac{n\Sigma x^2-(\Sigma x)^2}{n(n-1)}}$

OPERATION

Fx 1

```
Statistics  [x]
>In,Del,Clear,List,T-score,P ?_
```

The menu illustrated above is displayed for single variable statistical calculations. The following six items can be selected from this menu:

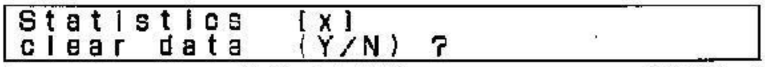
1. I : Data input (does not clear data already present in memory)
2. D: Data deletion (deletes erroneous or unnecessary data)
3. C: Data clear
4. L: Statistic display
   Displays number of data items, sum of data, sum of squares of data, mean of data, population standard deviation of data, and sample standard deviation of data in sequence. ⇩ (or EXE) scrolls to the following data item, ⇧ to the previous data item, and ⇦ or ⇨ terminate statistic display.
5. T: Calculates deviation value of obtained value.
6. P: Outputs all statistics to printer

EXAMPLE

Enter the following five test scores and display statistics. Also determine the deviation value for the score of 88.

Data : 98, 88, 62, 90, 78

C



```
Statistics  [x]
clear data  (Y/N) ?
```

(Data clear)

Y

```
Statistics  [x]
>In,Del,Clear,List,T-score,P ?_
```

(Data clear confirmation)

I

```
Input data (x)          [EXE]:menu
x?_
```

(Data input)

98 EXE 88 EXE 62 EXE 90 EXE 78 EXE

```
Input data (x)          [EXE]:menu
x?_
```

(Input of each score)

EXE

```
Statistics   [x]
>In.Del.Clear.List.T-score.P ?_
```

(Return to menu display)

L

```
CNT     :n     = 5
SUMX    :Σx    = 416
```

(Statistic display showing number of data and sum)

EXE

```
SUMX    :Σx    = 416
SUMX2   :Σx2   = 35376
```

(Sum of squares)

EXE

```
SUMX2   :Σx2   = 35376
MEANX   :Σx/n  = 83.2
```

(Mean)

EXE

```
MEANX   :Σx/n  = 83.2
SDXN    :xσn   = 12.36769987
```

(Population standard deviation)

EXE

```
SDXN    :xσn   = 12.36769987
SDX     :xσn-1 = 13.82750881
```

(Sample standard deviation)

EXE

```
Statistics   [x]
>In.Del.Clear.List.T-score.P ?_
```

(Return to menu display)

T

```
Statistics   [x]
x?_
```

(Deviation value)

88 EXE

```
Statistics   [x]
x?88            :T= 53.9
```

(Data for calculation of deviation value to be displayed)

EXE

```
Statistics   [x]
x?_
```

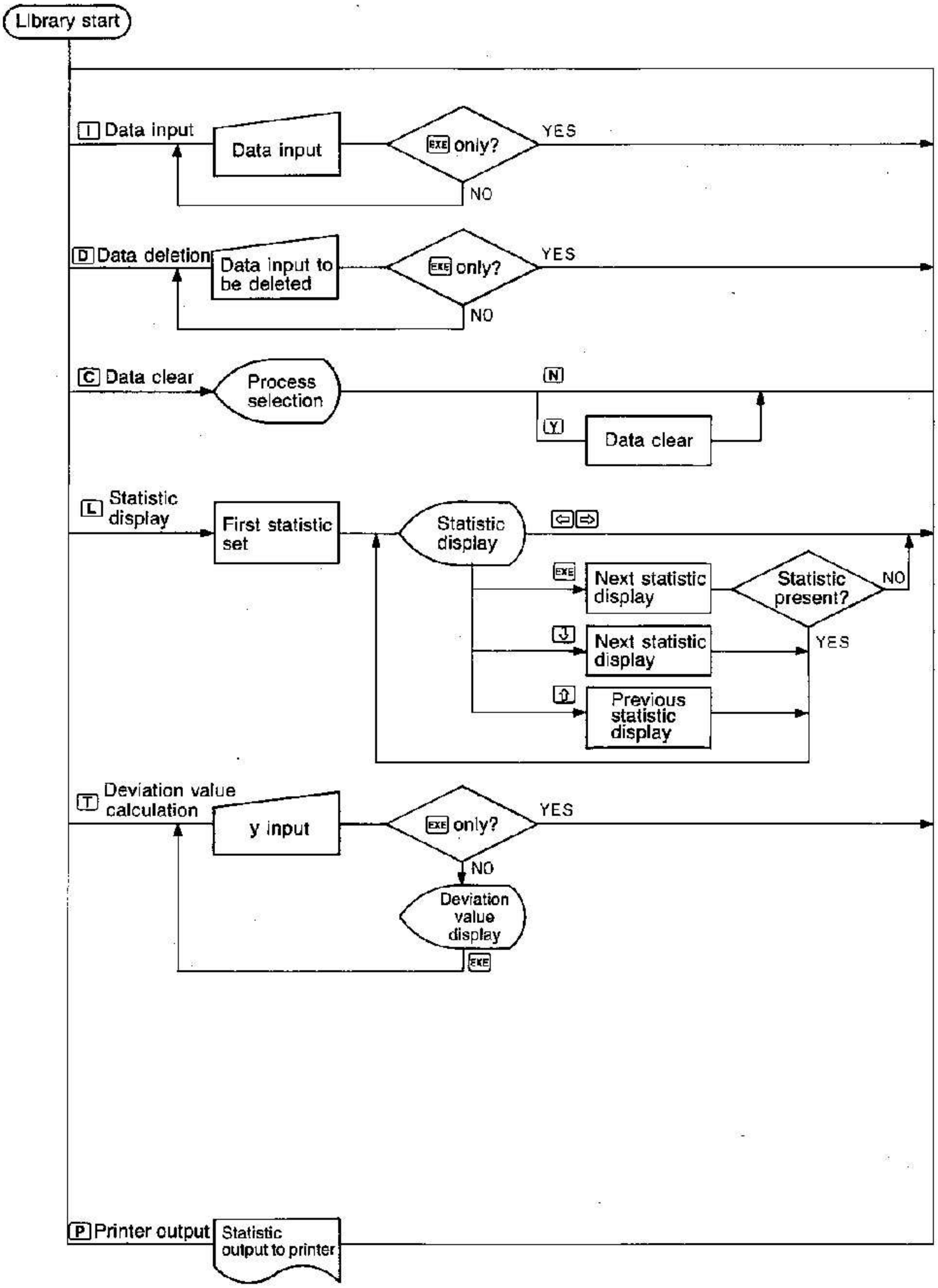
EXE

```
Statistics   [x]
>In.Del.Clear.List.T-score.P ?_
```

(Return to menu display)

Here, the deviation value of the 88 score is 53.9.

SINGLE VARIABLE STATISTICS FLOWCHART



2.1

LINEAR REGRESSION ANALYSIS

(y = a + bx)

Performs linear regression analysis on n data groups (x, y) and calculates the statistics listed below. Also determines the following on the regression line:

- Estimated value of x in relation to y (EOX)
- Estimated value of y in relation to x (EOY)

STATISTIC TABLE

Number of data items	CNT : n	
Sum of x data	SUMX : Σx	
Sum of y data	SUMY : Σy	
Sum of squares of x data	SUMX2 : Σx^2	
Sum of squares of y data	SUMY2 : Σy^2	
Sum of products of x and y data	SUMXY : Σxy	
Mean of x data	MEANX : $\Sigma x/n$	
Mean of y data	MEANY : $\Sigma y/n$	
Population standard deviation of x data	SDXN : $x\sigma n$	$\sqrt{\frac{n\Sigma x^2 - (\Sigma x)^2}{n^2}}$
Population standard deviation of y data	SDYN : $y\sigma n$	$\sqrt{\frac{n\Sigma y^2 - (\Sigma y)^2}{n^2}}$
Sample standard deviation of x data	SDX : $x\sigma n_{-1}$	$\sqrt{\frac{n\Sigma x^2 - (\Sigma x)^2}{n(n-1)}}$
Sample standard deviation of y data	SDY : $y\sigma n_{-1}$	$\sqrt{\frac{n\Sigma y^2 - (\Sigma y)^2}{n(n-1)}}$
Linear regression constant term	LRA : a	$\frac{\Sigma y - b\cdot\Sigma x}{n}$
Linear regression coefficient	LRB : b	$\frac{n\Sigma xy - \Sigma x\cdot\Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$
Correlation coefficient	COR : r	$\frac{n\Sigma xy - \Sigma x\cdot\Sigma y}{\sqrt{(n\Sigma x^2 - (\Sigma x)^2(n\Sigma y^2 - (\Sigma y)^2)}}$

OPERATION

Fx 2 1

```
Regression analysis     [y=a+bx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

The menu illustrated above is displayed for linear regression calculations. The following seven items can be selected from this menu:

1. I : Data input
2. D: Data deletion (deletes erroneous or unnecessary data)
3. C: Data clear

4. **L** : Statistic display
   Displays number of data items, sum of x data, sum of y data, sum of squares of x data, sum of squares of y data, sum of products of x and y data, mean of x data, mean of y data, population standard deviation of x data, population standard deviation of y data, sample standard deviation of x data, sample standard deviation of y data, linear regression constant term, linear regression coefficient, and correlation coefficient. ⇩ (or EXE) scrolls to the following data item, ⇧ to the previous data item, and ⇦ or ⇨ terminates statistic display.
5. **X** : Calculates x value for y on regression line.
6. **Y** : Calculates y value for x on regression line
7. **P** : Outputs all statistics to printer.

EXAMPLE

Enter the following five sets of height/weight, and display statistics. Also estimate the weight for a person whose height is 170cm.

	1	2	3	4	5
Height (x)	160	158	175	163	172
Weight (y)	43	45	60	46	58

C

```
Regression analysis     [y=a+bx]
clear data   (Y/N) ?
```

(Data clear)

Y

```
Regression analysis     [y=a+bx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Data clear confirmation)

I

```
Input data (x,y)      [EXE]:menu
x?_                 :y?
```

(Data input)

160 EXE

```
Input data (x,y)      [EXE]:menu
x?160               :y?_
```

(x input)

43 EXE

```
Input data (x,y)      [EXE]:menu
x?_                 :y?
```

(y input)

158 EXE 45 EXE 175 EXE 60 EXE 163 EXE 46 EXE 172 EXE 58 EXE

```
Input data (x,y)      [EXE]:menu
x?_                 :y?
```

(Remaining x, y data input)

EXE

```
Regression analysis     [y=a+bx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Return to menu display)

L

```
CNT      :n     = 5
SUMX     :Σx    = 828
```

(Statistic display showing number of data and sum of x data)

EXE

```
SUMX     :Σx    = 828
SUMY     :Σy    = 252
```

(Sum of y data)

EXE

```
SUMY     :Σy    = 252
SUMX2    :Σx²   = 137342
```

(Sum of squares of x data)

EXE

```
SUMX2    :Σx²   = 137342
SUMY2    :Σy²   = 12954
```

(Sum of squares of y data)

EXE

```
SUMY2    :Σy²   = 12954
SUMXY    :Σxy   = 41964
```

(Sum of products of x and y data)

EXE

```
SUMXY    :Σxy   = 41964
MEANX    :Σx/n  = 165.6
```

(Mean of x data)

EXE

```
MEANX    :Σx/n  = 165.6
MEANX    :Σy/n  = 50.4
```

(Mean of y data)

EXE

```
MEANY    :Σy/n  = 50.4
SDXN     :xσn   = 6.711184694
```

(Population standard deviation of x data)

EXE

```
SDXN     :xσn    = 6.711184694
SDYN     :yσn    = 7.11617875
```

(Population standard deviation of y data)

EXE

```
SDYN     :yσn    = 7.11617875
SDX      :xσn-1  = 7.503332593
```

(Sample standard deviation of x data)

EXE

```
SDX      :xσn-1  = 7.503332593
SDY      :yσn-1  = 7.956129712
```

(Sample standard deviation of y data)

EXE

```
SDY      :yσn-1  = 7.956129712
LRA      :a      =-120.7886323
```

(Linear regression constant term)

EXE

```
LRA      :a      =-120.7886323
LRB      :b      = 1.03374778
```

(Linear regression coefficient)

EXE

```
LRB      :b      = 1.03374778
COR      :r      = 0.9749154035
```

(Correlation coefficient)

EXE

```
Regression analysis     [y=a+bx]
>In,Del,Clear,List,eoX,eoY,P  ?_
```

(Return to menu display)

Y

```
Estimation of y [y=a+bx]
x?_
```

(Estimation of weight)

170 EXE

```
Estimation of y [y=a+bx]
x?170          :ŷ= 54.94849023
```

(Estimated value for weight following input of height)

EXE

```
Estimation of y [y=a+bx]
x?_
```

EXE

```
Regression analysis     [y=a+bx]
>In,Del,Clear,List,eoX,eoY,P  ?_
```

(Return to menu display)

Here, these data produce the line y = −120.7886323 + 1.03374778x. Also, input of a height of 170cm results in an estimated weight of 54.9kg.

2.2 LOGARITHMIC REGRESSION ANALYSIS

$(y = a + b\ln x)$

Performs logarithmic regression analysis on n data groups (x, y) and calculates the statistics listed below. Also determines the following on the logarithmic curve:

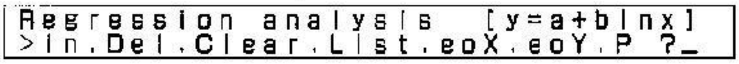
- Estimated value of x in relation to y (EOX)
- Estimated value of y in relation to x (EOY)

STATISTIC TABLE

Number of data items	CNT : n	
Sum of x data logarithmic values	SUMLNX : Σlnx	$\Sigma(lnx)$
Sum of y data	SUMY : Σy	
Sum of squares of x data logarithmic values	SUMLNX2 : Σlnx^2	$\Sigma(lnx)^2$
Sum of squares of y data	SUMY2 : Σy^2	
Sum of products of x data logarithmic values and of y data	SUMLNXY : $\Sigma lnxy$	$\Sigma\{(lnx)\cdot y\}$
Mean of x data logarithmic values	MEANLNX : $\Sigma lnx/n$	$\Sigma(lnx)/n$
Mean of y data	MEANY : $\Sigma y/n$	
Population standard deviation of x data logarithmic values	SDLNXN : $lnx\sigma n$	$\sqrt{\frac{n\Sigma(lnx)^2-(\Sigma lnx)^2}{n^2}}$
Population standard deviation of y data	SDYN : $y\sigma n$	$\sqrt{\frac{n\Sigma y^2-(\Sigma y)^2}{n^2}}$
Sample standard deviation of x data logarithmic values	SDLNX : $lnx\sigma n$ 1	$\sqrt{\frac{n\Sigma(lnx)^2-(\Sigma lnx)^2}{n(n-1)}}$
Sample standard deviation of y data	SDY : $y\sigma n_{-1}$	$\sqrt{\frac{n\Sigma y^2-(\Sigma y)^2}{n(n-1)}}$
Regression constant term	RA : a	$\frac{\Sigma y-b\cdot\Sigma lnx}{n}$
Regression coefficient	RB : b	$\frac{n\Sigma(lnx)y-\Sigma lnx\cdot\Sigma y}{n\Sigma(lnx)^2-(\Sigma lnx)^2}$
Correlation coefficient	COR : r	$\frac{n\Sigma(lnx)y-\Sigma lnx\cdot\Sigma y}{\sqrt{(n\Sigma(lnx)^2-(\Sigma lnx)^2)(n\Sigma y^2-(\Sigma y)^2)}}$

OPERATION

Fx 2 2



```
Regression analysis  [y=a+blnx]
>In.Del,Clear,List.eoX,eoY,P ?_
```

The menu illustrated above is displayed for logarithmic regression calculations. The following seven items can be selected from this menu:

1. I : Data input
2. D: Data deletion (deletes erroneous or unnecessary data)
3. C: Data clear

4. **L**: Statistic display
   Displays number of data items, sum of x data logarithmic values, sum of y data, sum of squares of x data logarithmic values, sum of squares of y data, sum of products of x data logarithmic values and y data, mean of x data logarithmic values, mean of y data, population standard deviation of x data logarithmic values, population standard deviation of y data, sample standard deviation of x data logarithmic values, sample standard deviation of y data, regression constant term, regression coefficient, and correlation coefficient. ⇩ (or EXE) scrolls to the following data item, ⇧ to the previous data item, and ⇦ or ⇨ terminates statistic display.
5. **X**: Calculates x value for y on logarithm curve.
6. **Y**: Calculates y value for x on logarithm curve.
7. **P**: Outputs all statistics to printer.

EXAMPLE

Enter the following measured data for microbes, perform logarithmic regression, and display the statistics. Also estimate the number of microbes with a temperature of 18 degrees using the logarithm curve obtained.

	1	2	3	4	5
Temperature (x)	5°	12°	20°	27°	36°
Microbes (y)	680	1100	1300	1440	1600

C

```
Regression analysis  [y=a+blnx]
clear data  (Y/N) ?
```

(Data clear)

Y

```
Regression analysis  [y=a+blnx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Data clear confirmation)

I

```
Input data (x,y)      [EXE]:menu
x?_              :y?
```

(Data input)

5 EXE

```
Input data (x,y)      [EXE]:menu
x?5              :y?_
```

(x input)

680 EXE

```
Input data (x,y)      [EXE]:menu
x?_              :y?
```

(y input)

12 EXE **1100** EXE **20** EXE **1300** EXE **27** EXE **1440** EXE **36** EXE **1600** EXE

```
Input data (x,y)      [EXE]:menu
x?_              :y?
```

(Remaining x, y data input)

EXE

```
Regression analysis  [y=a+blnx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Return to menu display)

L

```
CNT     :n     =  5
SUMlnx  :Σlnx  =  13.96943264
```

(Statistic display showing number of data and sum of x data logarithmic values)

EXE

```
SUMlnx  :Σlnx  =  13.96943264
SUMY    :Σy    =  6120
```

(Sum of y data)

EXE

```
SUMY    :Σy    =  6120
SUMlnX2:Σlnx²=  41.44361194
```

(Sum of squares of x data logarithmic values)

EXE

```
SUMlnX2:Σlnx²=  41.44361194
SUMY2   :Σy²   =  7996000
```

(Sum of squares of y data)

EXE

```
SUMY2   :Σy²   =  7996000
SUMlnXY:Σlnxy=  18201.90244
```

(Sum of products of x data logarithmic values and y data)

EXE

```
SUMlnXY:Σlnxy  =  18201.90244
MEANlnX:Σlnx/n=  2.793886528
```

(Mean of x data logarithmic values)

EXE

```
MEANlnX:Σlnx/n=  2.793886528
MEANY  :Σy/n   =  1224
```

(Mean of y data)

EXE

```
MEANY  :Σy/n    = 1224
SDlnXN :lnxσn   = 0.6949247842
```

(Population standard deviation of x data logarithmic values)

EXE

```
SDlnXN :lnxσn   = 0.6949247842
SDYN   :yσn     = 317.8427284
```

(Population standard deviation of y data)

EXE

```
SDYN   :yσn     = 317.8427284
SDlnX  :lnxσn-1= 0.7769495284
```

(Sample standard deviation of x data logarithmic values)

EXE

```
SDlnX  :lnxσn-1= 0.7769485284
SDY    :yσn-1   = 355.3589734
```

(Sample standard deviation of y data)

EXE

```
SDY    :yσn-1   = 355.3588734
RA     :a       =-52.62523046
```

(Regression constant term)

EXE

```
RA     :a       =-52.62523046
RB     :b       = 456.935247
```

(Regression coefficient)

EXE

```
RB     :b       = 456.935247
COR    :r       = 0.9990337973
```

(Correlation coefficient)

EXE

```
Regression analysis  [y=a+blnx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Return to menu display)

Y

```
Estimation of y [y=a+blnx]
x?_
```

(Estimation of y)

18 EXE

```
Estimation of y [y=a+blnx]
x?18           :ŷ= 1268.087503
```

(Estimated value for y following input of 18 degrees)

EXE

```
Estimation of y [y=a+blnx]
x?_
```

EXE

```
Regression analysis  [y=a+blnx]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Return to menu display)

Here, these data produce the curve $y = -52.62523046 + 456.935247 \cdot \ln x$. Also, input of a temperature of 18 degrees results in an estimated total of 1,268 microbes.

2.3 EXPONENTIAL REGRESSION ANALYSIS

(y = ab^x)

Performs exponential regression analysis on n data groups (x, y) and calculates the statistics listed below. Also determines the following on the exponential curve:

- Estimated value of x in relation to y (EOX)
- Estimated value of y in relation to x (EOY)

STATISTIC TABLE

Number of data items	CNT : n	
Sum of x data	SUMX : Σx	
Sum of y data logarithmic values	SUMLNY : Σlny	$\Sigma(lny)$
Sum of squares of x data	SUMX2 : Σx^2	
Sum of squares of y data logarithmic values	SUMLNY2 : Σlny^2	$\Sigma(lny)^2$
Sum of products of x data and y data logarithmic values	SUMXLNY : $\Sigma xlny$	
Mean of x data	MEANX : $\Sigma x/n$	
Mean of y data logarithmic values	MEANLNY : $\Sigma lny/n$	
Population standard deviation of x data	SDXN : $x\sigma n$	$\sqrt{\frac{n\Sigma x^2-(\Sigma x)^2}{n^2}}$
Population standard deviation of y data logarithmic values	SDLNYN : $lny\sigma n$	$\sqrt{\frac{n\Sigma(lny)^2-(\Sigma lny)^2}{n^2}}$
Sample standard deviation of x data	SDX : $x\sigma n$ 1	$\sqrt{\frac{n\Sigma x^2-(\Sigma x)^2}{n(n-1)}}$
Sample standard deviation of y data logarithmic values	SDLNY : $lny\sigma n_{-1}$	$\sqrt{\frac{n\Sigma(lny)^2-(\Sigma lny)^2}{n(n-1)}}$
Regression constant term	RA : a	$\mathrm{EXP}\left(\frac{\Sigma(lny)-b\cdot\Sigma x}{n}\right)$
Regression coefficient	RB : b	$\mathrm{EXP}\left(\frac{n\Sigma xlny-\Sigma x\cdot\Sigma lny}{n\Sigma x^2-(\Sigma x)^2}\right)$
Correlation coefficient	COR : r	$\frac{n\Sigma xlny-\Sigma x\cdot\Sigma lny}{\sqrt{(n\Sigma x^2-(\Sigma x)^2)(n\Sigma(lny)^2-(\Sigma lny)^2}}$

OPERATION

Fx 2 3

```
Regression analysis   [y=ab^x]
>In.Del.Clear.List.eoX.eoY.P ?_
```

The menu illustrated above is displayed for exponential regression calculations. The following seven items can be selected from this menu:

1. I : Data input
2. D: Data deletion (deletes erroneous or unnecessary data)
3. C: Data clear

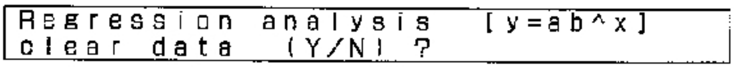
4. L : Statistic display
   Displays number of data items, sum of x data, sum of y data logarithmic values, sum of squares of x data, sum of squares of y data logarithmic values, sum of products of x data and y data logarithmic values, mean of x data, mean of y data logarithmic values, population standard deviation of x data, population standard deviation of y data logarithmic values, sample standard deviation of x data, sample standard deviation of y data logarithmic values, regression constant term, regression coefficient, and correlation coefficient. ⇩ (or EXE) scrolls to the following data item, ⇧ to the previous data item, and ⇦ or ⇨ terminate statistic display.
5. X : Calculates x value for y on regression line.
6. Y : Calculates y value for x on regression line.
7. P : Outputs all statistics to printer.

EXAMPLE

Enter the following data for the amount of sales per customer and number of customers for a store, perform exponential regression, and display the statistics. Also estimate the amount of sales per customer for 150 customers using the exponential curve obtained.

	1	2	3	4	5
Customers (x)	115	124	130	138	142
Sales/customer (y) ($)	40	41.6	43.0	46.0	46.5

C



```
Regression analysis   [y=ab^x]
clear data   (Y/N) ?
```

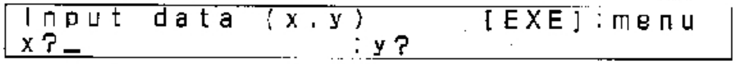
(Data clear)

Y

```
Regression analysis   [y=ab^x]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Data clear confirmation)

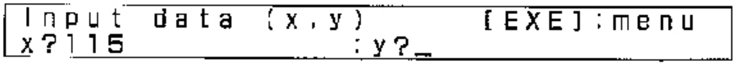
I



```
Input data (x,y)        [EXE]:menu
x?_               :y?
```

(Data input)

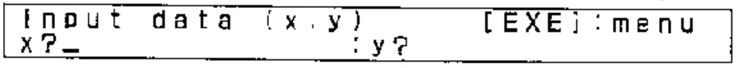
115 EXE



```
Input data (x,y)        [EXE]:menu
x?115             :y?_
```

(x input)

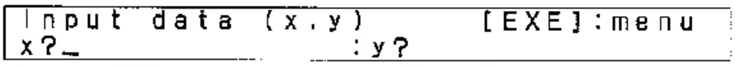
40 EXE



```
Input data (x,y)        [EXE]:menu
x?_               :y?
```

(y input)

124 EXE 41.6 EXE 130 EXE 43 EXE 138 EXE 46 EXE 142 EXE 46.5 EXE



```
Input data (x,y)        [EXE]:menu
x?_               :y?
```

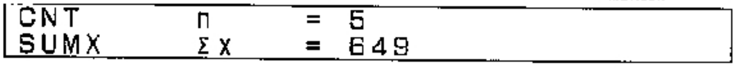
(Remaining x, y data input)

EXE

```
Regression analysis   [y=ab^x]
>In.Del.Clear.List.eoX.eoY.P ?_
```

(Return to menu display)

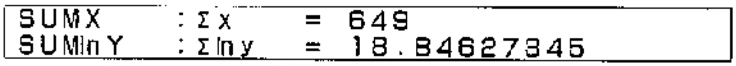
L



```
CNT       n     = 5
SUMX      Σx    = 649
```

(Statistic display showing number of data and sum of x data)

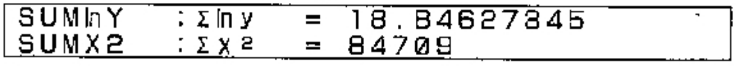
EXE



```
SUMX     :Σx      = 649
SUMlnY   :Σlny    = 18.84627345
```

(Sum of y data logarithmic values)

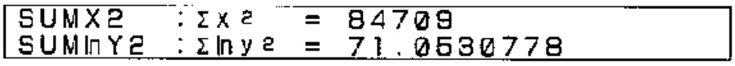
EXE



```
SUMlnY   :Σlny    = 18.84627345
SUMX2    :Σx2     = 84709
```

(Sum of squares of x data)

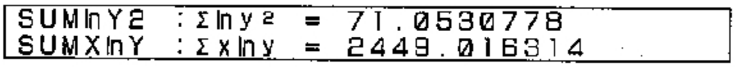
EXE



```
SUMX2    :Σx2     = 84709
SUMlnY2  :Σlny2   = 71.0530778
```

(Sum of squares of y data logarithmic values)

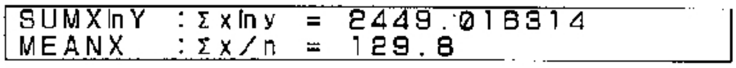
EXE



```
SUMlnY2  :Σlny2   = 71.0530778
SUMXlnY  :Σxlny   = 2449.016314
```

(Sum of products of x data and y data logarithmic values)

EXE



```
SUMXlnY  :Σxlny   = 2449.016314
MEANX    :Σx/n    = 129.8
```

(Mean of x data)

EXE
```
MEANX    :Σx/n  =  129.8
MEANlnY  :Σlny/n=  3.769254689
```
(Mean of y data logarithmic values)

EXE
```
MEANlnY  :Σlny/n=  3.769254689
SDXN     :xσn   =  9.68297475
```
(Population standard deviation of x data)

EXE
```
SDXN     :xσn   =  9.68297475
SDlnYN   :lnyσn =  5.774640647E-02
```
(Population standard deviation of y data logarithmic values)

EXE
```
SDlnYN   :lnyσn =  5.774640647E-02
SDX      :xσn-1 =  10.82589488
```
(Sample standard deviation of x data)

EXE
```
SDX      :xσn-1 =  10.82589488
SDlnY    :lnyσn-1= 6.456244516E-02
```
(Sample standard deviation of y data logarithmic values)

EXE
```
SDlnY    :lnyσn-1= 6.456244516E-02
RA       :a     =  20.1317721
```
(Regression constant term)

EXE
```
RA       :a     =  20.1317721
RB       :b     =  1.005926239
```
(Regression coefficient)

EXE
```
RB       :b     =  1.005926239
COR      :r     =  0.9907846423
```
(Correlation coefficient)

EXE
```
Regression analysis  [y=ab^x]
>In.Del.Clear.List.eoX.eoY.P ?_
```
(Return to menu display)

Y
```
Estimation of y [y=ab^x]
x?_
```
(Estimation of y)

150 EXE
```
Estimation of y [y=ab^x]
x?150         :ŷ= 48.84301552
```
(Estimated value for y following input of 150 customers)

EXE
```
Estimation of y [y=ab^x]
x?_
```

EXE
```
Regression analysis  [y=ab^x]
 In.Del.Clear.List.eoX.eoY.P ?_
```
(Return to menu display)

Here, these data produce the curve $y = 20.1317721 \times 1.005926239^x$. Also, input of a total of 150 customers results in an estimated amount per customer of $48.843.

2.4 POWER REGRESSION ANALYSIS

(y = ax^b)

Performs power regression analysis on n data groups (x, y) and calculates the statistics listed below. Also determines the following on the power curve:

- Estimated value of x in relation to y (EOX)
- Estimated value of y in relation to x (EOY)

STATISTIC TABLE

Number of data items	CNT	: n	
Sum of x data logarithmic values	SUMLNX	: Σlnx	
Sum of y data logarithmic values	SUMLNY	: Σlny	
Sum of squares of x data logarithmic values	SUMLNX 2	: Σlnx^2	$\Sigma(lnx)^2$
Sum of squares of y data logarithmic values	SUMLNY 2	: Σlny^2	$\Sigma(lny)^2$
Sum of products of x data logarithmic values and y data logarithmic values	SUMLNXLNY	: $\Sigma lnxlny$	$\Sigma(lnx \cdot lny)$
Mean of x data logarithmic values	MEANLNX	: $\Sigma lnx/n$	
Mean of y data logarithmic values	MEANLNY	: $\Sigma lny/n$	
Population standard deviation of x data logarithmic values	SDLNXN	: $lnx\sigma n$	$\sqrt{\frac{n\Sigma(lnx)^2-(\Sigma lnx)^2}{n^2}}$
Population standard deviation of y data logarithmic values	SDLNYN	: $lny\sigma n$	$\sqrt{\frac{n\Sigma(lny)^2-(\Sigma lny)^2}{n^2}}$
Sample standard deviation of x data logarithmic values	SDLNX	: $lnx\sigma n_{-1}$	$\sqrt{\frac{n\Sigma(lnx)^2-(\Sigma lnx)^2}{n(n-1)}}$
Sample standard deviation of y data logarithmic values	SDLNY	: $lnx\sigma n_{-1}$	$\sqrt{\frac{n\Sigma(lny)^2-(\Sigma lnx)^2}{n(n-1)}}$
Regression constant term	RA	: a	$\frac{\Sigma lny-b\cdot\Sigma lnx}{n}$
Regression coefficient	RB	: b	$\frac{n\Sigma lnx\cdot lny-\Sigma lnx\cdot\Sigma lny}{n\Sigma(lnx)^2-(\Sigma lnx)^2}$
Correlation coefficient	COR	: c	$\frac{n\Sigma lnx\cdot lny-\Sigma lnx\Sigma lny}{\sqrt{(n\Sigma(lnx)^2-(\Sigma lnx)^2)(n\Sigma(lny)^2-(\Sigma lny)^2)}}$

OPERATION

Fx 2 4

```
Regression analysis     [y=ax^b]
>In.Del.Clear.List.eoX.eoY.P ?_
```

The menu illustrated above is displayed for power regression calculations. The following seven items can be selected from this menu:

1. I : Data input
2. D: Data deletion (deletes erroneous or unnecessary data)
3. C: Data clear

4. L : Statistic display
   Displays number of data items, sum of x data logarithmic values, sum of y data logarithmic values, sum of squares of x data logarithmic values, sum of squares of y data logarithmic values, sum of products of x data logarithmic values and y data logarithmic values, mean of x data logarithmic values, mean of y data logarithmic values, population standard deviation of x data logarithmic values, population standard deviation of y data logarithmic values, sample standard of x data logarithmic values, sample standard of y data logarithmic values, regression constant term, regression coefficient, and correlation coefficient.
   ⇩ (or EXE) scrolls to the following data item, ⇧ to the previous data item, and ⇦ or ⇨ terminate statistic display.
5. X : Calculates x value for y on power curve.
6. Y : Calculates y value for x on power curve.
7. P : Outputs all statistics to printer.

EXAMPLE

Enter the following data for the characteristics of voltage and current for a semiconductor, perform power regression, and display the statistics. Also produce an estimated value for current at 40V.

	1	2	3	4	5
Voltage (x)	10	15	20	25	30
Current (y)	13	22	31	38	43

C

```
Regression analysis     [y=ax^b]
clear data   (Y/N) ?
```

(Data clear)

Y

```
Regression analysis     [y=ax^b]
>In,Del,Clear,List,eoX,eoY,P ?_
```

(Data clear confirmation)

I

```
Input data (x,y)      [EXE]:menu
x?_                :y?
```

(Data input)

10 EXE

```
Input data (x,y)      [EXE]:menu
x?10               :y?_
```

(x input)

13 EXE

```
Input data (x,y)      [EXE]:menu
x?_                :y?
```

(y input)

15 EXE 22 EXE 20 EXE 31 EXE 25 EXE 38 EXE 30 EXE 43 EXE

```
Input data (x,y)      [EXE]:menu
x?_                :y?
```

(Remaining x, y data input)

EXE

```
Regression analysis     [y=ax^b]
>In,Del,Clear,List,eoX,eoY,P ?_
```

(Return to menu display)

L

```
CNT      :n     =  5
SUMlnX   :Σlnx  =  14.62644077
```

(Statistic display showing number of data and sum of x data logarithmic values)

EXE

```
SUMlnX   :Σlnx  =  14.62644077
SUMlnY   :Σlny  =  16.48876529
```

(Sum of y data logarithmic values)

EXE

```
SUMlnY   :Σlny  =  16.48876529
SUMlnX2  :Σlnx² =  43.53915106
```

(Sum of squares of x data logarithmic values)

EXE

```
SUMlnX2  :Σlnx² =  43.53915106
SUMlnY2  :Σlny² =  55.30443616
```

(Sum of squares of y data logarithmic values)

EXE

```
SUMlnY2   :Σlny2   =  55.30443616
SUMlnXlnY :Σlnxlny =  49.06554072
```

(Sum of products of x data logarithmic values and y data logarithmic values)

EXE

```
SUMlnXlnY :Σlnxlny =  49.06554072
MEANlnX   :Σlnx/n  =  2.925288155
```

(Mean of x data logarithmic values)

EXE

```
MEANlnX   :Σlnx/n  =  2.925288155
MEANlnY   :Σlny/n  =  3.297753058
```

(Mean of y data logarithmic values)

EXE

```
MEANlnY   :Σlny/n  =  3.297753058
SDlnXN    :lnxσn   =  0.3879683282
```

(Population standard deviation of x data logarithmic values)

EXE

```
SDlnXN    :lnxσn   =  0.3879683282
SDlnYN    :lnyσn   =  0.4309431503
```

(Population standard deviation of y data logarithmic values)

EXE

```
SDlnYN    :lnyσn   =  0.4309431503
SDlnX     :lnxσn-1 =  0.4337617775
```

(Sample standard deviation of x data logarithmic values)

EXE

```
SDlnX     :lnxσn-1 =  0.4337617775
SDlnY     :lnyσn-1 =  0.4818090893
```

(Sample standard deviation of y data logarithmic values)

EXE

```
SDlnY     :lnyσn-1 =  0.4818090893
RA        :a       =  1.069436811
```

(Regression constant term)

EXE

```
RA        :a       =  1.069436811
RB        :b       =  1.104376978
```

(Regression coefficient)

EXE

```
RB        :b       =  1.104376978
COR       :r       =  0.9842455045
```

(Correlation coefficient)

EXE

```
Regression analysis         [y=ax^b]
>ln.Del.Clear.List.eoX.eoY.P  ?_
```

(Return to menu display)

Y

```
Estimation of y  [y=ax^b]
x?_
```

(Estimation of y)

40 EXE

```
Estimation of y  [y=ax^b]
x?40           :ŷ=  62.8685293
```

(Estimated value for y following input of 40 volts)

EXE

```
Estimation of y  [y=ax^b]
x?_
```

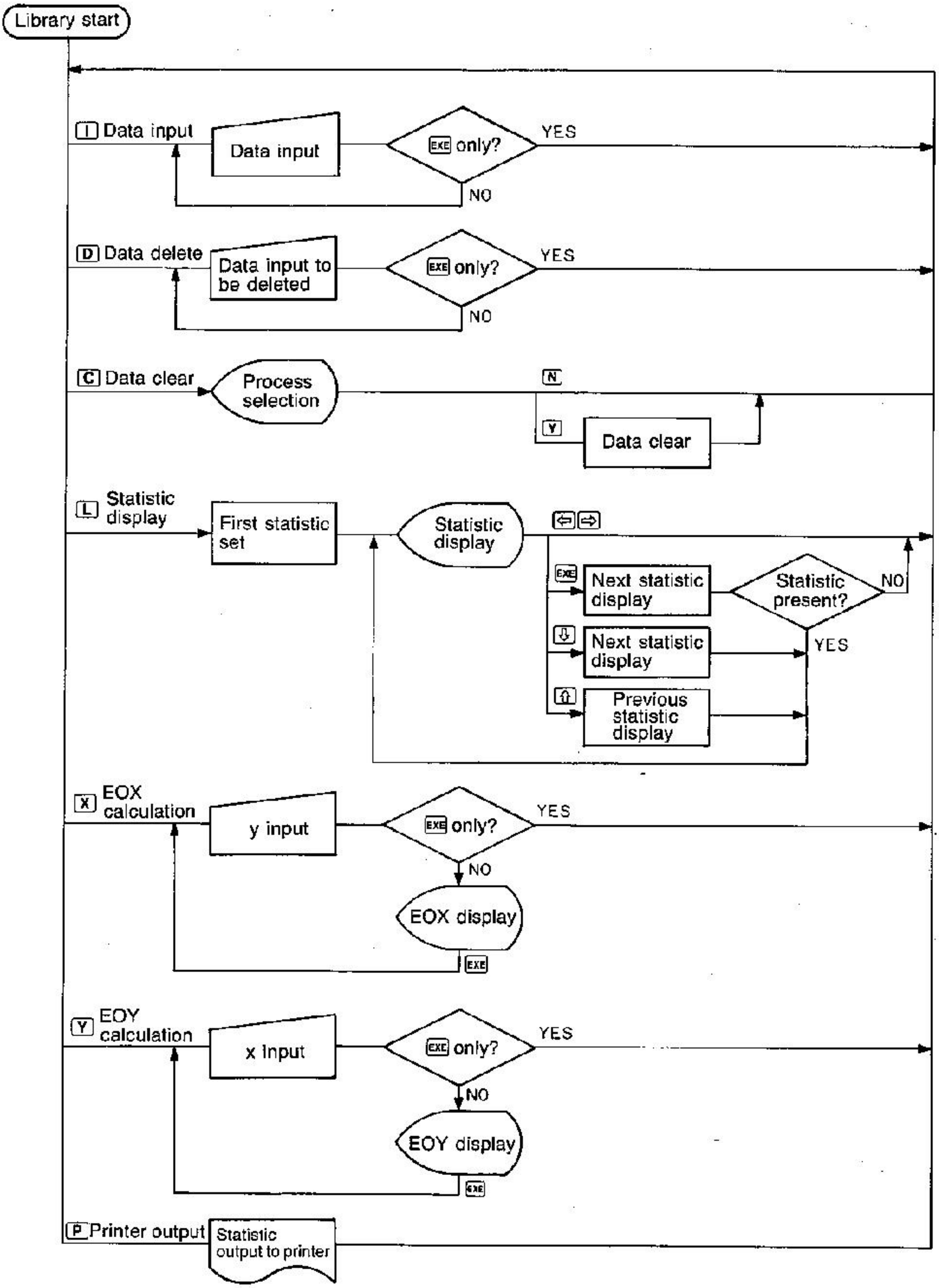
EXE

```
Regression analysis         [y=ax^b]
>ln.Del.Clear.List.eoX.eoY.P  ?_
```

(Return to menu display)

Here, these data produce the power curve $y = 1.069436811 \times x^{1.104376978}$. Also, input of 40 volts results in an estimated current of 62.9mA.

REGRESSION ANALYSIS FLOWCHART (2.1,2.2,2.3,2.4)



3

COMPLEX NUMBER

Complex number calculations encompass arithmetic operations, and to determine absolute values, arguments, squares, square roots, and reciprocal numbers.
This unit is capable of a wide variety of complex number calculations, with the allowable range of input value$<$1E50.

OPERATION

Fx 3

```
 0
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

The complex number menu display allows selection of the following processes:

A :	Input of complex number A (a+bi)	
G :	Absolute value (r) and arguments (θ) for complex number A (resulting angle unit determined by current mode setting)	
I :	Reciprocal number for complex number A	$1/(a+bi) \rightarrow (a+bi)$
S :	Square root of complex number A	$\sqrt{(a+bi)} \rightarrow (a+bi)$
^ :	Square of complex number A	$(a+bi)^2 \rightarrow (a+bi)$
+ :	Addition of complex number A and complex number B (c+di)	$(a+bi) + (c+di) \rightarrow (a+bi)$
− :	Subtraction of complex number A and complex number B	$(a+bi) - (c+di) \rightarrow (a+bi)$
* :	Multiplication of complex number A and complex number B	$(a+bi) \times (c+di) \rightarrow (a+bi)$
/ :	Division of complex number A and complex number B	$(a+bi) \div (c+di) \rightarrow (a+bi)$
M :	Assigns contents of complex number A to complex number memory M (e+fi)	$(a+bi) \rightarrow (e+fi)$
L :	Assigns contents of complex number memory M (e+fi) to complex number A	$(a+bi) \leftarrow (e+fi)$
C :	Exchanges contents of complex number A and complex number B	$(a+bi) \leftrightarrow (c+di)$
. :	Help (explanation of each operation)	

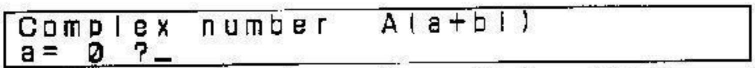
• Complex Number Specification

Complex number specification is performed by pressing [A] while in the menu display.

EXAMPLE

Assign 5+7i to complex number A.

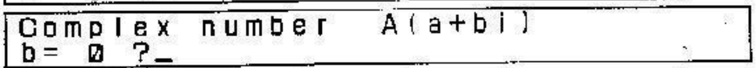
[A]



```
Complex number  A(a+bi)
a= 0 ?_
```

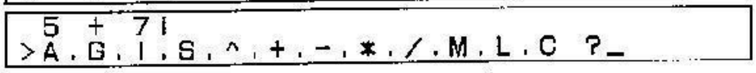
(Specification of complex number input)

5 [EXE]



```
Complex number  A(a+bi)
b= 0 ?_
```

7 [EXE]



```
 5 + 7i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

• Arithmetic Operations

EXAMPLE

Perform the following operations:

$(2+3i) + (3-2i)$

A

```
Complex number  A(a+bi)
a= 0 ?_
```

(Specification of complex number input)

2 EXE **3** EXE

```
 2 + 3i
<A.G.I.S.^.+.-.*./.M.L.C ?_
```

(Input of complex number A)

+

```
Complex number  B(c+di)
c= 0  ?_
```

(Addition)

3 EXE − **2** EXE

```
 5 + i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

(Input of complex number B)

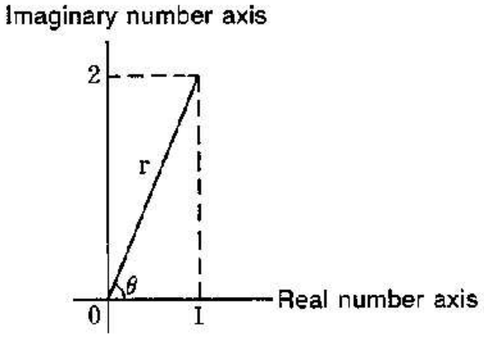
This display indicates $(2+3i) + (3-2i) = 5+i$.
The same procedure can be performed for subtraction, multiplication and division.

• Absolute Values/Arguments

EXAMPLE

Determine the absolute value (r) and argument (θ) for $(1+2i)$.

Angle unit : DEG (MODE 4)



A

```
Complex number  A(a+bi)
a= 0 ?_
```

(Specification of complex number input)

1 EXE **2** EXE

```
 1 + 2i
>A.G.I.S.^.+.-.*./.M.L.C?_
```

(Input of complex number A)

G

```
r  =  2.236067977
θ  =  63.43494882
```

(Calculation of absolute value and argument)

Here, the absolute value (r) for $(1+2i)$ is 2.236067977, and the argument is 63.43494882 (DEG). The resulting angle unit is determined by the current ANGLE mode setting.

* The angle unit is specified as follows:

MODE 4 : Degrees
MODE 5 : Radians
MODE 6 : Grads

• Square/Square Root/Reciprocal number

EXAMPLE

Calculate the following:

① $(2+i)^2$ ② $\sqrt{(-7+24i)}$ ③ $\frac{1}{3+2i}$

① Square

A
```
Complex number  A(a+bi)
a= 0 ?_
```
(Specification of complex number input)

2 EXE 1 EXE
```
 2 + i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Input of complex number A)

^
```
 3 + 4i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Square)

This display indicates $(2+i)^2=3+4i$.

② Square Root

A
```
Complex number  A(a+bi)
a= 0 ?_
```
(Specification of complex number input)

- 7 EXE 24 EXE
```
-7 + 24i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Input of complex number A)

S
```
 3 + 4i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Square root)

This display indicates $\sqrt{(-7+24i)}=3+4i$.

③ Reciprocal Number

A
```
Complex number  A(a+bi)
a= 0 ?_
```
(Specification of complex number input)

3 EXE 2 EXE
```
 3 + 2i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Input of complex number A)

I
```
 0.2307692 - 0.1538462i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Reciprocal number)

• Memory Calculations

EXAMPLE

Perform the following calculations using the memory function:

$(3+2i) + (4+6i)$
$(3+2i) - (-3+9i)$

A
```
Complex number  A(a+bi)
a= 0 ?_
```
(Specification of complex number input)

3 EXE 2 EXE
```
 3 + 2i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Input of complex number A)

M
```
 3 + 2i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Assigns complex number A to complex number memory)

+
```
Complex number  B(c+di)
c= 0 ?_
```
(Addition)

4 EXE 6 EXE
```
 7 + 8i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```
(Assigns 4+6i to B)

This display indicates $(3+2i) + (4+6i) = 7+8i$.

$\boxed{L}$

```
 3 + 2i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

(Assigns complex number in memory to complex number A in place of reinput)

$\boxed{-}$

```
Complex number  B(c+di)
c= 4 ?_
```

(Subtraction)

$\boxed{-}$ **3** $\boxed{EXE}$ **9** $\boxed{EXE}$

```
 6 - 7i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

(Assigns −3+9i to B)

This display indicates (3+2i) − (−3+9i) = 6−7i.

- **Exchange**

EXAMPLE

Set the following two complex numbers for complex numbers A and B:

(5+2i), (3+4i)

$\boxed{A}$

```
Complex number  A(a+bi)
a= 0 ?
```

(Complex number A input)

3 $\boxed{EXE}$ **4** $\boxed{EXE}$

```
 3 + 4i
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

(First set 3+4i for complex number A)

$\boxed{C}$

```
0
>A.G.I.S.^.+.-.*./.M.L.C ?_
```

(Assign the contents of complex number A to complex number B)

$\boxed{A}$

```
Complex number  A(a+bi)
a= 0 ?
```

(Input complex number A)

5 $\boxed{EXE}$ **2** $\boxed{EXE}$

```
 5 + 2i
>A.G.I.S.^.+.-.*./.M.L.C.?_
```

(Set 5+2i for complex number A)

The above operation sets 5+2i for complex number A, and 3+4i for complex number B.

*** Help Display**

Pressing $\boxed{\cdot}$ while in the menu display produces an explanation of each command.

```
 A : Input A
 G : Gauss  r=|A|:θ=argument(A)
```

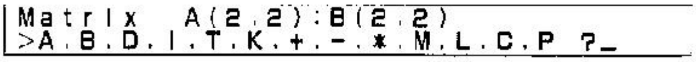
At this time $\boxed{⇧}$, $\boxed{EXE}$, and $\boxed{⇩}$ can be used to scroll the display. Each press of $\boxed{EXE}$ ($\boxed{⇩}$) advances to the next command, while pressing $\boxed{⇧}$ returns to the previous command. Pressing $\boxed{⇦}$ or $\boxed{⇨}$ returns to the menu display. The menu display is also returned to after the final command is displayed.

4 MATRIX OPERATIONS

Matrix operations make it possible to perform addition, subtraction, multiplication, scalar product, determinant, inverse matrix, and transposed matrix calculations.

OPERATION

Fx 4



```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

The following process can be selected from the menu display illustrated above.

A : Definition of MATRIX A and data input
B : Definition of MATRIX B and data input
D : Determinant of MATRIX A (det A)
I : Inverse matrix of MATRIX A and assignment of result to MATRIX A ($A^{-1} \rightarrow A$)
T : Transposed matrix of MATRIX A and assignment of result to MATRIX A ($A^{t} \rightarrow A$)
K : Scalar product of MATRIX A and assignment of result to MATRIX A (kA→A)
+ : Addition of MATRIX A and MATRIX B and assignment of result to MATRIX A (A + B→A)
− : Subtraction of MATRIX A and MATRIX B and assignment of result to MATRIX A (A − B→A)
∗ : Multiplication of MATRIX A and MATRIX B and assignment of result to MATRIX A (A·B→A)
M : Assignment of MATRIX A contents to MEMORY MATRIX M (A→M)
L : Assignment of MEMORY MATRIX M contents to MATRIX A (M→A)
C : Exchange of MATRIX A and MATRIX B contents (A↔B)
P : Display of MATRIX A contents
· : Help display

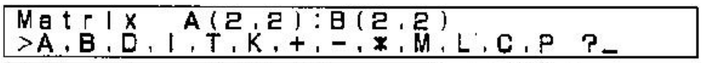
MATRIX SET UP

Select either [A] (MATRIX A) or [B] (MATRIX B) from the menu display for matrix set up.

EXAMPLE 1

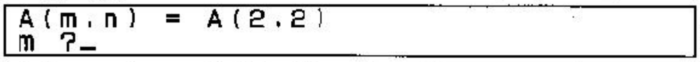
Set up the 3-row by 4-column matrix shown to the right.

$$\begin{array}{c|c} & \text{Column } (n) \rightarrow \\ \hline \text{Row } (m) \downarrow & \begin{bmatrix} 1 & 0 & 3 & 4 \\ 2 & 1 & 0 & -1 \\ 3 & 1 & -2 & 3 \end{bmatrix} \end{array}$$



```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

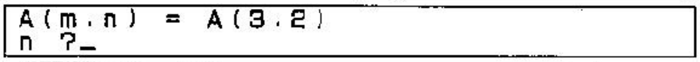
A



```
A(m,n)  =  A(2,2)
m ?_
```

(MATRIX A specification)

3 EXE



```
A(m,n)  =  A(3,2)
n ?_
```

(Row input)

4 EXE

```
a(1,1)  =  0
?_
```

(Column input)

* A 2-row by 2-column $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ matrix is automatically set up when this library is activated.

Now enter the elements in the sequence shown in the illustration to the right (①~⑫).

$$\begin{bmatrix} 1 & 0 & 3 & 4 \\ \text{①} & \text{②} & \text{③} & \text{④} \\ 2 & 1 & 0 & -1 \\ \text{⑤} & \text{⑥} & \text{⑦} & \text{⑧} \\ 3 & 1 & -2 & -3 \\ \text{⑨} & \text{⑩} & \text{⑪} & \text{⑫} \end{bmatrix}$$

```
a(1.1)=0
?_
```

1 EXE

```
a(1.2)  =  0
?_
```

(Matrix element entries)

0 EXE

```
a(1.3)  =  0
?_
```

The unit returns to the menu display once input of all of the elements is complete. At this point, it is advisable to review the values to confirm that input was performed correctly.

A

```
A(m.n)  =  A(3.4)
m  ?_
```

(MATRIX A specification)

EXE

```
A(m.n)  =  A(3.4)
n  ?_
```

(Press EXE after confirmation)

EXE

```
a(1.1)  =  1
?_
```

EXE

```
a(1.2)  =  0
?_
```

⋮

EXE

```
a(3.3)  =-2
?_
```

EXE

```
a(3.4)  =-3
?_
```

EXE

```
Matrix  A(3.4):B(2.2)
>A.B.D.I.T.K.+.-.*.M.L.C.P  ?_
```

(Return to initial display)

CORRECTION

Errors discovered before the EXE key is pressed can be corrected by simply entering the correct value and then pressing EXE. After EXE is pressed, press SHIFT EXE to return to the previous value display and then make necessary corrections.

* The P command can also be used to view matrix contents.

Matrix addition/subtraction/multiplication

EXAMPLE 2

$$A=\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad B=\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

Perform A+B, A−B, A•B, and B•A for the two following matrices.

Perform the following operation from the menu display.

```
Matrix  A(2.2):B(2.2)
>A.B.D.I.T.K.+.-.*.M.L.C.P  ?_
```

A

```
A(m,n) = A(2,2)
m ?_
```

(MATRIX A specification)

2 EXE **2** EXE

```
a(1,1) = 0
?_
```

(2-row/2-column specification)

1 EXE **1** EXE **2** EXE **1** EXE

```
Matrix A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element Input)

B

```
B(m,n) = B(2,2)
m?_
```

(MATRIX B specification)

2 EXE **2** EXE

```
b(1,1) = 0
?_
```

(2-row/2-column specification)

2 EXE **3** EXE **2** EXE **1** EXE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element input)

M

```
Memory  A → M(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Transfer of MATRIX A to MATRIX M)

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

The results of most matrix operations are stored in MATRIX A, deleting any contents currently stored in MATRIX A. Therefore, it is advisable to first transfer the contents from MATRIX A to MATRIX M so they can be recalled if later required before performing a matrix operation.

Once matrix set up is complete, proceed with the following calculations.

- A+B

+

```
A+B → A(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Addition specification)

```
a(1,1) = 3
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Result display)

EXE

```
a(1,2) = 4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Press EXE after confirmation)

EXE

```
a(2,1) = 4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

EXE

```
a(2,2) = 2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

EXE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Return to initial display)

Here, the result of A+B is $\begin{bmatrix} 3 & 4 \\ 4 & 2 \end{bmatrix}$.

- A−B

L

```
Load  A ← M(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Transfer of MATRIX M elements to MATRIX A)

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

−

```
A-B → A(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Subtraction specification)

```
a(1,1) =-1
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Result display)

EXE

```
a(1,2) =-2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Press EXE after confirmation)

EXE

```
a(2,1) = 0
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(2,2) = 0
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Return to initial display)

Here, the result of A − B is $\begin{bmatrix} -1 & -2 \\ 0 & 0 \end{bmatrix}$.

- A•B

L
```
Load  A ← M(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
*
```
A*B → A(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Multiplication specification)
```
a(1,1) = 4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Result display)

EXE
```
a(1,2) = 4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Press EXE after confirmation)

EXE
```
a(2,1) = 6
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
EXE
```
a(2,2) = 7
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
EXE
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Return to initial display)

Here, the result of A• B is $\begin{bmatrix} 4 & 4 \\ 6 & 7 \end{bmatrix}$.

- B•A

L
```
Load  A ← M(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Transfer of MATRIX M elements to MATRIX A)

C
```
Change  A(2,2) ←→ B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Exchange of MATRIX A and MATRIX B elements)
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
*
```
A*B → A(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Multiplication specification)
```
a(1,1) = 8
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Result display)

EXE
```
a(1,2) = 5
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Press EXE after confirmation)

EXE
```
a(2,1) = 4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
EXE
```
a(2,2) = 3
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
EXE
```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Return to initial display)

Here, the result of B• A is $\begin{bmatrix} 8 & 5 \\ 4 & 3 \end{bmatrix}$.

EXAMPLE 3

Calculate the determinant for the following matrix.

$$\begin{bmatrix} 2 & 1 \\ 0 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 3 & -1 & 1 \\ 0 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 \\ 2 & -3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

First perform the multiplication in the first term by setting up the following matrices and then executing A· B.

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -1 \\ 1 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 2 & 1 \end{bmatrix}$$

```
Matrix  A(2,2),B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

A3 EXE **2** EXE

```
a(1,1)  = 0
?_
```

(MATRIX A set up)

2 EXE **1** EXE **0** EXE **−1** EXE **1** EXE **3** EXE

```
Matrix  A(3,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element input)

B2 EXE **3** EXE

```
b(1,1)  = 0
?_
```

(MATRIX B set up)

3 EXE **−1** EXE **1** EXE **0** EXE **2** EXE **1** EXE

```
Matrix  A(3,2):B(2,3)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element input)

*****

```
A*B  →  A(3,3)
>A,B,D,I,T,K,t,-,*,M,L,C,P ?
```

(A· B calculation)

```
a(1,1)  =  6
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Result display)

Next perform calculation for second term.

B3 EXE **3** EXE

```
b(1,1)  = 0
?_
```

(MATRIX B set up)

1 EXE **0** EXE **1** EXE **2** EXE **−3** EXE **0** EXE **0** EXE **0** EXE **2** EXE

```
Matrix  A(3,3):B(3,3)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element input)

+

```
A+B  →  A(3,3)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

```
a(1,1)  =  7
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(1,2)  =  0
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Press EXE after confirmation)

EXE

```
a(1,3)  =  4
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(2,1)  =  2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(2,2)  =-5
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(2,3)  =-1
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE
```
a(3,1) = 3
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE
```
a(3,2) = 5
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE
```
a(3,3) = 6
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE
```
Matrix  A(3,3):B(3,3)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Return to initial display)

Here, the result of the calculation is

$$\begin{bmatrix} 2 & 1 \\ 0 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 3 & -1 & 1 \\ 0 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 1 \\ 2 & -3 & 0 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 0 & 4 \\ 2 & -5 & -1 \\ 3 & 5 & 6 \end{bmatrix}$$

Determinant, inverse matrix, and transposed matrix

EXAMPLE 4

Determine the determinant, inverse matrix and transposed matrix for the following 3-column/3-row matrix.

$$\begin{bmatrix} 2 & 0 & 0 \\ 3 & 1 & 2 \\ 4 & 2 & 3 \end{bmatrix}$$

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

A3 EXE 3 EXE
```
a(1,1) = 0
?_
```
(MATRIX A set up)

2 EXE 0 EXE 0 EXE 3 EXE 1 EXE 2 EXE 4 EXE 2 EXE 3 EXE
```
Matrix  A(3,3):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Element input)

M
```
Memory  A → M(3,3)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Transfer of MATRIX A to MEMORY MATRIX M)
```
Matrix  A(3,3):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

- Determinant (det A)

D
```
Determinant A = .....
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Determinant)
```
Determinant A =-2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Result display)

EXE
```
Matrix  A(3,3):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```
(Return to menu display)

Here, the determinant of MATRIX A is −2.

- Inverse matrix (A^{-1})

I
```
Inverse A → A
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Inverse matrix)
```
a(1,1) = 0.5
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```
(Inverse matrix display)

EXE
```
a(1,2)  = 0
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```
(Press EXE after confirmation)

EXE
```
a(1,3)  = 0
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,1)  = 0.5
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,2)  =-3
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,3)  = 2
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,1)  =-1
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,2)  = 2
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,3)  =-1
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
Matrix  A(3,3):B(2,2)
>A.B.D.I.T.K.+.-.*.M.L.C.P ?_
```
(Return to initial display)

Here, the inverse matrix of MATRIX A (A^{-1}) is $\begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & -3 & 2 \\ -1 & 2 & -1 \end{bmatrix}$

- Transposed matrix (A^t)

L
```
Load  A ← M(3,3)
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```
(Transfer of MEMORY MATRIX M to MATRIX A)
```
Matrix  A(3,3):B(2,2)
>A.B.D.I.T.K.+.-.*.M.L.C.P ?_
```

T
```
Transpose  A → A(3,3)
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```
(Transposed matrix)
```
a(1,1)  = 2
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```
(Transposed matrix display)

EXE
```
a(1,2)  = 3
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```
(Press EXE after confirmation)

EXE
```
a(1,3)  = 4
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,1)  = 0
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,2)  = 1
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(2,3)  = 2
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,1)  = 0
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,2)  = 2
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
a(3,3)  = 3
>A.B.D.I.T.K.+.-.*.M.L.C.P ?
```

EXE
```
Matrix  A(3,3):B(2,2)
>A.B.D.I.T.K.+.-.*.M.L.C.P ?_
```
(Return to initial display)

Here, the transposed matrix for MATRIX A (A^t) is $\begin{bmatrix} 2 & 3 & 4 \\ 0 & 1 & 2 \\ 0 & 2 & 3 \end{bmatrix}$.

Scalar product

EXAMPLE 5

Calculate the scalar products for the following matrices.

$$3\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}\begin{bmatrix}3 & 1\\ 0 & 2\end{bmatrix}$$

$$A=\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix},\quad B=\begin{bmatrix}3 & 1\\ 0 & 2\end{bmatrix}$$

Multiply MATRIX B by the result of MATRIX A times three.

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

A2 EXE 2 EXE

```
a(1,1) = 0
?_
```

(MATRIX A set up)

1 EXE 2 EXE 2 EXE 1 EXE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Element input)

K

```
k*A(2,2) → A
k = 1 ?_
```

(Scalar product)

3 EXE

```
a(1,1) = 3
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Multiplication by 3 result assigned to MATRIX A)

B2 EXE 2 EXE

```
b(1,1) = 0
?_
```

(MATRIX B set up)

3 EXE 1 EXE 0 EXE 2 EXE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

*****

```
A*B → A(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(A and B multiplication)

EXE

```
a(1,1) = 9
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Display result)

EXE

```
a(1,2) = 15
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

(Press EXE after confirmation)

EXE

```
a(2,1) = 18
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
a(2,2) = 12
>A,B,D,I,T,K,+,-,*,M,L,C,P ?
```

EXE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Return to initial display)

Here, the result of the example calculation is $3\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}\begin{bmatrix}3 & 1\\ 0 & 2\end{bmatrix}=\begin{bmatrix}9 & 15\\ 18 & 12\end{bmatrix}$.

- HELP menu

Pressing ⊡ in the menu display produces a HELP display which explains the meaning of each command.

```
A : input A(m,n)
B : input B(m,n)
```

Pressing ⇧, ⇩ or EXE at this time scrolls through the commands. Pressing either ⇦ or ⇨ returns to the initial display.

- Matrix display

After performing matrix addition, subtraction, multiplication, scalar product, determinant, inverse matrix, and transposed matrix calculations, the result of the calculation (contents of MATRIX A) is shown on the display. As with the HELP menu, [⇩] and [⇧] ([EXE]) can be used to scroll through MATRIX A.

* The operation of [⇩] and [EXE] is identical, with display being performed in the same sequence as the matrix element input. The [⇧] key displays the elements in reverse sequence.
* Pressing [⇦] or [⇨] returns to the menu display regardless of the current display.
* The [P] key can be used from the menu display to display the contents of MATRIX A. [⇧], [⇩], [EXE], [⇨] and [⇦] can also be used as desired.

EXAMPLE

```
Matrix  A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

P

```
a(1,1) = 1
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(MATRIX A element display selection)

[⇩]

```
a(1,2) = 2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Confirmation of each element)

[EXE]

```
a(2,1) = 3
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

[⇧]

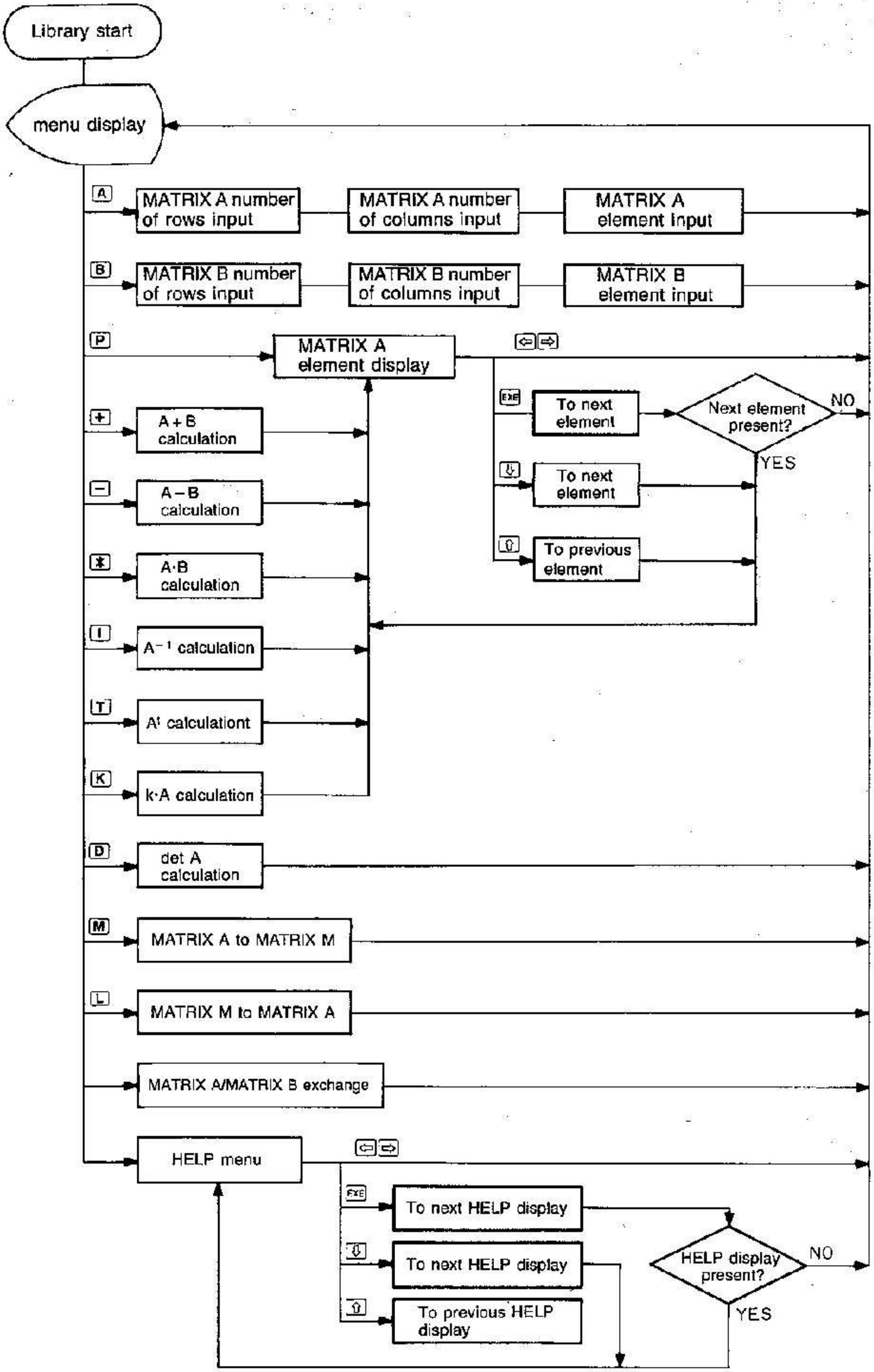
```
a(1,2) = 2
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

[⇨]

```
Matrix A(2,2):B(2,2)
>A,B,D,I,T,K,+,-,*,M,L,C,P ?_
```

(Return to initial display)

MATRIX OPERATION FLOWCHART



5

BINARY-DECIMAL-HEXADECIMAL

Binary, decimal and hexadecimal calculations encompass basic arithmetic operations, logical operations, twos complement, logical shift, and conversions.
This unit is capable of combining binary, decimal and hexadecimal values, with the allowable range values being −2147483648～2147483647 (32-bit).

OPERATION

Fx 5

```
[DEC]   0
>I,B,D,H,+,-,*,/,A,O,X,N,C,L,R?_
```

The binary/decimal/hexadecimal calculation menu display allows selection of the following processes:

I : Value input
B : Converts displayed value to binary number
D : Converts displayed value to decimal number
H : Converts displayed value to hexadecimal number
+ : Addition
− : Subtraction
* : Multiplication
/ : Division
A : AND (logical product)
O : OR (logical sum)
X : XOR (exclusive logical sum)
N : NOT (negation)
C : Twos complement
L : Logical shift left
R : Logical shift right
. : Help (explanation of each operation)

*** Operations and Display**

1. The following indicators in the upper left of the display in the menu indicate the current base mode setting:
   [DEC] : Decimal mode
   [HEX] : Hexadecimal mode
   Blank : Binary mode
2. Entering values besides 0 and 1 for binary calculations, values besides 0～9 for decimal calculations, values besides 0～9/A～F (upper case or lower case) for hexadecimal calculations, or values greater than 32 bits causes the entered value to be disregarded. Binary, decimal and hexadecimal values may be used in combination in a single calculation.

EXAMPLE

The following operations may be used to enter values regardless of the current base mode setting:

15, D : Decimal 15 (hexadecimal F, binary 1111)
15, H : Hexadecimal 15 (decimal 21, binary 10101)
1010, B : Binary 1010 (decimal 10, hexadecimal A)

Results are always displayed using the current base mode setting.

• Arithmetic Operations

EXAMPLE

Perform the following calculations:

① $1011001_B + 1100_B$
② $2AC_H \times 1B_H$
③ $FF00_H + 1010_B$

①

```
[DEC]  0
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

[B]

```
00000000000000000000000000000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Binary mode)

[I]

```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

1011001 [EXE]

```
00000000000000000000000001011001
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[+]

```
00000000000000000000000001011001
 +_
```

(Addition)

1100 [EXE]

```
00000000000000000000000001100101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

This display indicates $1011001_B + 1100_B = 1100101_B$

②

[H]

```
[HEX] 00000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Hexadecimal mode)

[I]

```
Input data x (.B.D.H)       [HEX]
x ?_
```

(Value input specification)

2 [A] [C] [EXE]

```
[HEX] 000002AC
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[*]

```
[HEX] 000002AC *_
```

(Multiplication)

1 [B] [EXE]

```
[HEX] 00004824
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

This display indicates $2AC_H \times 1B_H = 4824_H$.

The same procedure can be performed for subtraction and division.

③

[H]

```
[HEX] 00000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Hexadecimal mode)

[I]

```
Input data x (.B.D.H)       [HEX]
x ?_
```

(Value input specification)

[F] [F] 00 [EXE]

```
[HEX] 0000FF00
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[+]

```
[HEX] 0000FF00 + _
```

(Addition)

1010 [,] [B] [EXE]

```
[HEX] 0000FF0A
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Binary value input)

This display indicates $FF00_H + 1010_B = FF0A_H$

• Logical Operations

EXAMPLE

Perform the following operations for $A = 110101_B$ and $B = 101111_B$.

① A OR B (logical sum) ② A AND B (logical product)
③ A XOR B (exclusive logical sum) ④ A NOT (negation)

①

```
[DEC] 0
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

[B]

```
00000000000000000000000000000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Binary mode)

[I]

```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

110101 [EXE]

```
00000000000000000000000000110101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[O]

```
00000000000000000000000000110101
  OR _
```

(OR)

101111 [EXE]

```
00000000000000000000000000111111
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

This display indicates A OR $B = 111111_B$.

②

[I]

```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

110101 [EXE]

```
00000000000000000000000000110101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[A]

```
00000000000000000000000000110101
  AND
```

(AND)

101111 [EXE]

```
00000000000000000000000000100101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

This display indicates A AND $B = 100101_B$.

③

[I]

```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

110101 [EXE]

```
00000000000000000000000000110101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[X]

```
00000000000000000000000000110101
  XOR _
```

(XOR)

101111 [EXE]

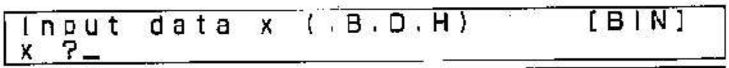
```
00000000000000000000000000110110
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

This display indicates A XOR $B = 11010_B$.

④

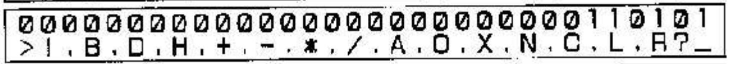
I



```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

110101 **EXE**



```
00000000000000000000000000110101
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

N

```
11111111111111111111111111001010
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(NOT)

This display indicates NOT A = 11111111111111111111111111001010B.

• Complement/Shift Operations

EXAMPLE

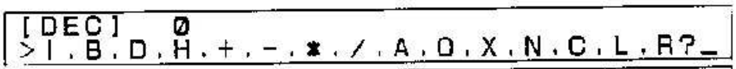
Perform the following operations:

①Twos complement of 11001010B

②1-bit logical shift left of 110000B

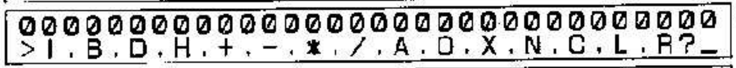
③2-bit logical shift right of 1FCH

①



```
[DEC]  0
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

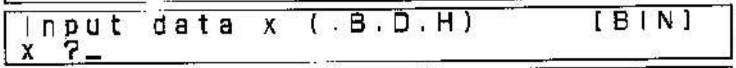
B



```
00000000000000000000000000000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Binary mode)

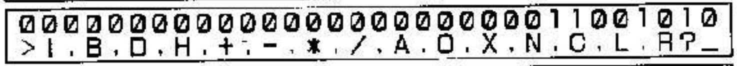
I



```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

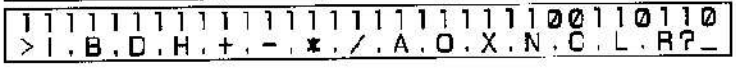
11001010 **EXE**



```
00000000000000000000000011001010
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

C



```
11111111111111111111111100110110
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Twos complement)

This display indicates that the twos complement of 11001010B is 11111111111111111111111100110110B.

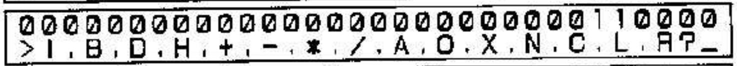
②

I

```
Input data x (.B.D.H)       [BIN]
x ?_
```

(Value input specification)

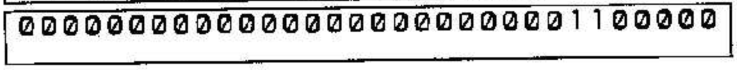
110000 **EXE**



```
00000000000000000000000000110000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

L



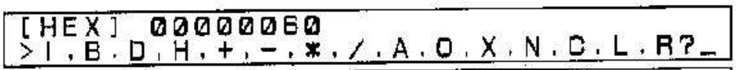
```
00000000000000000000000001100000
```

(Shift left)

This display indicates that shifting 110000B one bit to the left results in 1100000B.

③

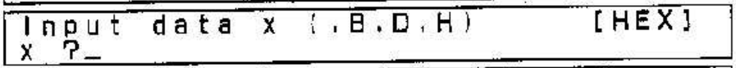
H



```
[HEX]  00000060
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Hexadecimal mode)

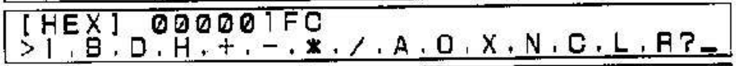
I



```
Input data x (.B.D.H)       [HEX]
x ?_
```

(Value input specification)

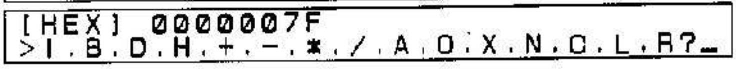
1 **F** **C** **EXE**



```
[HEX]  000001FC
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

R **R**



```
[HEX]  0000007F
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

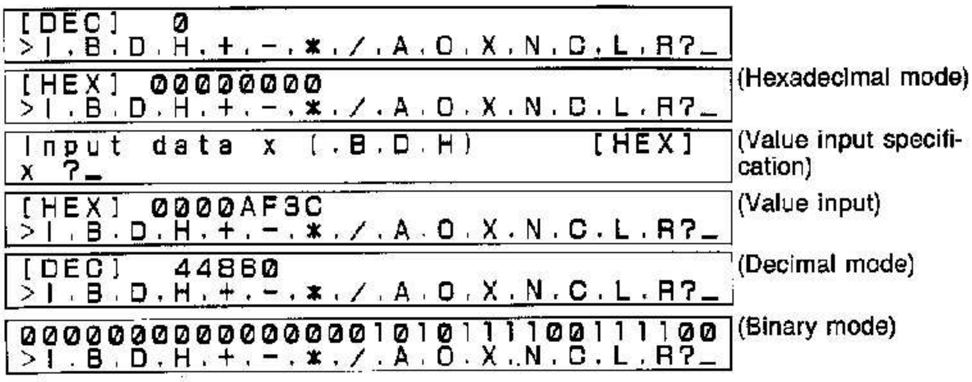
(2-bit shift right)

This display indicates that shifting 1FCH two bits to the right results in 7FH.

• Base conversion

EXAMPLE

Convert the hexadecimal value AF3C to its decimal and binary equivalents.



```
[DEC]   0
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

[H]

```
[HEX]  00000000
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Hexadecimal mode)

[I]

```
Input data x (.B.D.H)      [HEX]
x ?_
```

(Value input specification)

[A] [F] 3 [C] [EXE]

```
[HEX]  0000AF3C
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Value input)

[D]

```
[DEC]   44860
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Decimal mode)

[B]

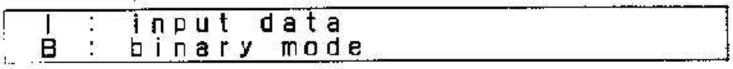
```
00000000000000001010111100111100
>I.B.D.H.+.-.*./.A.O.X.N.C.L.R?_
```

(Binary mode)

This display indicates that the decimal equivalent of hexadecimal AF3C is 44860, and the binary equivalent is 1010111100111100B.

*** Help Display**

Pressing [•] while in the menu display produces an explanation of each command.



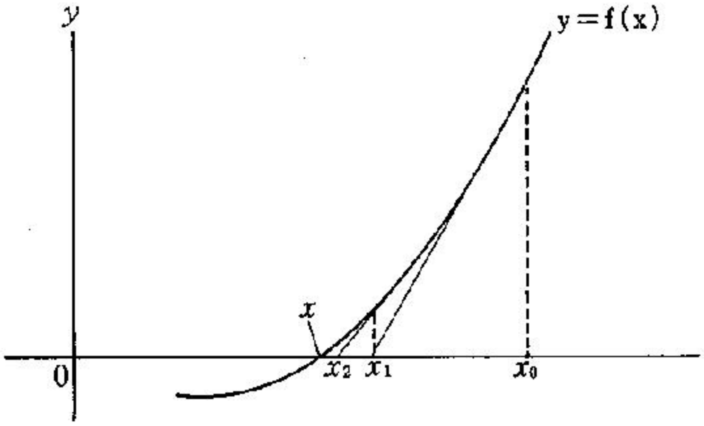
```
 I  :  input data
 B  :  binary mode
```

At this time [⇧], [EXE], and [⇩] can be used to scroll the display. Each press of [EXE] ([⇩]) advances to the next command, while pressing [⇧] returns to the previous command. Pressing [⇦] or [⇨] returns to the menu display. The menu display is also returned to after the final command is displayed.

6

NUMERIC SOLUTION OF AN EQUATION
(NEWTON'S METHOD)

Determines the solution of the function y = f(x) graphed below for f(x) = 0, using Newton's Method.



(Angle unit = radians)

The following parameters are specified in order to determine the numeric solution using Newton's Method.

x_0 : Initial value
h : Minute interval for x-axis when performing numerical differentiation at point (x, f(x))

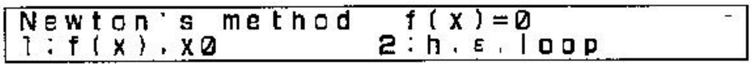
$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

ϵ : Solution convergence ($\epsilon > |x_{n+1} - x_n|$: continuously calculate and return value of ϵ as long as inequality is true)
loop : Maximum number of convergences (positive integer)

* The following arithmetic operators and functions can be applied here:
+, −, *, /, ^ , SIN, COS, TAN, ASN, ACS, ATN, LOG, LN, EXP, SQR, HYP
* The variable name for the function f(x) must be represented by x.
* The value input for ϵ must be 1E−90 or more. Since internal calculations are performed in 12 digits, smaller values have little meaning.

OPERATION

Fx 6



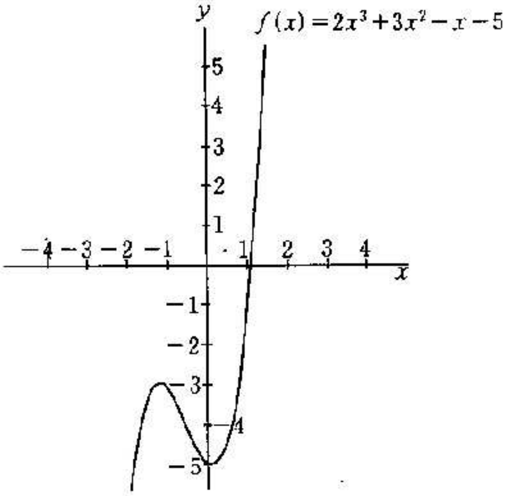
```
Newton's method   f(x)=0
1:f(x),X0         2:h,ε,loop
```

The menu display illustrated above appears when the library is activated. Either 1 or 2 should be selected in accordance with the type of processing to be performed.

1 : Function f(x) specification/initial value input
2 : Input of minute interval, convergence condition, and maximum number of convergences

EXAMPLE

Determine the f(x) = 0 solution of the following equation for $f(x) = 2x^3 + 3x^2 - x - 5$, where the minute interval is 0.00001, the convergence condition is 0.0001, and the maximum number of convergences is 30.



```
Newton's method   f(x)=0
1:f(x),x0          2:h,ε,loop
```

2

```
f'(x)=(f(x+h)-f(x))/h   (h>0)
h= 0.00001 ?_
```

(Parameter input selection)

0.00001 EXE

```
Err |Xn+1-Xn|<ε   (ε>0)
ε= 0.0000001 ?_
```

(Minute interval input)

0.0001 EXE

```
Max loop   (n>0)
n= 20 ?_
```

(Convergence condition input)

30 EXE

```
Newton's method   f(x)=0
1:f(x),x0          2:h,ε,loop
```

(Maximum number of convergences input)

1

```
Define function
f(x) ?_
```

(Function/initial value input selection)

2*x^3+3*x^2−x−5 EXE

```
f(x)  = 2*x^3+3*x^2-x-5
x0= 0 ?_
```

(Function input)

1 EXE

```
f(x)  = 2*x^3+3*x^2-x-5
. . . . .
```

(Initial value input)

```
f(x)  = 2*x^3+3*x^2-x-5
x  = 1.0849
```

(Solution display)

EXE

```
Newton's method   f(x)=0
1:f(x),x0          2:h,ε,loop
```

(Return to initial display)

This display indicates that the solution for the example equation is 1.0849.

The message "not found" is displayed when a solution cannot be found.

```
f(x)  = x^2+1
not found
```

Pressing EXE at this time returns the display to point at which calculation was discontinued. Pressing EXE again returns to menu for numeric solution of an equation (certain calcualtions may not initially display discontinued point display).

```
f(x)  = x^2+1
loop = 20   :Xn = 1.070479459
```

7 Intégrales

8 QUADRATIC EQUATION

Determines the solution for α and β when coefficients a, b, and c are input for the quadratic equation $ax^2+bx+c=0$.
Root equations are used to determine the solution.

Root equation: $x=\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$

When $d=b^2-4ac$:

i) When $d>0$, two real roots (α, β) are present: $\alpha=\dfrac{-b+\sqrt{d}}{2a}$, $\beta=\dfrac{-b-\sqrt{d}}{2a}$

ii) When $d=0$, one real root (α) is present: $\alpha=\dfrac{-b}{2a}$ (multiple root)

iii) When $d<0$, two imaginary roots (α, β) are present:

$$\alpha=\frac{-b}{2a}+\frac{\sqrt{-d}}{2a}i,\ \beta=\frac{-b}{2a}-\frac{\sqrt{-d}}{2a}i$$

OPERATION

Fx 8

```
a x ² + b x + c = 0
a =  1  ?_
```

EXAMPLE

Determine the solution of the following quadratic equation:

$2x^2-5x+3=0$

```
a x ² + b x + c = 0
a =  1  ?_
```

2 EXE

```
a x ² + b x + c = 0
b =  0  ?_
```

(Coefficient a input)

−5 EXE

```
a x ² + b x + c = 0
c =  0  ?_
```

(Coefficient b input)

3 EXE

```
a x ² + b x + c = 0
 . . . . .
```

(Coefficient c input)

```
a ( x − α ) ( x − β ) = 0
α  =  1 . 5
```

(Solution α display)

EXE

```
a ( x − α ) ( x − β ) = 0
β  =  1
```

(Solution β display)

EXE

```
a x ² + b x + c = 0
a =  2  ?_
```

(Return to initial display)

Here, the solutions of $2x^2-5x+3=0$ are $\alpha=1.5$, $\beta=1$.

SOLUTION DISPLAY

Pressing EXE or ⇩ scrolls from display of α to β (only α displayed for multiple root). Pressing ⇧ while β is displayed returns to the display of α.

9 CUBIC EQUATIONS

Determines the solution for α, β and γ when coefficients a, b, c, and d are input for the cubic equation $ax^3+bx^2+cx+d=0$.
Root equations are used to determine the solution.
Transformation to $y^3+3py+q=0$ can be performed
when $x=y-\frac{b}{3a}$, $p=\frac{c}{3a}-\frac{b^2}{9a^2}$, $q=\frac{2b^3}{27a^3}-\frac{bc}{3a^2}+\frac{b}{a}$ are substituted in $ax^3+bx^2+cx=0$.

Here, substituting $A=\sqrt[3]{\frac{q+\sqrt{c}}{2}}$, $B=\sqrt[3]{\frac{q-\sqrt{c}}{2}}$, $c=q^2+4p^3$ results in the following:

i) When $c>0$, one real root (α), and two imaginary roots (β, γ) are present:

$$\alpha=-(A+B),\ \beta=\frac{A+B}{2}+\frac{\sqrt{3}}{2}(A-B)i,\ \gamma=\frac{A+B}{2}-\frac{\sqrt{3}}{2}(A-B)i$$

ii) When $c=0$, $p=0$, one real root (α) is present: $\alpha=-(A+B)$

iii) When $c=0$, $p\neq0$, two real roots (α, β) are present:

$$\alpha=-(A+B),\ \beta=\frac{A+B}{2}\qquad\text{(multiple roots)}$$

iv) When $c<0$, three real roots (α, β, γ) are present:

$$\alpha=-2\sqrt{-p}\cos\theta,\ \beta=-2\sqrt{-p}\cos(\theta+120°),$$
$$\gamma=-2\sqrt{-p}\cos(\theta+240°)\qquad \theta=\frac{1}{3}\cos^{-1}\frac{q}{2\sqrt{-p^3}}$$

OPERATION

Fx 9

$ax^3+bx^2+cx+d=0$
a= 1 ?_

EXAMPLE

Determine the solution of the following cubic equation:
$2x^3+x^2-13x+6=0$

$ax^3+bx^2+cx+d=0$
a= 1 ?_

2 EXE

$ax^3+bx^2+cx+d=0$
b= 0 ?_
(Coefficient a input)

1 EXE

$ax^3+bx^2+cx+d=0$
c= 0 ?_
(Coefficient b input)

−13 EXE

$ax^3+bx^2+cx+d=0$
d= 0 ?_
(Coefficient c input)

6 EXE

$ax^3+bx^2+cx+d=0$
. . . . .
(Coefficient d input)

$a(x-\alpha)(x-\beta)(x-\gamma)=0$
α =−3
(Solution α display)

EXE

$a(x-\alpha)(x-\beta)(x-\gamma)=0$
β = 2
(Solution β display)

A

SIMULTANEOUS EQUATIONS
(GAUSS-JORDAN ELIMINATION)

Solves for $x_1 \sim x_n$ in the following n simultaneous equations ($2 \leqq n \leqq 7$) for input of coefficients $a_1 \sim a_n$, $b_1 \sim b_n$ ··· and $y_1 \sim y_n$.

$$a_1 \cdot x_1 + b_1 \cdot x_2 + c_1 \cdot x_3 + \cdots = y_1$$
$$a_2 \cdot x_2 + b_2 \cdot x_2 + c_2 \cdot x_3 + \cdots = y_2$$
$$\vdots$$
$$a_n \cdot x_n + b_n \cdot x_n + c_n \cdot x_n + \cdots = y_n$$

Solutions may not be exact for coefficients with a difference in excess of 1×10^{10} due to internal rounding.

OPERATION

Fx A

```
ax1+bx2+cx3+···=y        (2≦n≦7)
n= 2 ?_
```

Pressing SHIFT EXE during coefficient input returns to the previous coefficient entry.
Pressing ⇩ or EXE during display of a solution scrolls to the following solution, while ⇧ scrolls to the previous solution.
The message "not found" appears on the display when a solution cannot be found.

EXAMPLE

Solve the following simultaneous cubic equations for x_1, x_2, and x_3.

$$2x_1 + 3x_2 - x_3 = 15$$
$$3x_1 - 2x_2 + 2x_3 = 4$$
$$5x_1 + 3x_2 - 4x_3 = 9$$

```
ax1+bx2+cx3+···=y        (2≦n≦7)
n= 2 ?_
```

3 EXE

```
ax1+bx2+cx3=y
1:a= 0 ?_
```

(Input 3 to specify cubic equations)

2 EXE

```
ax1+bx2+cx3=y
1:b= 0 ?_
```

(Input coefficients for first equation)

3 EXE

```
ax1+bx2+cx3=y
1:c= 0 ?_
```

(Input coefficients)

−1 EXE

```
ax1+bx2+cx3=y
1:y= 0 ?_
```

15 EXE

```
ax1+bx2+cx3=y
2:a= 0 ?_
```

3 EXE −2 EXE 2 EXE 4 EXE

```
ax1+bx2+cx3=y
3:a= 0 ?_
```

(Input coefficients for second equation)

5 EXE 3 EXE −4 EXE 9 EXE

```
ax1+bx2+cx3=y
x1  = .....
```

(Input coefficients for third equation)

```
ax1+bx2+cx3=y
x1  = 2
```

(Display value for x_1)

EXE

```
ax1+bx2+cx3=y
x2  = 5
```

(Display value for x_2)

EXE

```
ax1+bx2+cx3=y
x3  = 4
```

(Display value for x_3)

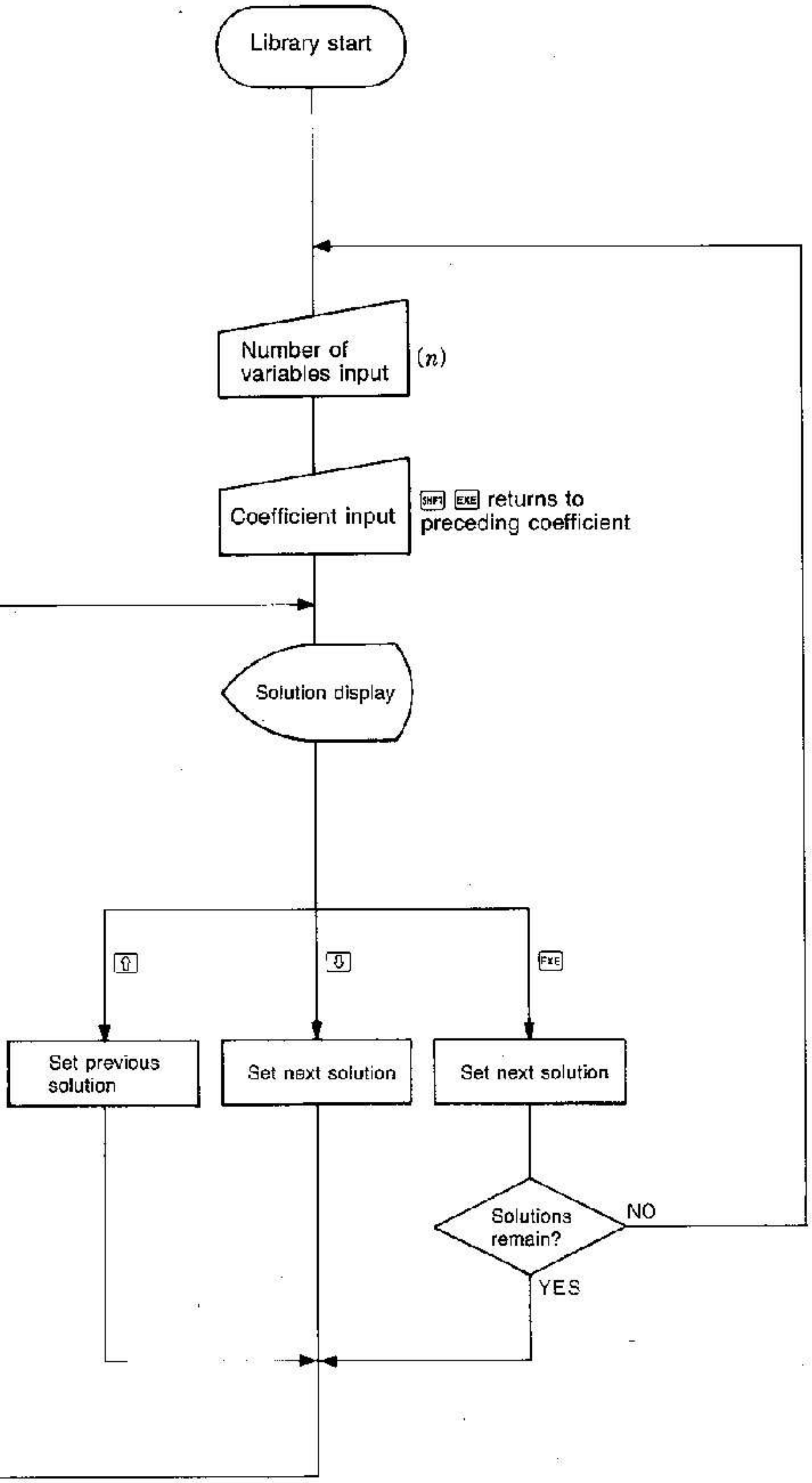
EXE

```
ax1+bx2+cx3+···=y        (2≦n≦7)
n= 3 ?_
```

(Return to initial display)

Here, the solutions of the simultaneous equations are $x_1 = 2$, $x_2 = 5$, $x_3 = 4$.

SIMULTANEOUS EQUATION FLOWCHART



B MEMORY CALCULATIONS

This function makes it possible to use the cursor keys to perform the four key memory (MC, MR, M –, M +) operations.

The following list shows the corresponding memory operation that corresponds to each key.

Hyp :	MC	(Memory Clear)	Clears data stored in memory
Sin :	MR	(Memory Recall)	Recalls data stored in memory
Cos :	M –	(Memory minus)	Subtracts from memory
Tan :	M +	(Memory plus)	Adds to memory

Besides the four basic arithmetic functions, numeric scientific function, logical operation, and comparison calculations can all be performed. One-key commands, however, cannot be used for numeric scientific function calculations, and direct function keys cannot be used.

EXAMPLE

[sin] [3] [0] [EXE] cannot be used to enter sin30°. It mused be entered as [S] [I] [N] [3] [0] [EXE].

The formula memory is used for memory calculations. Therefore, it should be noted that contents of the formula memory are changed when memory calculations are performed.

OPERATION

Fx B

```
[Memory Cal] M:0
0
```

EXAMPLE 1

Perform the calculation: 15÷3+7–6=6

```
[Memory Cal] M:0
0
```

15 [/] 3 [+] 7 [–] 6

```
[Memory Cal] M:0
15/3+7-6_
```

(Formula input)

[EXE]

```
[Memory Cal] M:6
 6
```

(Formula execution)

EXAMPLE 2

Perform the following calculations: 120 × 1.4 = 168
1.4 × 170 = 238

MC	[Memory Call] M: 0 0	(Memory clear)
1.4 M+	[Memory Call] M: 1.4 1.4	(Storage of 1.4 in memory)
120 * MR	[Memory Call] M: 1.4 120*1.4	(Formula input)
EXE	[Memory Call] M: 1.4 168	(Formula execution)
MR * 170	[Memory Call] M: 1.4 1.4*170	(Recall of 1.4 from memory)
EXE	[Memory Call] M: 1.4 238	(Formula execution)

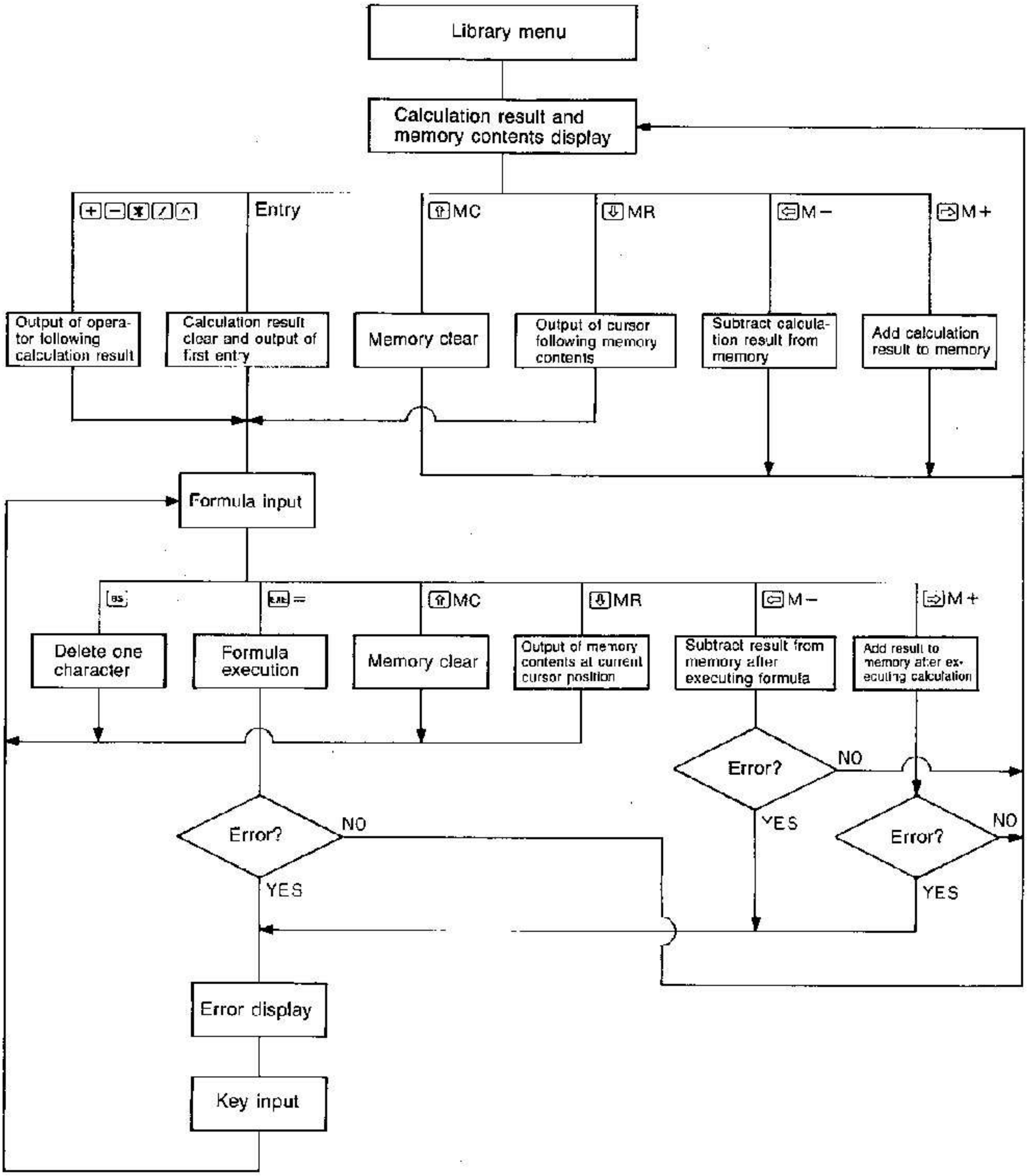
EXAMPLE 3

Perform the following calculation: 3 + 7 + sin 30° (angle unit = degree)

MC	[Memory Call] M: 0 0	(Memory clear)
3 M+	[Memory Call] M: 3 3	(Storage of value in memory)
7 M+	[Memory Call] M: 10 7	(Add to memory)
SIN30 M+	[Memory Call] M: 10.5 0.5	(Add to memory following function calculation)

Set the mode for the desired angle unit (DEG, RAD, GRA) before activating the library.

MEMORY CALCULATION FLOWCHART



CASIO

CASIO COMPUTER CO., LTD.
6-2, Hon-machi 1-chome
Shibuya-ku, Tokyo 151-8543, Japan

US.Pats. 4,120,583 4,121,284 4,371,923 4,396,976
4,398,263 4,410,956 4,531,182 4,686,622
4,829,419 4,942,516

MO9802-008149C 英 Printed in Japan